An Exact Method for the Minimum Feedback Arc Set Problem

ALI BAHAREV, HERMANN SCHICHL, and ARNOLD NEUMAIER,
University of Vienna
TOBIAS ACHTERBERG, Gurobi GmbH

A feedback arc set of a directed graph G is a subset of its arcs containing at least one arc of every cycle in G. Finding a feedback arc set of minimum cardinality is an NP-hard problem called the *minimum feedback arc set problem*. Numerically, the minimum set cover formulation of the minimum feedback arc set problem is appropriate as long as all simple cycles in G can be enumerated. Unfortunately, even those sparse graphs that are important for practical applications often have $\Omega(2^n)$ simple cycles. Here we address precisely such situations: An exact method is proposed for sparse graphs that enumerates simple cycles in a lazy fashion and iteratively extends an incomplete cycle matrix. In all cases encountered so far, only a tractable number of cycles has to be enumerated until a minimum feedback arc set is found. The practical limits of the new method are evaluated on a test set containing computationally challenging sparse graphs, relevant for industrial applications. The 4,468 test graphs are of varying size and density and suitable for testing the scalability of exact algorithms over a wide range.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**; *Integer programming*; • **Mathematics of computing → Paths and connectivity problems**; **Combinatorial optimization**; **Graph algorithms**; *Discrete optimization*;

Additional Key Words and Phrases: Linear ordering problem, maximum acyclic subgraph, minimum feedback arc set, minimum feedback vertex set, tearing

ACM Reference format:
Ali Baharev, Hermann Schichl, Arnold Neumaier, and Tobias Achterberg. 2021. An Exact Method for the Minimum Feedback Arc Set Problem. *J. Exp. Algorithmics* 26, 1, Article 1.4 (March 2021), 28 pages.
https://doi.org/10.1145/3446429

1 INTRODUCTION

Before summarizing the computational complexity results and several applications of the minimum feedback arc set problem, we first give definitions that make it possible to describe the problem itself and other well-known related problems. In the following, we only consider graphs

The research was funded by the Austrian Science Fund (FWF): P27891-N32. Support was also provided by the Austrian Research Promotion Agency (FFG) under project number 846920.
Authors' address: A. Baharev (corresponding author), H. Schichl, and A. Neumaier, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Wien, Austria; emails: ali.baharev@gmail.com, {Hermann.Schichl, arnold.neumaier}@univie.ac.at; T. Achterberg, Gurobi GmbH, Gurobi GmbH, Ulmenstraße 37-39, 60325 Frankfurt am Main; email: achterberg@gurobi.com.

1084-6654/2021/03-ART1.4 $15.00
https://doi.org/10.1145/3446429

without self-loops or multiple edges. Thus, in this work, a *directed graph* G is a pair (V, E) consisting of a set V of *vertices* (or nodes) and a set E of *arcs* (or directed edges), ordered pairs (u, v) with distinct $u, v \in V$. (An arc that connects a vertex to itself is a self-loop; we only consider graphs without self-loops.) The number of vertices is denoted by $n = |V|$ and the number of arcs by $m = |E|$. $H(V', E')$ is subgraph of the graph G if $V' \subseteq V$ and $E' \subseteq E$. It is said to be *induced* if E' consists of all of the edges in E that have both endpoints in V'. A directed *path* of length $s \geq 1$ from vertex v to w is a sequence $v = v_0, \ldots, v_s = w$ of vertices such that $(v_{i-1}, v_i) \in E$ for $i = 1, \ldots, s$; if $v = w$, it is called a *cycle*. The arcs (v_{i-1}, v_i) and (v_i, v_{i+1}) are called *consecutive arcs of the path*. A *simple cycle* is a cycle with no repeating arcs and no repeating nodes. Two simple cycles are *distinct* if one is not a cyclic permutation of the other. Throughout this article, whenever simple cycles are mentioned, distinct simple cycles are meant.

A *topological order* of G is a linear ordering of all of its nodes such that if G contains an arc (u, v), then u appears before v in the ordering. The nodes in a directed graph can be arranged in a topological order if and only if the directed graph is *acyclic* [29, Sec. 14.8]. The *topological sort algorithm* of Cormen et al. [23, Sec. 22.4] runs in time $\Theta(n + m)$; it is a simple and asymptotically optimal algorithm for checking whether a directed graph is acyclic.

A strongly connected component (SCC) of a directed graph $G = (V, E)$ is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices u and v in C, there is a directed path both from u to v and from v to u (u and v are reachable from each other). The SCCs of a directed graph can be found in linear time—that is, in $\Theta(n + m)$ time (see [110] and [23, Sec. 22.5]); these algorithms are asymptotically optimal. A *trivial SCC* consists of a single node. A trivial SCC must be acyclic, since we assume that G has no self-loops.

A *tournament* is a directed graph without self-loops such that for every two distinct nodes u and v there is exactly one arc with end-nodes u and v.

A *feedback vertex set* of a simple directed graph G is a set of vertices whose removal together with all connected arcs makes G acyclic; a feedback vertex set contains at least one vertex of every cycle in G. The term *feedback vertex set* also appears as *essential set* in the literature. A feedback vertex set S is *minimal* if no proper subset of S is a feedback vertex set.

A *feedback arc set* is a subset of arcs containing at least one arc of every cycle in a directed graph. In other words, removing the arcs in the feedback arc set from the graph makes the remaining graph a directed acyclic graph. A feedback arc set S is *minimal* if reinsertion of any arc $s \in S$ to the directed acyclic graph induces a cycle. If the arcs in a minimal feedback arc set are reversed rather than removed from the original graph, then the graph also becomes acyclic.

1.1 Computational Complexity

Given a directed graph G and an integer parameter k, the *(parameterized) feedback arc set problem* is to either construct a feedback arc set of at most k arcs for G, or to prove that no such arc set exists. This problem is called the *feedback arc set problem*, and it was item 8 on the list of Karp's 21 NP-complete problems [67]. A minimum feedback arc set can be computed in $O^*(2^n)$ time and $O^*(2^n)$ space, or in polynomial space and $O^*(4^{n+o(n)})$ time with dynamic programming [94]. The minimum feedback arc set problem is NP-hard for tournaments [3, 17]. An exact algorithm for tournaments with a running time of $O^*(1.5541^m)$ is available in the work of Raman et al. [95].

The feedback arc set problem is *fixed-parameter tractable* (FPT): an $O(n^4 4^k k^3 k!)$ time algorithm is given by Chen et al. [19]—that is, this algorithm runs in polynomial time if k is bounded above by a constant. The reader is referred to Dimian et al. [31] regarding further details on parameterized complexity and FPT. An exact algorithm was proposed by Hecht [54], which uses dynamic programming, and achieves $O(2^p m^4 \log(n))$ time, where $p \leq m - n + 1$ and p can be computed in $O(m^3)$ time.

The preceding definition is also referred to as the *unweighted feedback arc set problem*. In the *weighted feedback arc set problem*, each arc has its associated weight; the unweighted version can be regarded as the weighted version with each arc having unit weight. The *cost* refers either to the cardinality of the feedback arc set if the unweighted problem is solved or to the total weight of the feedback arc set if the weighted version of the problem is solved.

Finding a feedback arc set of minimum cardinality is the *minimum feedback arc set problem*. Similarly, finding a feedback vertex set of minimum cardinality is the *minimum (directed) feedback vertex set problem*. The reductions between the minimum feedback arc set problem and the minimum feedback vertex set problem preserve feasible solutions and their cost. In general, "these problems are equally hard to approximate in polynomial time" [36]. The fixed-parameter tractability result of Chen et al. [19] applies to the feedback vertex set problem with the same $O(n^4 4^k k^3 k!)$ time (see the work of Chen et al. [19] and the first paragraph of this section). The dynamic programming technique of Hecht [54] achieves $O(2^r d^4 n^4 \log(m))$ time, where $r \leq (d-1)n - m + 1$, d is the maximum degree of the graph, and r can be computed in $O(d^3 n^3)$ time.

Hereafter we focus on the minimum feedback arc set problem, but we wanted to indicate that results for the minimum feedback vertex set problem are also directly relevant.

The minimum feedback arc set problem is *APX-hard* [66]: Unless $P = NP$, the minimum feedback arc set problem does not have a polynomial-time approximation scheme (PTAS). The minimum feedback arc set problem is *approximation resistant*: Conditioned on the Unique Games Conjecture (UGC) [71], for every $C > 0$, it is NP-hard to find a C-approximation to the minimum feedback arc set problem (see Corollary 1.2. of Guruswami et al. [52]). One can construct a feedback arc set with cardinality of at most $m/2$ by taking either the forward or backward arcs (whichever has smaller cardinality) in an arbitrary ordering of the vertices of G. A better upper bound, $m/2 - n/6$, was derived by Eades et al. [34] by taking into account that arcs incident to sources or sinks cannot be part of a cycle; the algorithm runs in linear time and space. An $O(\log n \log\log n)$ approximation algorithm was implicitly described by Seymour [103] in the proof; the corresponding algorithm was explicitly given by Even et al. [36]. Simple combinatorial algorithms were presented by Demetrescu and Finocchi [28] that achieve an approximation ratio bounded by the length, in terms of number of arcs, of a longest simple cycle of the graph.

The minimum feedback arc set problem is solvable in polynomial time for planar graphs [78, 79], weakly acyclic graphs [45], and reducible flow graphs [93]. (Planar directed graphs are weakly acyclic.) A PTAS for minimum weighted feedback arc sets on tournaments is presented in the work of Kenyon-Mathieu and Schudy [70].

The complementary problem to the minimum feedback arc set problem is the *maximum acyclic subgraph problem*. The problem was proved to be APX-complete by Papadimitriou and Yannakakis [90]. The algorithm of Berger and Shor [11] finds an acyclic subgraph with $(1/2 + \Omega(1/\sqrt{d_{max}}))m$ arcs, where d_{max} is the maximum vertex degree in the graph; the algorithm runs in $O(mn)$ time. This lower bound on the number of arcs is sharp in the sense that an infinite class of directed graphs is exhibited in the work of Berger and Shor [11] realizing this bound. The previously cited algorithm of Eades et al. [34] provides an acyclic graph with at least $m/2 + n/6$ arcs and runs in $O(m)$ time. Note that in sparse graphs (i.e., $m = \Theta(n)$), this bound achieves the same asymptotic performance bound as the one in the work of Berger and Shor [11]. A PTAS (running in $n^{O(1/\epsilon^2)}$ time) was given by Arora et al. [5] for dense graphs (i.e., when $m = \Omega(n^2)$).

The more recent results regarding the maximum acyclic subgraph problem concern inapproximability. The best known approximation factor is $1/2 + \Omega(1/\log n)$ from Charikar et al. [18], which is a slight improvement over $1/2 + \Omega(1/(\log n \log\log n))$ that follows from other works [36, 103]. The problem is approximation resistant: Conditioned on the UGC, it is NP-hard to approximate the maximum acyclic subgraph problem within $1/2 + \epsilon$ for every $\epsilon > 0$ [51, 52]. Without assuming the

UGC and subject only to $P \neq NP$, the best known inapproximability result is $14/15 + \epsilon$, derived in the work of Austrin et al. [7].

The *linear ordering problem* can be defined as searching in a complete weighted directed graph for an acyclic tournament with a maximal sum of arc weights (see, e.g., the work of Marti and Reinelt [81] for further details). The maximum acyclic subgraph problem and the linear ordering problem "are in an obvious way polynomially related" [45]. The minimum feedback arc set problem is complementary to the maximum acyclic subgraph problem. The maximum acyclic subgraph problem and the linear ordering problem were extensively studied from a polyhedral point of view [44, 45], and it was shown by Grötschel et al. [45] that the maximum acyclic subgraph problem for weakly acyclic graphs can be solved in polynomial time. Insights into the facet structure of polytopes associated with these problems also lead to the formulation and implementation of a practical cutting plane algorithm for the linear ordering problem [43], which we detail in Section 3.3.

1.2 Applications

The minimum feedback arc set problem and the well-known problems it is linked to via polynomial time reductions have many real-world applications; we only summarize some of these applications here. Our primary interest in the minimum feedback arc set problem for sparse graphs comes from chemical engineering: It provides means to find favorable computation sequences in process flowsheet calculations through decomposition. We detail this particular application in Section 1.3. Some of our test problems are related to large-scale chemical engineering systems (see Section 5.4). The large-scale nature of those systems is no longer visible on the test graphs of Section 5.4 due to coarsening.

The linear ordering problem (also known as the permutation problem or triangulation problem) plays an important role in economics: Triangulated input-output matrices allow interesting interpretations of the structure of an economy and comparisons between different countries [43, 82]. This analysis is referred to as input-output analysis. The oldest application area of the linear ordering problem is ranking and rank aggregation [69, 104]. This includes ranking in sport tournaments, too [81, Ch. 1.2.7].

Solving the one-sided crossing minimization problem represents a fundamental step in the construction of aesthetically pleasing layouts of hierarchies and directed graphs, a problem also proved to be NP-complete. There is a strong relation between the one-sided crossing minimization problem and the minimum feedback arc set problem; in the work of Demetrescu and Finocchi [27], an approximation algorithm is devised that exploits this dependency. In general, solving the feedback arc set problem well is essential for good quality layered drawings of directed graphs [10, 109]. This community refers to the minimum feedback arc set problem as the penalty minimization method or penalty approach.

Further areas of applications include anthropology (computing a global ordering with as few contradictions as possible) [42], optimizing UMTS mobile phone telecommunication [12], deadlock recovery, and circuit testing [19]. This list is nonexhaustive; we refer to Marti and Reinelt [81, Ch. 1.2] for a comprehensive review of applications.

1.3 Connection to Tearing in Chemical Engineering

We define the task of *tearing* as follows. Given a bipartite graph B, we first orient it—that is, we assign a direction to each edge so that B becomes a directed graph D. Then, we compute the minimum feedback arc set F of D. *In our terminology*, the task of tearing is to find an orientation such that the cardinality of F is minimal among all possible orientations of B. If the edges of B are weighted, then the total weight of F should be minimal and not its cardinality. It is obvious that tearing and the minimum feedback arc set problem are related, although they are not equivalent problems.

Unfortunately, the term *tearing* is used in three different ways in the chemical engineering literature: It is sometimes used (a) exclusively for the (weighted) minimum feedback arc set problem (e.g., [9, 30, 40, 48, 73, 87, 92, 96, 112, 114, 117] and [13, Ch. 8]), (b) for both the minimum feedback arc set problem and for tearing as in our terminology as defined earlier (see, e.g., [56, 80, 86, 100]), and (c) primarily in our sense (e.g., [14, 20, 49, 55, 72, 106–108, 115]). This issue seems to be specific to the chemical engineering literature: For example, in the electrical engineering literature, tearing is generally used in our sense.

The reason the (weighted) minimum feedback arc set problem has received considerable attention in the field of chemical engineering is that it provides means to find favorable computation sequences in process flowsheet calculations. These computation sequences are referred to as the sequential-modular approach, and they can be faster to evaluate than solving the whole model simultaneously (equation-oriented approach). The sequential-modular approach can increase the robustness of the equation-oriented approach significantly: The steady-state solution found with the sequential-modular approach can be used for initializing equation-oriented models (see, e.g., [6]).

Tearing dates back to the 1930s [75, 111] and has been widely adapted across many engineering fields since: State-of-the-art steady-state and dynamic simulation environments all implement some variant of tearing, such as Aspen Aspen Technology, Inc. [6], MOSAICmodeling [16], Dymola [26], JModelica [85], and OpenModelica [88]. The applicability of tearing is not limited to a particular engineering discipline: It is generic, and it is used in all state-of-the-art Modelica simulators to model "complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents" [84].

2 HEURISTICS

We discuss heuristics for two reasons. In general, branch and bound algorithms tend to achieve better performance when supplied with (good) feasible solutions early in their search procedure. This is also the case for the proposed method, and we use a greedy local heuristic, discussed in the following. In practical applications, a good feasible solution may also be acceptable, and having a provably optimal solution is not necessarily a requirement. Heuristics are useful in such cases as well.

The literature on the various heuristics for the minimum feedback arc set, minimum feedback vertex set, maximum acyclic subgraph, and the linear ordering problem is overwhelming. Only a few of the published heuristics are presented here, since a proper review of them would require a monograph. Apart from the tractable special cases (e.g., planar graphs, reducible flow graphs), all known heuristics must obey the fact that the minimum feedback arc set problem is approximation resistant.

The minimum set cover problem approach. The greedy heuristic of Lee and Rudd [73] tends to give good results in our numerical experience if enumerating all simple cycles happens to be tractable for the input graph; for an enumeration algorithm, see the work of Johnson [62]. We gradually build the feedback arc set by always picking that arc as the next element that, when removed, destroys the most of the remaining simple cycles. Ties are broken arbitrarily. This heuristic (i.e., pick that arc that breaks the most cycles) is a well-known greedy heuristic for the minimum set cover problem, with an $O(1 + \log d)$ approximation factor guarantee, where d is the maximum cardinality of any subset [22, 63, 77, 98]. A simple example that makes this greedy heuristic fail was given by Ispolatov and Maslov [60]. Simplification rules can be applied to reduce the graph in each iteration step, before removing an arc or arcs (see [39, 73, 92], [13, p. 279], or [31, p. 114]). Sophisticated tie-breaking rules are also proposed by Lee and Rudd [73]. The weakness of this heuristic is that it requires all simple cycles to be enumerated, but just counting all simple cycles is already

#P-complete (see [113] and [4, Ch. 9]). Even *sparse* graphs can have $\Omega(2^n)$ simple cycles [102], and such graphs appear in practice (see Section 5.6).

Greedy local heuristics. Other heuristics that do not require enumerating all simple cycles are often based on local information only and make greedy choices. (By local information, we mean, e.g., that only the in-degree and out-degree of the individual nodes are taken into account but not global properties of the input graph.) A common pattern in these greedy heuristics is described in the following.

The feedback arc set is built up iteratively. The input graph is simplified in each step before removing an arc or arcs; this simplification can include splitting into SCCs, then dropping the trivial SCCs (a trivial SCC consists of a single node), breaking two-cycles appropriately, and so forth. Further simplification examples with figures are given in Appendix A.3. After the simplification, the algorithm looks for a node in the remaining graph where many simple cycles are likely to be destroyed when one or a few arcs of that node are removed. For example, a node in an SCC with a single in-arc but with many out-arcs is a good candidate: Removing its single in-arc breaks all cycles that pass through that node, and the number of destroyed simple cycles is at least the out-degree of that node (each out-arc must participate in at least one simple cycle in an SCC by definition). This is the intuition behind the greedy score functions: A node gets a higher score if it is more "asymmetric" regarding its in- and out-degrees. Such score functions are, for example,

$$score(i) = |d_i^{in} - d_i^{out}| \tag{1}$$

and

$$score(i) = \max\left(\frac{d_i^{in}}{d_i^{out}}, \frac{d_i^{out}}{d_i^{in}}\right), \tag{2}$$

where $score(i)$ is the score of node i; d_i^{in} and d_i^{out} are the in- and out-degree of node i, respectively. (The weighted variants of these score functions can be used if the input is a weighted graph.) The node with the highest score is selected (breaking ties arbitrarily). If the selected node has more in-arcs than out-arcs, all of the out-arcs of this node are removed; otherwise, all of the in-arcs are removed. The algorithm continues with the simplification. The heuristic terminates when there are no arcs left. This pattern can be recognized, for example, in other works [33, 34, 48, 99], but this list is by no means complete.

Sorting heuristics. Given an arbitrary ordering of the nodes of G, one can unambiguously categorize all of the arcs as either forward or backward arcs depending on whether the terminal node of the arc (head) appears after the initial node (tail) of the same arc or before. In the former case, the arc is a forward arc (it is pointing forward in the ordering); in the latter case, it is a backward arc. We select the set of backward arcs as the feedback arc set.

The sorting heuristics view the minimum feedback arc set problem as an ordering problem: They try to find the minimum cost ordering by sorting the nodes appropriately. Various sorting heuristics have been reviewed and new ones have been proposed by Brandenburg and Hanauer [15]. Numerical results are reported on both sparse and dense random graphs where n ranges from 100 to 1,000, and also for tournaments that have been reported to trigger particularly poor performance for certain heuristics. The authors also report promising results for their novel hybrid sorting heuristics.

A heuristic based on depth-first search and local search. A heuristic that does not resemble any of the preceding ones is given by Park and Akers [91]. Beside the common simplifications (removing self-loops, sources, and sinks, then partitioning into SCCs), the SCCs are also partitioned into biconnected components at the articulation points. (A node v is an articulation point if the removal

of v causes the graph to become disconnected.) After these simplifications, a depth-first search (DFS) is performed on each component to identify a (hopefully large) acyclic subgraph D; the arcs not in D form a feedback arc set F. The cardinality of F is further reduced by a local search heuristic that works on consecutive subgraphs.

Heuristics for the closely related linear ordering problem. Finally, the reader is referred to the heuristics for the linear ordering problem, which are discussed in great detail in the work of Marti and Reinelt [81].

3 EXACT METHODS

The published exact methods include (a) dynamic programming (e.g., [101, 112]), (b) custom branch and bound methods (or smart enumeration with special exclusion rules; e.g., [37, 46, 64, 89, 92]), and (c) integer programming formulations. The latter will be reviewed in the following sections in detail, since the present work focuses on an approach based on integer programming.

3.1 Integer Programming Formulation with Triangle Inequalities

We seek a minimum cost ordering π^* of the nodes of $G = (V, E)$. Let $c_{i,j}$ denote the cost associated with the directed arc $(i, j) \in E$, and let $c_{i,j} = 0$ if $(i, j) \notin E$. If the cardinality of the feedback arc set is to be minimized, then for each $(i, j) \in E$ we have $c_{i,j} = 1$. If the weighted minimum feedback arc set problem is to be solved, then all $c_{i,j}$ associated with a directed arc equal the weight of the corresponding arc (i, j). Furthermore, for all $i, j \in V$, $i \neq j$, let the binary variables $y_{i,j}$ associated with a given ordering π encode the following: Let $y_{i,j} = 0$ if node i precedes j in π, and let $y_{i,j} = 1$ otherwise. Any ordering π uniquely determines a corresponding y. This results in the following integer programming formulation:

$$\begin{aligned} &\min_{y} \sum_{j=1}^{n} \left(\sum_{k=1}^{j-1} c_{k,j} y_{k,j} + \sum_{\ell=j+1}^{n} c_{\ell,j} (1 - y_{j,\ell}) \right) \\ &\text{subject to} \\ &\qquad y_{i,j} + y_{j,k} - y_{i,k} \leq 1, \quad 1 \leq i < j < k \leq n \\ &\qquad -y_{i,j} - y_{j,k} + y_{i,k} \leq 0, \quad 1 \leq i < j < k \leq n \\ &\qquad y_{i,j} = \{0, 1\}, \quad 1 \leq i < j \leq n. \end{aligned} \tag{3}$$

Any y that satisfies the *triangle inequalities* (3) must correspond to an ordering [43, 74, 83]. Note that there are $O(n^2)$ binary variables and $O(n^3)$ constraints in (3). Custom-tailored cutting plane algorithms have been developed to solve this integer program efficiently (and the linear ordering problem in general; see, e.g., [43, 83] and [81, Ch. 5], but also Section 3.3).

3.2 Integer Programming Formulation as Minimum Set Cover

An alternative to the formulation of Section 3.1 is the minimum set cover formulation (see, e.g., [92, Eq. (1)] or [13, Sec. 8.4]).

$$\begin{aligned} &\min_{y} \sum_{j=1}^{m} w_j y_j \\ &\text{s.t.} \sum_{j=1}^{m} a_{ij} y_j \geq 1 \qquad \text{for each } i = 1, 2, \ldots, \ell \\ &\quad y_j \text{ is binary} \end{aligned} \tag{4}$$

Here, m denotes the number of arcs; w_j are nonnegative weights (often integer); y_j is 1 if arc j is in the feedback arc set, and 0 otherwise; a_{ij} is 1 if arc j participates in cycle i, and 0

otherwise; and ℓ denotes the number of simple cycles. The matrix $A = (a_{ij})$ is called the *cycle matrix*.

In practice, the cycle matrix can often be significantly reduced in a pre-solve phase ([39, 73, 92], [13, p. 279], or [31, p. 114]—for example, by iteratively removing dominating rows and dominated columns of the cycle matrix, and by removing columns that intersect a row with a single nonzero entry). These simplifications were also referenced in the minimum set cover approach in Section 2 on heuristics. State-of-the-art integer programming solvers such as Gurobi [50] or SCIP [2] implement these simplifications (and other simplifications as well). After the pre-solve phase, further specialized methods are available for handling the set covering constraints of (4) efficiently in a branch and bound solver (see, e.g., [1]).

The weakness of this formulation has already been discussed in Section 2: even sparse graphs can have $\Omega(2^n)$ simple cycles [102], and such graphs appear in practice (see Section 5.6).

3.3 The Cutting Plane Algorithm of Grötschel et al.

In their seminal works, Grötschel et al. [44, 45] investigated the facet structure of the 0/1-polytope associated with the linear ordering problem and showed that the maximum acyclic subgraph problem for weakly acyclic graphs can be solved in polynomial time (see Section 1.1). The actual algorithm that is based on the theoretical results of Grötschel et al. [44, 45] appeared in their earlier work [43], and it is applicable to general directed graphs. This computer implementation is, of course, not guaranteed to run in polynomial time for general directed graphs. The authors provide numerical results. They also report issues that surface in practice—for example: "we do not even know how to describe all Möbious ladders in D_n in a 'handy way' " (p. 1201, last paragraph of Grötschel et al. [43]), or "we have only some simple heuristics for some subclasses of (5) and (6)" (p. 1202, first paragraph). Like the integer programming formulation with triangle inequalities of Section 3.1, the algorithm of Grötschel et al. [43] was designed for dense graphs. As a consequence, both algorithms have the same $O(n^3)$ scalability issue: "The 3-dicycle inequalities of (4) are handled by brute-force. We enumerate all $2\binom{n}{3}$ 3-dicycles and find those that are violated by x^*" (p. 1205, last paragraph). When applied to large and sparse graphs, this becomes a severe performance issue (see Section 5.6).

We have no doubt that the preceding quoted issues can be addressed, but this is outside the scope of the present work. Nevertheless, it is our hope that future researchers, targeting such novel cutting plane algorithms, will find the present work and its benchmark problems a useful baseline. See also Section 4.4 on how the proposed method differs from cutting plane algorithms.

4 AN INTEGER PROGRAMMING APPROACH WITH LAZY CONSTRAINT GENERATION

The traditional set covering formulation is used in our implementation; the reader is referred back to Section 3.2 regarding the notation. If enumerating all simple cycles of G happens to be tractable (see the work of Johnson [62] for enumerating all simple cycles), the integer program (4) with the complete cycle matrix A can be fed to a general-purpose integer programming solver such as Gurobi [50] or SCIP [2]. These state-of-the-art integer programming solvers usually do not have any difficulty solving (4) to optimality in reasonable time, even with 10^5 cycles in A. In practice, the real challenge is enumerating all simple cycles: It is often intractable in practice, and the proposed method addresses exactly such situations.

4.1 Informal Overview of the Proposed Method

The proposed method enumerates simple cycles in a lazy fashion and extends an incomplete cycle matrix iteratively. Let us refer to problem (4) with the complete cycle matrix as P, and let $\tilde{P}^{(k)}$

denote its relaxation in iteration k where only a subset of simple cycles is included in the incomplete cycle matrix $A^{(k)}$. The first cycle matrix $A^{(1)}$ can be initialized as follows. We invoke the algorithm for extending the cycle matrix, Algorithm 2, with *all* of the arcs of G as feedback arc set, and with an empty cycle matrix; the algorithm returns the first cycle matrix for $\tilde{P}^{(1)}$. (Other initialization procedures are also possible.) Internally, Algorithm 2 tries to find a simple cycle with breadth-first search (BFS) for each arc it was given and appends the newly found cycles to the cycle matrix.

In iteration k, the optimal solution to the relaxed problem $\tilde{P}^{(k)}$ gives a feedback arc set, and we remove all arcs in this feedback arc set from G to get $G^{(k)}$. Since not all simple cycles are included in the cycle matrix $A^{(k)}$ (only a relaxation is solved), $G^{(k)}$ is not necessarily acyclic. Therefore, we need to check acyclicity: Topological sort succeeds if and only if $G^{(k)}$ is acyclic. If the topological sort succeeds, the algorithm has found an optimal solution to P and the algorithm terminates.

If the topological sort on $G^{(k)}$ fails, then $G^{(k)}$ must have cycles. In this case, we first create a feasible solution to P as follows. We identify a feedback arc set $F^{(k)}$ of $G^{(k)}$ using an appropriate heuristic (see Section 2). The proposed algorithm is guaranteed to make progress with *any* feedback arc set, but the algorithm is likely to make better progress with an $F^{(k)}$ of small cardinality. Removing the arcs in $F^{(k)}$ makes $G^{(k)}$ acyclic, and therefore the associated y yields a feasible solution to P, cf. (4). We keep track of the best feasible solution to P found (incumbent solution).

After we have created a feasible solution to P, we improve the relaxation $\tilde{P}^{(k)}$ by adding new rows to the cycle matrix $A^{(k)}$. The directed graph $G^{(k)}$ must have at least one cycle because topological sort failed previously. The feedback arc set $F^{(k)}$ contains at least one arc of every cycle in $G^{(k)}$ by definition; therefore, there must be at least one arc $e \in F^{(k)}$ that participates in a cycle. For each arc $e \in F^{(k)}$, we compute the shortest path from the head of e to the tail of e with BFS. (Although it has not been observed, this procedure based on BFS can potentially lead to poor performance if the arc weight distribution is pathological. It is subject to future research to improve this procedure in such pathological and not yet seen cases; probably neither Dijkstra's shortest path algorithm nor BFS is the best algorithm one can do, and some other algorithm should be developed for such difficult graphs.) Such a shortest path exists if and only if e participates in a cycle; we extended this shortest path with e which then gives a simple cycle (even without chords). A new row is appended to the cycle matrix for each simple cycle found. The cycle matrix $A^{(k)}$ is guaranteed to grow at least by one row by the time we finish processing all arcs in $F^{(k)}$. We then proceed with the next iteration step, starting with solving the next relaxed problem $\tilde{P}^{(k+1)}$ with this extended cycle matrix $A^{(k+1)}$.

The algorithm terminates if $G^{(k)}$ is acyclic (as already discussed) or the objective at the optimal solution of a relaxed problem equals the objective at the best known feasible solution to P. A minimum feedback arc set has been found in both terminating cases. Finite termination is guaranteed: The cycle matrix must grow by at least one row in each iteration, and there is only a finite number of simple cycles in the graph.

On the implementation level, the Gurobi [50] parameter `LazyConstraints` is set to 1, and adding new rows to the cycle matrix is implemented in a callback function. For further details, the reader is referred to the reference manual of Gurobi and to the source code on GitHub [8].

The cycle matrix is only extended as the algorithm runs; rows are never removed from it. A theoretical worst-case scenario for the proposed method is that the accumulating rows make the cycle matrix intractable. Our test set has 4,468 graphs, and we have not encountered a single graph that would trigger this worst-case behavior. The largest cycle matrix was the one with 2,282 rows, and was produced for the random sparse graph with parameters $n = 100$, $c = 7$, $seed = 43$. Note that the integer programming solver Gurobi (a) maintains a lazy constraint pool internally and removes superfluous rows that we added from that internal pool, and (b) automatically generates

ALGORITHM 1: Finding a minimum feedback arc set based on integer programming and lazy constraint generation

Input: G, a directed graph with m arcs and nonnegative arc weights w_j $(j = 1, 2, \ldots, m)$
Output: A minimum weight feedback arc set
 # P denotes the integer program (4) with the complete cycle matrix of G
Let $\hat{y}$ denote the best feasible solution to P found at any point during the search (incumbent solution)
Compute a feedback arc set $F^{(0)}$ of G using, e.g., any of the heuristics cited in Section 2
Set the solution associated with $F^{(0)}$ as the incumbent $\hat{y}$
Set the lower bound $\underline{z}$ and the upper bound $\bar{z}$ on the objective to 0 and $\sum w_j \hat{y}_j$, respectively
Let $A^{(i)}$ denote the incomplete cycle matrix in (4), giving the relaxed problem $\tilde{P}^{(i)}$ $(i = 1, 2 \ldots)$
call Algorithm 2 with G, $F^{(0)}$, and an empty cycle matrix to get the first cycle matrix $A^{(1)}$
for $i = 1, 2, \ldots$ **do**
    Solve the relaxed problem $\tilde{P}^{(i)}$; results: solution $y^{(i)}$, the associated feedback arc set S and objective value $z^{(i)}$
    # Optional: When the integer programming solver is invoked on the line just above,
    # $\hat{y}$ can be used as a starting point
    Set the lower bound $\underline{z}$ to $\max(\underline{z}, z^{(i)})$
    if $\underline{z}$ *equals* $\bar{z}$ **then**
        stop, $\hat{y}$ is optimal
    Let $G^{(i)}$ denote the graph obtained by removing all arcs of S from G
    if $G^{(i)}$ *can be topologically sorted* **then**
        stop, $y^{(i)}$ is the optimal solution to P as well
    Compute a feedback arc set $F^{(i)}$ of $G^{(i)}$ using, e.g., any of the heuristics cited in Section 2
    Set those components of $y^{(i)}$ to 1 that correspond to an arc in $F^{(i)}$
    # $y^{(i)}$ is now a feasible solution to P
    Let $\hat{z}$ be the new objective value at $y^{(i)}$
    if $\hat{z} < \bar{z}$ **then**
        Set $\bar{z}$ to $\hat{z}$
        Set $\hat{y}$ to $y^{(i)}$
    call Algorithm 2 with $G^{(i)}$, $F^{(i)}$, and $A^{(i)}$ to get the extended cycle matrix $A^{(i+1)}$
    # $A^{(i+1)}$ is guaranteed to have at least one additional row compared to $A^{(i)}$

additional cutting planes independent of our lazy constraints and maintains that cut pool internally as well.

4.2 Pseudo-Code of the Proposed Algorithm

The pseudo-code of the algorithm is given as Algorithm 1 and at Algorithm 2; the Python implementation is available from GitHub [8].

4.3 Novelties

The idea of building up an integer program incrementally, by adding constraints to it in a lazy fashion, dates back at least to 1954 [25]. The well-known column generation approach corresponds to this idea but works on the dual problem. Probably the first published work applying column generation is from 1958 [38]. Not surprisingly, state-of-the-art integer programming solvers have high-level API support for implementing such algorithms (see, e.g., `LazyConstraints` in Gurobi [50]).

ALGORITHM 2: Extending the cycle matrix given an arbitrary feedback arc set

Input: G, a directed graph; F, a feedback arc set of G; the incomplete cycle matrix A
Output: The extended cycle matrix A

```
foreach e ∈ F do
    Find a shortest path p from the head of e to the tail of e with BFS in G
    if such a path p exists then
        Turn the path p into a simple cycle s by adding the arc e to p
        Add a new row r to the cycle matrix corresponding to s if r is not already in the matrix
```

We are not aware of any published algorithm that would apply lazy constraint generation in the context of the minimum feedback arc set problem. However, the real novelty is in *how* the lazy constraint generation is carried out: (a) We apply a greedy heuristic to find a feedback arc set, (b) we then find the tightest simple cycles with BFS that contain this feedback arc set, (c) and finally we extend the cycle matrix with these simple cycles found—that is, append them as new constraints. As the numerical evidence of the next section suggests, this is an efficient approach on sparse graphs.

4.4 Lazy Constraints Compared to Cutting Planes

State-of-the-art integer solvers like Gurobi [50] have sophisticated algorithms for generating cutting planes. Any research attempting to implement a novel cutting plane algorithm has to compete with that strong baseline. The proposed method of Sections 4.1 and 4.2 is not a cutting plane algorithm but a lazy constraint generation algorithm: Lazy constraints are added to *ensure correctness* when solving an incomplete linear program, whereas cutting planes are applied in an attempt to *improve performance* but the linear program would be correct and complete without the cutting planes, too. The proposed method appends new constraints whenever the (partial) integer program (as seen by the solver) is solved to optimality but the optimal solution is not feasible to the original input problem. In terms of Gurobi's implementation, a lazy constraint generation algorithm uses `LazyConstraints` and is implemented as a callback for `MIPSOL`. In general, this callback function is typically invoked far less often than there are nodes in the branch and bound search tree. However, a custom cutting plane algorithm can add new cuts at each node of the branch and bound search tree (in practice, cuts should be added sparingly). Cutting planes are typically used to cut off the current relaxation solution. In terms of Gurobi's implementation, it is implemented as a callback for `MIPNODE`, and the callback function is invoked whenever a node is being explored. Therefore, a lazy constraint generation algorithm does not compete with cutting plane algorithms but augments them: The proposed algorithm of Sections 4.1 and 4.2 leverages all 17 state-of-the-art cutting plane algorithms that Gurobi implements. Although we have no doubt that one could implement a novel cutting plane algorithm specifically for the minimum feedback arc set problem that outperforms the implementation of Gurobi, such research is outside the scope of the present work.

5 COMPUTATIONAL RESULTS

We first cross checked correctness of the implementation on sparse graphs from the field of chemical engineering (Section 5.4). The proposed method was designed for sparse graphs, so we studied its scalability on sparse graphs (Sections 5.5 and 5.6) and compared that to the method of Section 3.1. Dense graphs trigger, in some sense, the worst-case behavior of the proposed method since it was designed for sparse graphs, whereas the method of Section 3.1 was designed for the dense case; we give comparisons between these two approaches on tournaments in Section 5.7.

Solving the entire benchmark suite of 4,468 test graphs pushes the limits of the state of the art: It takes more than 1 CPU-month on a modern desktop machine just for the proposed method alone, and as we will argue in Section 5.3, many of these graphs are intractable for the other methods. (This is also the reason we did not recompute the results of Sections 5.5 and 5.7 with Setting B of Section 5.2.)

The benchmark problems must be diverse enough to enable testing the scalability of various algorithms with respect to the size and sparsity of the input graphs. As a consequence, some of the benchmark graphs can be solved without difficulty with today's computational power. The reason for including them nevertheless is that we want to plot the lower end of the scalability curve as well to see the trend better; certain comparisons would have been impossible without these easier graphs. Finally, easier graphs are indispensable for cross checking correctness of algorithms, and a comprehensive collection of test graphs should also support that need.

5.1 Pre-Solve Phase

In the pre-solve phase, we attempt to generate an equivalent but simpler graph than the input. From those pre-solve techniques that are discussed in the appendix, only the following procedures were applied here in Section 5: splitting into nontrivial SCCs, then iteratively removing runs and 3-arc bypasses (see *Hand-coded procedures for common patterns* in Appendix A.3 and also Figure 4). The reason is that our pure Python implementation for pre-processing is not competitive with Gurobi's pre-processing implemented in C; one of the reasons is that Python is an interpreted language, whereas C compiles to machine code. Apart from the graphs for cross checking correctness (see Section 5.4), our simple pre-processing made very little difference, if any. All of the tested algorithms were started from the same simplified graphs to ensure fair comparisons.

5.2 Hardware and Software Environment

The computations were carried out in two different settings:

Setting A: Processor: Intel Core i5-4670S CPU at 3.10 GHz; operating system: Ubuntu 14.04.3 LTS with 3.13.0-86-generic kernel; the state-of-the-art integer programming solver Gurobi 6.5.1 [50] was called through its API from Python 2.7.11; the graph library `NetworkX` [53] 1.9.1 was used.

Setting B (when revising the manuscript): Processor: 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. This setting was used for Section 5.6 and for the complete graph in Section 5.7; all other computations were carried out in Setting A.

5.3 The Limit of Tractability with the Minimum Set Cover Formulation

For the integer programming approach of Section 3.2, we consider a graph *intractable* if it has more than 10^7 simple cycles. To prove that a graph has more than 10^7 simple cycles, we enumerate $10^7 + 1$ of them with Johnson's algorithm [62] before giving up on enumerating all of them. Our firsthand experience shows that 10^7 simple cycles typically consume more memory than 4 GB, so even if we could enumerate all simple cycles of a graph that we consider intractable, we would have difficulties fitting the corresponding linear program into memory, let alone solving it to optimality.

5.4 Cross Checking Correctness

A benchmark of chemical engineering problems was given by Gundersen and Hertzberg [48]. It consists of coarsened graphs of large-scale chemical engineering systems. The large-scale nature of

Table 1. Properties of the Test Graphs for Cross Checking Correctness

ID	Nodes	Arcs	SCCs	Cycles	Optimum	Original Source
1	6	30	1	409	15	Complete graph
2	12	21	1	22	2	[92]
3	15	35	3	27	6	[9]
4	19	31	1	20	6	[101]
5	25	32	1	10	3	[21] (“first”)
6	29	37	1	11	5	[61] (HF-alkylation)
7	30	42	1	31	3	[21] (“second”)
8	41	61	1	103	5	Shannon (sulfuric acid; see [48])
9	50	79	1	22	8	[61] (vegetable oil)
10	109	163	1	13746	12	[47] (heavy water)
11	32	52	1	187	6	See Appendix A.4

These graphs once used to be a benchmark. The proposed method solves all of them immediately on the root node. The most time consuming is Problem 10; solving it requires 0.02 seconds.

the original systems is no longer visible on the test graphs due to the coarsening. This set primarily serves for cross checking the correctness of our implementation against both the published results and the integer programming approaches of Sections 3.1 and 3.2. The properties of these test graphs are given in Table 1.

The graphs were taken from the work of Gundersen and Hertzberg [48]: The test problems with ID=1..10 correspond to the problem with the same ID in their work [48]. (The self-loops were removed from Problem 1 of Gundersen and Hertzberg [48].) Problem 11 was obtained by running the integer linear programming–based arc removing algorithm of Appendix A.3 on Problem 10 with very aggressive settings, as discussed in Appendix A.4. The goal was to isolate the core of Problem 10 that makes this test graph inherently difficult. The corresponding graphs are shown in Figures 6 and 7.

For each problem in Table 1, the initial cycle matrix was already sufficient for the proposed method to prove that the solution found is optimal, meaning that line 15 of Algorithm 1 is not reached. In other words, all problems were solved immediately on the root node. The most time consuming is Problem 10; solving it requires 0.02 seconds. Although these graphs once used to be a benchmark, they can be solved without any significant difficulties with today’s computational power. The minimum feedback arc set problem is still important for many engineering applications (see the last paragraph of Section 1.3); however, our attempts at compiling an updated benchmark suite, similar to that of Gundersen and Hertzberg [48], failed.

5.5 Sparse Random Graphs

In the $G(n, p)$ Erdős–Rényi random graph model, each arc of an order-n graph is included with probability p independently from every other arc [35]. This model was introduced independently by Gilbert [41]. The Erdős–Rényi random graph model plays an important role in the probabilistic method to prove the existence of graphs satisfying various properties, or to provide a rigorous definition of what it means for a property to hold for almost all graphs.

The computational results for 3,737 random graphs are shown in Figure 1. For a fixed (n, c), the median execution time of the proposed method is consistently less than that of the method of Section 3.1. As expected, the median execution time increases for both methods as the graph becomes denser (i.e., increases with $c = pn$).

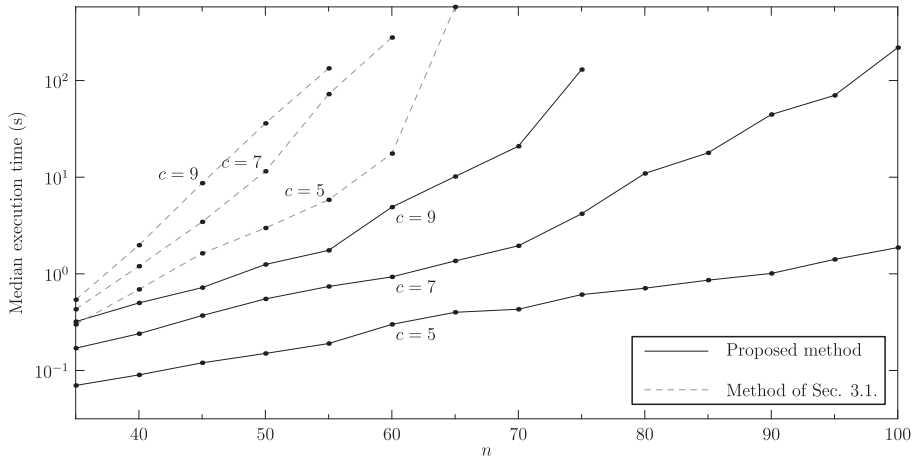

Fig. 1. Computing the minimum feedback arc set of 3,737 random graphs, generated according to the Erdős–Rényi model $G(n, p)$ with arc probability $p = \frac{c}{n}$, where the number of nodes n and the parameter c are shown in the figure. Each dot represents the median of the execution times over 101 random graphs. Solid lines: proposed method; dashed lines: integer programming formulation of Section 3.1 using triangle inequalities.

5.6 Highly Structured Sparse Graphs

The *generalized de Bruijn graph* $B(n, d)$ is defined as follows [32]. The nodes are labeled $0, 1, \ldots, n-1$, and the directed arc set consists of

$$u \to u \cdot d + r \mod n \quad \text{for each } 0 \le u \le n-1,\ 0 \le r \le d-1, \tag{5}$$

where d is the degree of the graph. The generalized de Bruijn graphs were first proposed independently in the work of Pradhan et al. [97] and Imase and Itoh [57]. Certain grid network topologies are de Bruijn graphs [105, Secs. 5.6.1.2 and 5.6.2.2]. The distributed hash table protocol Koorde uses de Bruijn graphs [65]. In bioinformatics, de Bruijn graphs are used for de novo assembly of (short) read sequences into a genome [116]. Table 2 shows the computational results.

Kautz graphs are closely related to de Bruijn graphs. The directed Kautz graph $K(d, k)$ of degree d and diameter k is the graph defined as follows [68]. A node is labeled with a word of length k, $(x_1, \ldots, x_k)$, on the alphabet $\Sigma = \{0, \ldots, d\}$, $|\Sigma| = d + 1$, in which $x_i \neq x_{i+1}$ for $1 \le i \le k-1$. There is an arc from a node $x = (x_1, \ldots, x_k)$ to all vertices y such that $y = (x_2, \ldots, x_k, z)$, $z \in \Sigma$, $z \neq x_k$. The Kautz graph is a good static topology to construct distributed hash table schemes [76], for fault-tolerant processor interconnection networks [59], and for multi-OPS optical network topologies [24].

The definition of Kautz graphs cannot yield graphs of any size. Graphs by Imase and Itoh are a generalization of Kautz graphs to obtain graphs of arbitrary size. The directed *graphs of Imase and Itoh* of degree d and order n are defined as follows [58]: The nodes are labeled $0, 1, \ldots, n-1$, and there is a directed arc from node i to node j if and only if

$$j \equiv i \cdot d + \alpha \mod n,\ \alpha = 0, \ldots, d-1. \tag{6}$$

The directed Kautz graph of degree d and diameter k is isomorphic to the directed graph of Imase and Itoh of degree d and order $d^{k-1}(d+1)$. The Imase and Itoh graphs have applications in the design of building-block switching systems, communication networks, and distributed computer systems [58]. Table 3 shows computational results on graphs of Imase and Itoh.

Table 2. Computational Results on de Bruijn Graphs with a Time Limit of 3 Hours (10,800 seconds); the Execution Times t_{PM} Belong to the Proposed Method and t_{TI} to the Method of Section 3.1 with Triangle Inequalities

Nodes	Arcs	Parameter d	Optimum	t_{PM} (s)	t_{TI} (s)
100	296	3	58	37.95	Timeout
100	396	4	91	2.07	Timeout
100	492	5	116	1.79	378.73
100	590	6	158	31.03	Timeout
110	326	3	63	1.90	Timeout
110	436	4	97	332.61	Timeout
110	544	5	134	39.95	798.23
110	650	6	172	10,606.27	Timeout
120	356	3	66	1.98	951.32
120	474	4	108	6.53	Timeout
120	592	5	150	1.95	1,250.45
120	710	6	180	6,699.13	Timeout

These graphs are considered intractable with the method of Section 3.2 (see Section 5.3); exploiting fixed-parameter tractability is not an option either. Except for the parameter d, all data are given for the graphs after the pre-solve phase (i.e., after removing self-loops).

Table 3. Computational Results on Graphs of Imase and Itoh with a Time Limit of 3 Hours (10,800 seconds); the Execution Times t_{PM} Belong to the Proposed Method and t_{TI} to the Method of Section 3.1 with Triangle Inequalities

Nodes	Arcs	Parameter d	Optimum	t_{PM} (s)	t_{TI} (s)
100	300	3	66	0.78	208.04
100	400	4	90	1.38	817.93
100	496	5	126	1.42	671.37
100	594	6	156	32.60	Timeout
100	696	7	192	25.89	1,274.77
110	328	3	62	3.08	Timeout
110	440	4	100	5.93	Timeout
110	546	5	135	7.14	1,283.38
110	654	6	172	59.04	Timeout
110	764	7	210	6,821.63	Timeout
120	360	3	72	11.98	2,052.25
120	480	4	114	Timeout	Timeout

These graphs are considered intractable with the method of Section 3.2 (see Section 5.3); exploiting fixed-parameter tractability is not an option either. Except for the parameter d, all data are given for the graphs after the pre-solve phase (i.e., after removing self-loops).

5.6.1 Interpretation of the Results. The proposed method used to succeed within 3 hours in all cases in Setting A (see Section 5.2); however, with the newer version of Gurobi in Setting B, we get a timeout in one case. When the time limit was reached, the best feasible solution was an optimal one, but the best bound was still off by 1. In all other cases, the proposed method was consistently and significantly faster than the method of Section 3.1.

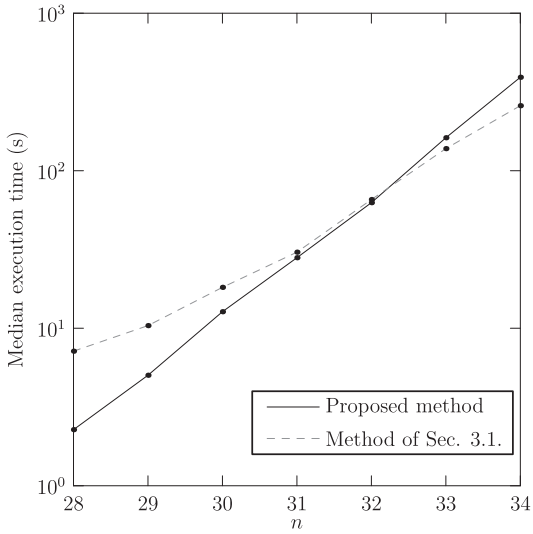

Fig. 2. Computing the minimum feedback arc set of 707 random tournaments. Each dot represents the median of the execution times over 101 random tournaments. The number of nodes is denoted by n. Solid line: proposed method; dashed line: integer programming formulation of Section 3.1 using triangle inequalities.

In contrast, in those cases where the method of Section 3.1 was able to prove optimality within 3 hours, an optimal solution was already found on the root node. In all other cases where the method of Section 3.1 failed to prove optimality within 3 hours, the best feasible solution was still a suboptimal one when the time limit was reached. This method struggles due to the size of the model: There are $O(n^2)$ binary variables and $O(n^3)$ constraints in (3). As a consequence, there was only one case where the solver managed to explore three nodes of the branch and bound search tree within 3 hours, and in all other cases it finished exploring even less (typically 0 or 1 node).

Finally, the sparse graphs considered in this section are intractable with the integer programming approach of Section 3.2 due to the sheer number of simple cycles: We already fail to fit the model into memory, let alone solving it to optimality (see Section 5.3). Exploiting fixed-parameter tractability, the $O(n^4 4^k k^3 k!)$ time algorithm of Chen et al. [19] (Section 1.1) is not an option either due to the size of the minimum feedback arc set (k is of order 10^2).

5.7 Dense Graphs for Testing the Worst-Case Behavior

Random tournaments. As discussed in Section 1.1, a tournament is an orientation of an undirected complete graph. Both a PTAS (see Section 1.1) and custom-tailored cutting plane algorithms have been developed to solve tournaments efficiently (see Section 3.1). The sole reason random tournaments were included in our test set is to examine, in some sense, the worst-case behavior of the proposed method since the algorithm was meant to be used with *sparse* graphs, whereas tournaments stem from the complete graph. The results for 707 graphs are shown in Figure 2. The proposed method, despite being in a worst-case scenario, performs better than the method of Section 3.1 for $n < 32$. As expected, the method of Section 3.1 eventually outperforms the proposed method, since the method of Section 3.1 was tailored for tournaments.

Complete directed graph. Even though the complete directed graph has an analytic solution, it may trigger the worst-case performance of certain algorithms. For example, the complete graph is intractable with the minimum set cover approach of Section 3.2 for $n > 10$ (it has more than 10^7

simple cycles; see Section 5.3). The proposed method extends the minimum set cover approach, and hence it is reasonable to test on complete graphs as well. Both the proposed method and the integer programming approach of Section 3.1 solve the complete graph already on the root node for $n = 30$ well under 1 second. The only conclusion that we could draw from this experiment is that the complete graph does not trigger poor performance in either method (although it could have been the case).

6 CONCLUSION

Apart from a review of the literature, this work has three main contributions:

(1) It proposes a novel exact method for solving the minimum feedback arc set problem.
(2) It establishes a large benchmark suite of graphs of theoretical and practical interest as a test bed for future exact algorithms.
(3) The new method is tested on this benchmark suite, and the practical limits are clearly evaluated.

Solving the entire benchmark suite takes more than 1 CPU-month on a modern desktop machine just for the proposed method alone. However, as discussed in Section 5.3, many of these graphs are intractable for the other known exact methods. We are not aware of any existing benchmark that would be comparable to ours—that is, (a) a benchmark for generic sparse graphs, (b) using integer linear programming and a general-purpose integer linear programming solver, and (c) reproducible with reasonable effort.

The 4,468 test graphs used in this study, together with their minimum feedback arc set, are available in electronic form on GitHub as plain text files [8]. The format of these text files is simple and documented; it should be easy to parse them in any mainstream programming language. The source of the proposed method is also available on GitHub.

In Section 5.5, the proposed method was compared to the alternative integer programming approach of Section 3.1 on 3,737 sparse random graphs of varying size (n) and sparsity (c). The proposed method shows significantly better scaling on these sparse random graphs than the alternative approach. The upper end of the benchmark set gives the practical limits of the proposed method: With a timeout limit of 1 CPU-hour, 16 of 101 test problems would have remained unsolved for $n = 75$, $c = 9$, and 28 out 101 problems for $n = 100, c = 7$.

The highly structured sparse graphs of Section 5.6 are intractable with the alternative integer programming approach of Section 3.2, due to the sheer number of simple cycles. Exploiting fixed-parameter tractability is not an option either, due to the size of the minimum feedback arc set of these challenging graphs. However, with the proposed method, it is still tractable to find the minimum feedback arc set of these graphs, and it performs consistently and significantly better than the method of Section 3.1. These graphs are also at the practical limit of the proposed method: With a timeout limit of 1 hour, 4 out of 24 problems would have remained unsolved even with the proposed method (see Tables 2 and 3).

As discussed in Section 1.1, specialized methods are available for tournaments or dense graphs; the proposed method is not meant to be used with such graphs. Tournaments trigger, in some sense, the worst-case behavior of the proposed method since the algorithm is meant to be used with *sparse* graphs, whereas tournaments are complete graphs when the orientation is ignored. Since the method of Section 3.1 was tailored for tournaments, it eventually outperforms the proposed method for random tournaments, as it is shown Section 5.7 on 707 random tournaments. Nevertheless, for random tournaments of size $n < 32$, the proposed method is faster on average. We consider this worst-case performance as satisfactory for a method designed for the sparse case.

These experiments also pushed the proposed method to its practical limit: 40 of the 101 problems would have remained unsolved with a timeout limit of 30 CPU-minutes for $n = 34$.

Pre-processing plays an insignificant role for the graphs in the benchmark set: The graphs could not be simplified considerably, if at all, and the algorithms had access to the same pre-processing techniques to ensure fair comparisons. For large, sparse graphs, the runtime improvements over the method of Section 3.1 are primarily attributable to the substantially smaller integer linear programs that the proposed method has to solve. These integer linear programs have $\Theta(n)$ times fewer columns (deterministically, see Sections 3.1 and 3.2), and, for example, for $n = 100$ approximately 100 times fewer rows in the integer linear programs (empirically, on this benchmark set; see Section 4.1). Although size is not necessarily a good measure of difficulty, the 10^4 times fewer nonzeros in the coefficient matrix of the integer linear programs certainly give the proposed method a significant advantage.

APPENDIX

A SAFELY REMOVING ARCS

We say that a set of arcs is *safe to remove* if removing these arcs from G and adding them to the feedback arc set does not change the minimum cost solution. In other words, there must be at least one minimum cost feedback arc set in G that contains all arcs of an arc set that is safe to remove.

The goal of the algorithm is to find arcs that can be safely removed. If the algorithm fails to find such an arc set, no simplification takes place and nothing is removed from G.

Our original intent was to create an algorithm for the pre-solve phase of the proposed method that generates an equivalent but simpler graph than the input. The algorithm presented in this section turned out to be impractical for such purposes due to its high computational costs. Nevertheless, certain pieces of it proved to be useful and are included in the proposed method (see *Hand-coded procedures for common patterns* in Appendix A.3), and it also provided us insights into the structure of Problem 10 by identifying a challenging subgraph of it that has no arcs that are safe to remove.

A.1 Intuition

We start with the following simple example. The input graph G is assumed to be unweighted—that is, the cardinality of the feedback arc set is to be minimized in this example. Furthermore, let us assume that the nodes u and v participate in a two-cycle, and u has an additional in-arc and v has an additional out-arc (Figure 3).

Fig. 3. An example showing how the simplification works on a two-cycle.

This two-cycle has to be broken to make G acyclic, and there are exactly three possibilities to break this two-cycle: We remove (a) the arc (u, v), or (b) the arc (v, u), or (c) both. The third option is obviously not an optimal solution to break the two-cycle. We now discuss the first two options.

The arc (v, u) cannot participate in any simple cycle other than the two-cycle shown in Figure 3, because u has a single out-arc and that points to v, or alternatively, because v has a single in-arc and that comes from u. However, the arc (u, v) can participate in other simple cycles of G; let C denote the set of these simple cycles. The cycles in C still have to be broken to make G acyclic. Therefore, we can conclude that removing the arc (v, u) cannot yield a strictly lower cost solution than removing (u, v) since removing (u, v) breaks both the two-cycle and all other cycles in C (if any). The arc (u, v) can be safely removed; the global optimum remains unchanged.

A.2 Rule to Identify Arcs That Are Safe to Remove

Our observations made in the previous section generalize. We compute two costs:

- the exact minimum cost c_1 of making an arbitrary induced subgraph G' of (the weighted or unweighted) G acyclic, and
- the cost c_2 of making both G' acyclic and breaking also all of those cycles of G that can have an arc in G' by removing arcs in G' only. Let F' denote such an arc set; this arc set has cost c_2.

If $c_1 = c_2$, then it is safe to remove F' from G and to add it to the feedback arc set of G. The argument is the same as it was in the example. Making G' acyclic alone is not cheaper than the cost of F', and removing F' makes G' acyclic and also breaks *all* of those cycles of G that have an arc in G'.

The algorithm reports failure if $c_1 < c_2$ and no simplification takes place. (Note that $c_1 \leq c_2$ must hold.) Only c_1 has to be computed exactly (rigorously); it is sufficient to use a heuristic to find an appropriate F'.

A.3 Implementation

Hand-coded procedures for common patterns. Although the arc removal rule of the previous section can be implemented in a generic fashion, it proved to be fruitful to hand-code certain common patterns (common induced subgraphs) and their simplified forms. The primary reason is efficiency, but our simple algorithm for generating subgraphs of the input graph also benefits from these simplifications, as we will see shortly.

Common patterns such as runs, self-loops, two-cycles, three-cycles, 3-arc bypasses, and their corresponding simplified forms are hand-coded (Figure 4). We assume throughout this article that the input graph G does not have self-loops. However, self-loops are temporarily allowed when the hand-coded rules are applied; self-loops are no longer present when the hand-coded simplifications finish. There is only one arc that can break a self-loop; this arc is always removed and added to the feedback arc set. The other patterns were selected by inspecting the graphs in our test set. One could derive rules for other patterns, too, depending on what is believed to be common in the expected input graphs.

Once all hand-coded simplifications have been performed, and the graph cannot be simplified any further with these rules, the remaining graph is split into nontrivial SCCs, and the hand-coded simplification procedures are run again on each SCC. If neither the hand-coded simplification procedures nor splitting into nontrivial SCCs result in any progress, we continue with the computationally more expensive integer programming–based simplification, as discussed next.

Selecting the induced subgraphs G'. To apply the rule of Appendix A.2, an induced subgraph G' must be selected. G' can be an arbitrary induced subgraph of G, but it is assumed that finding a minimum feedback arc set of G' with an exact method is still tractable. Furthermore, we assume that both G' and G are nontrivial SCCs; the algorithm would produce valid but mostly useless results otherwise.

The following procedure is used to construct G'. DFS is started from an appropriately chosen node n of G (more on this in the next paragraph), but the search is limited in depth by a pre-defined constant d. The induced subgraph of the visited nodes is created, and that nontrivial SCC (if any) is selected that contains n. This SCC is G'. The algorithm reports failure if there is no such nontrivial SCC (and therefore no simplification takes place). This procedure is rather plain: For example, in a complete graph, it produces $G' \equiv G$ even with $d = 1$.

Each node of G is probed in the arc removing algorithm, one after the other in an arbitrary sequence, and starting with $d = 1$ as depth limit for DFS. If no safe arc set is found at any of the

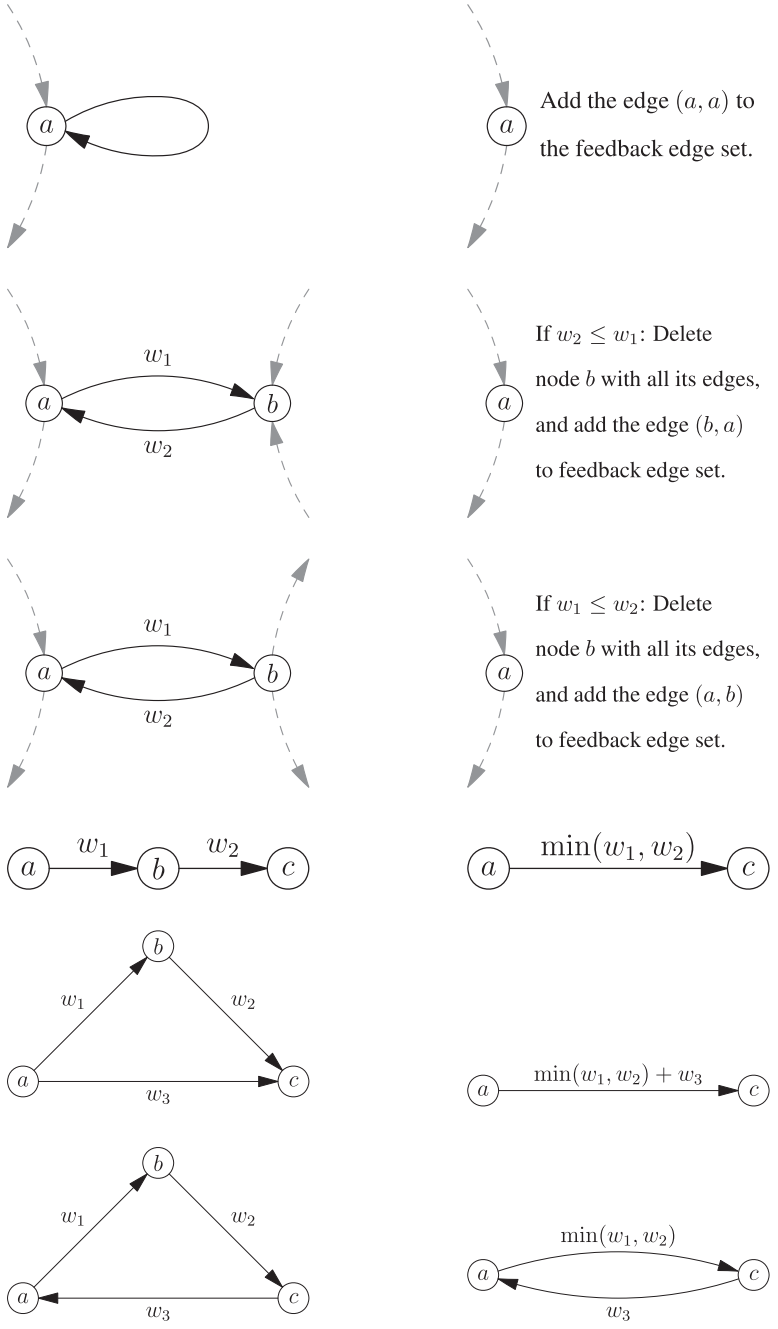

Fig. 4. Common patterns whose simplification is hand-coded mainly for efficiency reasons. The left column shows the induced subgraphs of G, and the right column shows the corresponding simplified form. From top to bottom: (1) removing self-loops, (2) breaking two-cycles where b has out-degree 1 in G, (3) breaking two-cycles where b has in-degree 1 in G, (4) removing runs, (5) rewriting 3-arc bypasses, and (6) rewriting three-cycles. In cases (4) through (6), the node b must have in-degree 1 and out-degree 1 in G.

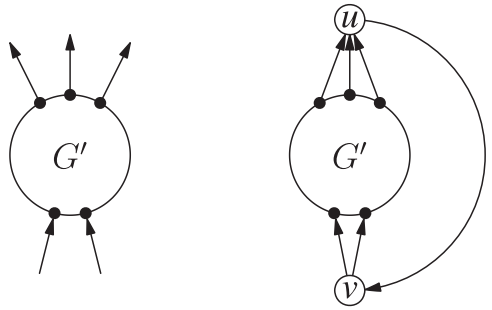

Fig. 5. Left: The nodes on the boundary of the induced subgraph G' are shown as black dots, together with their arcs not in G'. Right: The extended G', the H graph. The nodes u and v and their arcs are fake (not in G).

nodes, d is increased by 1, and each node of G is probed again. The procedure stops when d exceeds the user-defined limit d_{max} (= 5 by default in our implementation) without finding any safe arc set. However, whenever a safe arc set is found, it is removed and added to the feedback arc set. The remaining part of G is split into nontrivial SCCs, the runs and the 3-arc bypasses are iteratively removed with the hand-coded simplifications, and the resulting nontrivial SCCs are appended to the SCCs to be processed. This arrangement is not ideal but is easy to implement. Note that if the upper bound d_{max} is large enough, G' will be identical to G—that is, we get back the original minimum feedback arc set problem. The constant d_{max} will be referred to as *cutoff in DFS*.

Computing c_1. The computation of c_1 must be exact; any exact method (including the proposed method of Section 4) can be applied. In our implementation, c_1 is computed by solving (4) with the complete cycle matrix. Therefore, G' is assumed to be small enough so that all of its simple cycles can be enumerated. One way to enforce this is a naive trial and error approach: Johnson's algorithm [62] for enumerating simple cycles can be implemented in a lazy fashion [53]—that is, it can be aborted after a pre-defined number of simple cycles (e.g., 100 or 1,000) have been found. If an induced subgraph G' has more simple cycles than this pre-defined threshold, the algorithm gives up and reports failure. This user-defined limit for the number of simple cycles will be referred to as *cycle budget* per SCC.

If enumerating all simple cycles in G' finishes within the pre-defined limit for the number of simple cycles, the corresponding integer program (4) is solved. This gives the cost c_1 of making G' acyclic alone.

Computing c_2. We create a graph H in which any arc of G' that can possibly participate in a simple cycle in G necessarily participates in a simple cycle in H as well. We could select G as H, but it would not be practical: We do not want to unnecessarily introduce new simple cycles in H.

A node is on the *boundary* of G' if it has either an in- or out-arc whose other endpoint is not in G'; let B denote this set of nodes. Let us consider those simple cycles in G that have at least an arc in G' but not all of their arcs; let C denote the set of these simple cycles. The cycles in C must enter and leave G' at distinct nodes (possibly multiple times), and these nodes must be in B. We create a new graph H in which each node in B necessarily participates in at least one simple cycle that has arcs outside G'. This will ensure that any arc in G' that appears in a cycle in C will also be involved in a simple cycle in H that has arcs outside G'.

The reader is referred to Figure 5 before reading the explanation that follows. We extend G' by adding two fake nodes u and v to it, together with the following fake arcs. For each arc in G that has its initial node (tail) t in G' but its terminal node (head) not in G' (arcs "sticking out" of G'), we add the arc (t, u) to G'. Similarly, for each arc that has its terminal node (head) h in G' but its

initial node (tail) not in G', we add the arc (v, h) to G'. Finally, we add the arc (u, v). Let H denote the graph that we obtained; G' is obviously an induced subgraph of H.

H ensures that all nodes on the boundary of G' participate in at least one simple cycle that has an arc outside G': This is the cycle that goes through the arc (u, v). Therefore, if we compute a feedback arc set F' of H such that it only contains arcs that were present in G' (no arcs incident to u or v), then this arc set, when removed from G, will make G' acyclic and breaks all of the cycles in C as well.

There is a corner case in the preceding construction of H that is currently not handled by the algorithm: If a cycle of G has exactly one node in G', then it is not possible to make H acyclic by removing arcs from G' only. If this corner case is encountered, the algorithm gives up and reports failure. This is obviously a missed opportunity and should be handled in the future, but the correctness is not affected.

A.4 Arc Removal Experiments

In Table 4, we give the minimum cycle budget and the minimum cutoff that are necessary to solve the test problems exclusively with the arc removal algorithm of Appendix A.3. Although it is inefficient to solve these problems with the arc removal algorithm only, the numbers nevertheless show that Problems 2 through 10 can be simplified, Problem 10 even by a factor of 73 with respect to the number of simple cycles. Problem 11 is the SCC (G') with the most simple cycles that occurred during solving Problem 10 with the arc removal algorithm only. Accordingly, Problem 11 does not have a proper subgraph that has safe to remove arcs. See also Figures 6 and 7.

Table 4. Minimum Cycle Budget and Minimum Cutoff to Solve the Test Problems with the Arc Removal Algorithm Only

Problem ID	Optimum	Cycles	Cycle Budget	Cutoff
1	15	409	409	1
2	2	22	3	1
3	6	27	9	1
4	6	20	8	2
5	3	10	3	1
6	5	11	3	1
7	3	31	18	3
8	5	103	41	3
9	8	22	13	2
10	12	13,746	187	5
11	6	187	187	4

Problem 1 is the complete graph and has no safe to remove arcs. Problem 11 has no safe to remove arcs as expected (see the text).

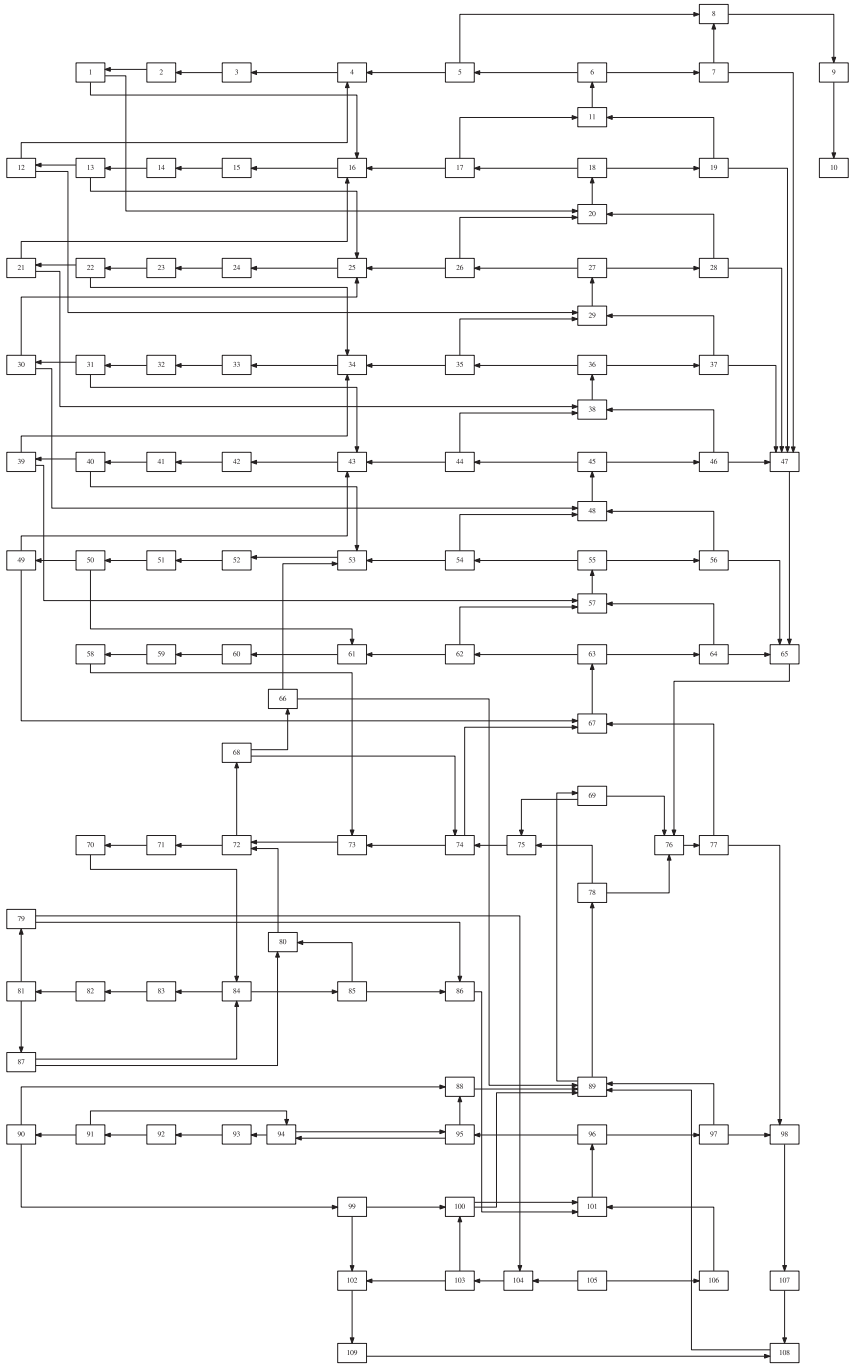

Fig. 6. Problem 10, origin: [47] (heavy water). This graph has 109 nodes, 163 arcs, and 13,746 simple cycles, and the cardinality of the minimum feedback arc set is 12.

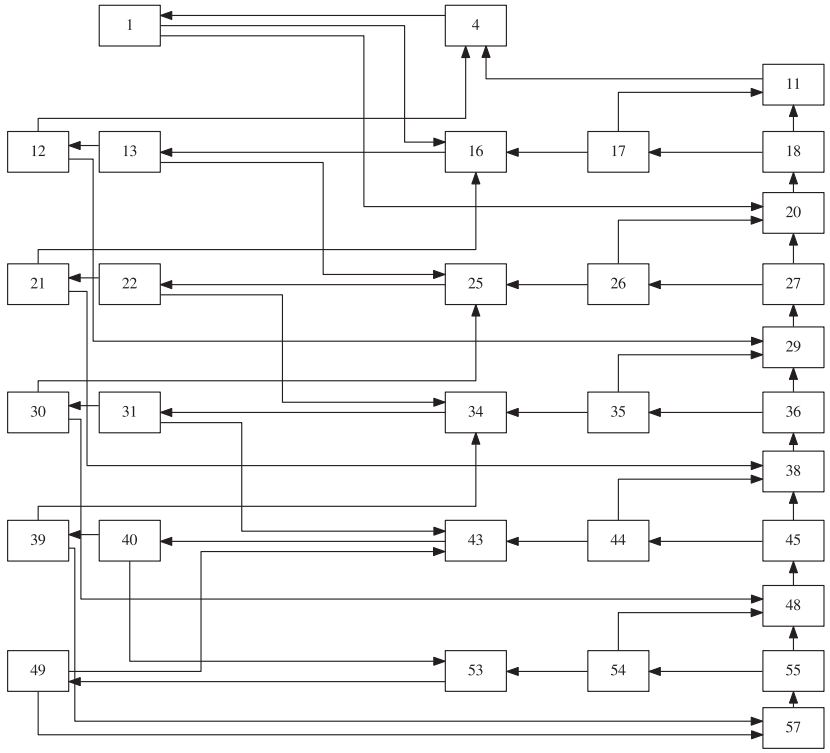

Fig. 7. Problem 11, derived from Problem 10 as discussed in Appendix A.4. This graph has 32 nodes, 52 arcs, and 187 simple cycles, and the cardinality of the minimum feedback arc set is 6.

REFERENCES

[1] Tobias Achterberg. 2007. *Constraint Integer Programming*. Ph.D. Dissertation. Technische Universität Berlin.

[2] Tobias Achterberg. 2009. SCIP: Solving constraint integer programs. *Mathematical Programming Computation* 1, 1 (July 2009), 1–41.

[3] N. Alon. 2006. Ranking tournaments. *SIAM Journal on Discrete Mathematics* 20, 1 (2006), 137–142. DOI: https://doi.org/10.1137/050623905

[4] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press; New York, NY.

[5] S. Arora, A. Frieze, and H. Kaplan. 1996. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *Proceedings of the 1996 37th Annual Symposium on Foundations of Computer Science*. 21–30.

[6] Aspen Technology, Inc. 2009. EO and SM variables and synchronization. In *Aspen Simulation Workbook, Version Number: V7.1*. Burlington, MA, 110.

[7] P. Austrin, R. Manokaran, and C. Wenner. 2013. On the NP-hardness of approximating ordering constraint satisfaction problems. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*. Lecture Notes in Computer Science, Vol. 8096. Springer, 26–41.

[8] A. Baharev. 2019. Exact and Heuristic Methods for Tearing. Retrieved February 10, 2021 from https://sdopt-tearing.readthedocs.io.

[9] R. W. Barkley and R. L. Motard. 1972. Decomposition of nets. *Chemical Engineering Journal* 3 (1972), 265–275.

[10] Oliver Bastert and Christian Matuszewski. 2001. Layered drawings of digraphs. In *Drawing Graphs—Methods and Models*, Michael Kaufman and Dorothea Wagner (Eds.). Springer, Berlin, Germany, 87–120. DOI: https://doi.org/10.1007/3-540-44969-8_5

[11] Bonnie Berger and Peter W. Shor. 1990. Approximation algorithms for the maximum acyclic subgraph problem. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'90)*. 236–243.

[12] Livio Bertacco, Lorenzo Brunetta, and Matteo Fischetti. 2008. The linear ordering problem with cumulative costs. *European Journal of Operational Research* 189, 3 (2008), 1345–1357. DOI: https://doi.org/10.1016/j.ejor.2006.03.071
[13] Lorenz T. Biegler, Ignacio E. Grossmann, and Arthur W. Westerberg. 1997. *Systematic Methods of Chemical Process Design*. Prentice Hall PTR, Upper Saddle River, NJ.
[14] N. L. Book and W. F. Ramirez. 1984. Structural analysis and solution of systems of algebraic design equations. *AIChE Journal* 30, 4 (1984), 609–622.
[15] F. J. Brandenburg and K. Hanauer. 2011. *Sorting Heuristics for the Feedback Arc Set Problem*. Technical Report MIP-1104. Department of Informatics and Mathematics, University of Passau, Germany.
[16] Saskia Bublitz, Erik Esche, Gregor Tolksdorf, Volker Mehrmann, and Jens-Uwe Repke. 2017. Analysis and decomposition for improved convergence of nonlinear process models in chemical engineering. *Chemie Ingenieur Technik* 89, 11 (2017), 1503–1514. DOI: https://doi.org/10.1002/cite.201700041
[17] Pierre Charbit, Stéphan Thomassé, and Anders Yeo. 2007. The minimum feedback arc set problem is NP-hard for tournaments. *Combinatorics, Probability and Computing* 16, 1 (2007), 1–4. DOI: https://doi.org/10.1017/S0963548306007887
[18] M. Charikar, K. Makarychev, and Y. Makarychev. 2007. On the advantage over random for maximum acyclic subgraph. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. IEEE, Los Alamitos, CA, 625–633.
[19] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. 2008. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM* 55, 5 (2008), Article 21, 19 pages.
[20] James H. Christensen. 1970. The structuring of process optimization. *AIChE Journal* 16, 2 (1970), 177–184.
[21] J. H. Christensen and D. F. Rudd. 1969. Structuring design computations. *AIChE Journal* 15 (1969), 94–100.
[22] V. Chvatal. 1979. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research* 4, 3 (1979), 233–235.
[23] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. 2009. *Introduction to Algorithms* (3rd ed.). MIT Press, Cambridge, MA.
[24] David Coudert, Afonso Ferreira, and Xavier Muñoz. 1999. OTIS-based multi-hop multi-OPS lightwave networks. In *Parallel and Distributed Processing: 11th IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing San Juan, Puerto Rico, USA, April 12–16, 1999 Proceedings*. Springer, Berlin, Germany, 897–910.
[25] G. Dantzig, R. Fulkerson, and S. Johnson. 1954. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America* 2, 4 (1954), 393–410.
[26] Dassault Systèmes AB. 2016. Advanced modelica support. In *Dymola—Dynamic Modeling Laboratory: User Manual*. Vol. 2. Dassault Systèmes AB, 399–445.
[27] Camil Demestrescu and Irene Finocchi. 2001. Breaking cycles for minimizing crossings. *ACM Journal of Experimental Algorithmics* 6 (Dec. 2001), Article 2. DOI: https://doi.org/10.1145/945394.945396
[28] Camil Demetrescu and Irene Finocchi. 2003. Combinatorial algorithms for feedback problems in directed graphs. *Information Processing Letters* 86, 3 (2003), 129–136. DOI: https://doi.org/10.1016/S0020-0190(02)00491-X
[29] Narsingh Deo. 1974. *Graph Theory with Applications to Engineering and Computer Science*. Prentice Hall, Englewood Cliffs, NJ.
[30] A. C. Dimian, C. S. Bildea, and A. A. Kiss. 2014. Steady-state flowsheeting. In *Integrated Design and Simulation of Chemical Processes* (2nd ed.). Computer-Aided Chemical Engineering, Vol. 35. Elsevier Amsterdam, Netherlands, 73–125.
[31] R. G. Downey and M. R. Fellows. 2013. In *Fundamentals of Parameterized Complexity*, D. Gries and F. B. Schneider (Eds.). Springer-Verlag, London, UK.
[32] D. Z. Du and F. K. Hwang. 1988. Generalized de Bruijn digraphs. *Networks* 18, 1 (1988), 27–38.
[33] P. Eades and X. Lin. 1995. A new heuristic for the feedback arc set problem. *Australasian Journal of Combinatorics* 35, 4 (1995), 15–25.
[34] Peter Eades, Xuemin Lin, and W. F. Smyth. 1993. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters* 47, 6 (1993), 319–323.
[35] P. Erdős and A. Rényi. 1959. On random graphs I. *Publicationes Mathematicae* 6 (1959), 290–297.
[36] G. Even, J. (Seffi) Naor, B. Schieber, and M. Sudan. 1998. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica* 20, 2 (1998), 151–174.
[37] F. V. Fomin and D. Kratsch. 2010. Measure and conquer. In *Exact Exponential Algorithms*. Springer, Berlin, Germany, 101–124.
[38] L. R. Ford and D. R. Fulkerson. 1958. A suggested computation for maximal multi-commodity network flows. *Management Science* 5, 1 (1958), 97–101.
[39] R. S. Garfinkel and G. L. Nemhauser. 1972. *Integer Programming*. Wiley, New York, NY.

[40] P. L. Genna and R. L. Motard. 1975. Optimal decomposition of process networks. *AIChE Journal* 21, 4 (1975), 656–663.
[41] E. N. Gilbert. 1959. Random graphs. *Annals of Mathematical Statistics* 30 (1959), 1141–1144.
[42] Fred Glover, T. Klastorin, and D. Kongman. 1974. Optimal weighted ancestry relationships. *Management Science* 20, 8 (1974), 1190–1193. DOI: https://doi.org/10.1287/mnsc.20.8.1190
[43] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. 1984. A cutting plane algorithm for the linear ordering problem. *Operations Research* 32, 6 (1984), 1195–1220.
[44] M. Grötschel, M. Jünger, and G. Reinelt. 1985. *Acyclic Subdigraphs and Linear Orderings: Polytopes, Facets, and a Cutting Plane Algorithm*. Springer, Dordrecht, Netherlands, 217–264. DOI: https://doi.org/10.1007/978-94-009-5315-4_7
[45] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. 1985. On the acyclic subgraph polytope. *Mathematical Programming* 33, 1 (Sept. 1985), 28–42. DOI: https://doi.org/10.1007/BF01582009
[46] G. Guardabassi. 1971. A note on minimal essential sets. *IEEE Transactions on Circuit Theory* 18, 5 (1971), 557–560.
[47] T. Gundersen. 1982. *Decomposition of Large Scale Chemical Engineering Systems*. Ph.D. Dissertation. Department of Chemical Engineering, University of Trondheim, Norway.
[48] T. Gundersen and T. Hertzberg. 1983. Partitioning and tearing of networks applied to process flowsheeting. *Modeling, Identification and Control* 4, 3 (1983), 139–165.
[49] Prem K. Gupta, Arthur W. Westerberg, John E. Hendry, and Richard R. Hughes. 1974. Assigning output variables to equations using linear programming. *AIChE Journal* 20, 2 (1974), 397–399.
[50] Gurobi. 2014. Gurobi Optimizer Version 6.0. Houston, Texas: Gurobi Optimization, Inc., May 2015. Software Program. Retrieved February 10, 2021 from http://www.gurobi.com.
[51] V. Guruswami, J. Håstad, R. Manokaran, P. Raghavendra, and M. Charikar. 2011. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM Journal on Computing* 40, 3 (2011), 878–914.
[52] V. Guruswami, R. Manokaran, and P. Raghavendra. 2008. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *Proceedings of the 2008 IEEE 49th Annual Symposium on Foundations of Computer Science (FOCS'08)*. 573–582.
[53] 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 (SciPy'08)*. 11–15.
[54] Michael Hecht. 2018. Exact localisations of feedback sets. *Theory of Computing Systems* 62, 5 (July 2018), 1048–1084. DOI: https://doi.org/10.1007/s00224-017-9777-6
[55] R. Hernandez and R. W. H. Sargent. 1979. A new algorithm for process flowsheeting. *Computers & Chemical Engineering* 3, 1–4 (1979), 363–371.
[56] V. Hlaváček. 1977. Analysis of a complex plant-steady state and transient behavior. *Computers & Chemical Engineering* 1, 1 (1977), 75–100.
[57] M. Imase and M. Itoh. 1981. Design to minimize diameter on building-block network. *IEEE Transactions on Computers* C-30, 6 (1981), 439–442.
[58] M. Imase and M. Itoh. 1983. A design for directed graphs with minimum diameter. *IEEE Transactions on Computers* C-32, 8 (1983), 782–784.
[59] Makoto Imase, Terunao Soneoka, and Keiji Okada. 1986. Fault-tolerant processor interconnection networks. *Systems and Computers in Japan* 17, 8 (1986), 21–30.
[60] Iaroslav Ispolatov and Sergei Maslov. 2008. Detection of the dominant direction of information flow and feedback links in densely interconnected regulatory networks. *BMC Bioinformatics* 9 (2008), 424. DOI: https://doi.org/10.1186/1471-2105-9-424
[61] Y. V. S. Jain and J. M. Eakman. 1971. Identification of process flow networks. In *Proceedings of the AIChE 68th National Meeting*.
[62] Donald B. Johnson. 1975. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing* 4, 1 (1975), 77–84.
[63] David S. Johnson. 1973. Approximation algorithms for combinatorial problems. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC'73)*. ACM, New York, NY, 38–49.
[64] R. Kaas. 1981. A branch and bound algorithm for the acyclic subgraph problem. *European Journal of Operational Research* 8, 4 (1981), 355–362.
[65] M. Frans Kaashoek and David R. Karger. 2003. Koorde: A simple degree-optimal distributed hash table. In *Peer-to-Peer Systems II: Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003. Revised Papers*. Springer, Berlin, Germany, 98–107.
[66] Viggo Kann. 1992. *On the Approximability of NP-Complete Optimization Problems*. Ph.D. Dissertation. Royal Institute of Technology Stockholm.
[67] Richard M. Karp. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger (Eds.). Springer, 85–103.

[68] W. H. Kautz. 1968. Bounds on directed (*d, k*) graphs. In *Theory of Cellular Logic Networks and Machines*. Technical Report AFCRL-68-0668 Final Report. Air Force Cambridge Research Laboratories, Cambridge, MA, 20–28.
[69] John G. Kemeny. 1959. Mathematics without numbers. *Daedalus* 88, 4 (1959), 577–591. http://www.jstor.org/stable/20026529
[70] Claire Kenyon-Mathieu and Warren Schudy. 2007. How to rank with few errors. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC'07)*. ACM, New York, NY, 95–103.
[71] Subhash Khot. 2002. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC'02)*. ACM, New York, NY, 767–775.
[72] W. Lee, J. H. Christensen, and D. F. Rudd. 1966. Design variable selection to simplify process calculations. *AIChE Journal* 12, 6 (1966), 1104–1115.
[73] W. Lee and D. F. Rudd. 1966. On the ordering of recycle calculations. *AIChE Journal* 12, 6 (1966), 1184–1190.
[74] H. W. Lenstra. 1973. *The Acyclic Subgraph Problem*. Technical Report BW 26/73. Faculteit der Wiskunde en Natuurwetenschappen, Mathematisch Centrum, Amsterdam. http://hdl.handle.net/1887/2116
[75] W. K. Lewis and G. L. Matheson. 1932. Studies in distillation. *Industrial & Engineering Chemistry* 24 (1932), 494–498.
[76] Dongsheng Li, Xicheng Lu, and Jinshu Su. 2004. Graph-theoretic analysis of Kautz topology and DHT schemes. In *Network and Parallel Computing: IFIP International Conference, NPC 2004, Wuhan, China, October 18-20, 2004. Proceedings*. Springer, Berlin, Germany, 308–315.
[77] L. Lovász. 1975. On the ratio of optimal integral and fractional covers. *Discrete Mathematics* 13, 4 (1975), 383–390.
[78] C. L. Lucchesi. 1976. *A Minimax Equality for Directed Graphs*. Ph.D. Dissertation. University of Waterloo, Ontario.
[79] C. L. Lucchesi and D. H. Younger. 1978. A minimax theorem for directed graphs. *Journal of the London Mathematical Society* 2, 17 (1978), 369–374.
[80] Richard S. H. Mah. 1990. Computation sequence in process flowsheet calculations. In *Chemical Process Structures and Information Flows*, Richard S. H. Mah (Ed.). Butterworth-Heinemann, 125–183.
[81] Rafael Martí and Gerhard Reinelt. 2011. *The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization*. Applied Mathematical Sciences, Vol. 175. Springer-Verlag, Berlin, Germany.
[82] Ronald E. Miller and Peter D. Blair. 2009. *Input-Output Analysis: Foundations and Extensions* (2nd ed.). Cambridge University Press. DOI: https://doi.org/10.1017/CBO9780511626982
[83] J. E. Mitchell and B. Borehers. 2000. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In *High Performance Optimization*, H. Frenk, K. Roos, T. Terlaky, and S. Zhang (Eds.). Applied Optimization, Vol. 33. Springer-Science+Business Media, B.V., Dordrecht, Netherlands, 349–366.
[84] Modelica. 2018. Modelica and the Modelica Association. Retrieved October 14, 2018 from https://www.modelica.org/.
[85] Modelon AB. 2018. JModelica.org User Guide, Version 2.2. Retrieved October 14, 2018 from https://jmodelica.org/downloads/UsersGuide.pdf.
[86] J. M. Montagna and O. A. Iribarren. 1988. Optimal computation sequence in the simulation of chemical plants. *Computers & Chemical Engineering* 12, 1 (1988), 71–79.
[87] Rodolphe L. Motard and Arthur W. Westerberg. 1981. Exclusive tear sets for flowsheets. *AIChE Journal* 27 (1981), 725–732.
[88] OpenModelica. 2018. OpenModelica User's Guide. Retrieved October 14, 2018 from https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omchelptext.html.
[89] T. Orenstein, Z. Kohavi, and I. Pomeranz. 1995. An optimal algorithm for cycle breaking in directed graphs. *Journal of Electronic Testing* 7, 1–2 (1995), 71–81.
[90] Christos H. Papadimitriou and Mihalis Yannakakis. 1991. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences* 43, 3 (1991), 425–440.
[91] S. Park and S. B. Akers. 1992. An efficient method for finding a minimal feedback arc set in directed graphs. In *Proceedings of the 1992 IEEE International Symposium on Circuits and Systems (ISCAS'92)*, Vol. 4. 1863–1866.
[92] 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.
[93] Vijaya Ramachandran. 1988. Finding a minimum feedback arc set in reducible flow graphs. *Journal of Algorithms* 9, 3 (1988), 299–313.
[94] Venkatesh Raman and Saket Saurabh. 2007. Improved fixed parameter tractable algorithms for two "edge" problems: MAXCUT and MAXDAG. *Informations Processing Letters* 104, 2 (2007), 65–72. DOI: https://doi.org/10.1016/j.ipl.2007.05.014
[95] Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. 2007. Efficient exact algorithms through enumerating maximal independent sets and other techniques. *Theory of Computing Systems* 41, 3 (Oct. 2007), 563–587. DOI: https://doi.org/10.1007/s00224-007-1334-2
[96] J. R. Roach, B. K. O'Neill, and D. A. Hocking. 1997. A new synthetic method for stream tearing in process systems analysis. *Chemical Engineering Communications* 161, 1 (1997), 1–14.

[97] D. K. Pradhan S. M. Reddy, and J. G. Kuhl. 1980. *Directed Graphs with Minimal Diameter and Maximal Connectivity*. Technical Report. School of Engineering, Oakland University, RochesterMI.
[98] R. Saket and M. Sviridenko. 2012. New and improved bounds for the minimum set cover problem. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*. Lecture Notes in Computer Science, Vol. 7408. Springer, 288–300.
[99] G. Sander. 1999. Graph layout for applications in compiler construction. *Theoretical Computer Science* 217, 2 (1999), 175–214.
[100] R. W. H. Sargent. 1978. The decomposition of systems of procedures and algebraic equations. In *Numerical Analysis*. Lecture Notes in Mathematics, Vol. 630. Springer, 158–178.
[101] R. W. H. Sargent and A. W. Westerberg. 1964. Speed-up in chemical engineering design. *Transactions of the Institution of Chemical Engineers* 42 (1964), 190–197.
[102] Benno Schwikowski and Ewald Speckenmeyer. 2002. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics* 117, 1–3 (2002), 253–265.
[103] P. D. Seymour. 1995. Packing directed circuits fractionally. *Combinatorica* 15, 2 (1995), 281–288.
[104] Patrick Slater. 1961. Inconsistencies in a schedule of paired comparisons. *Biometrika* 48, 3–4 (1961), 303–312. http://www.jstor.org/stable/2332752
[105] Ljiljana Spadavecchia. 2006. *A Network-Based Asynchronous Architecture for Cryptographic Devices*. Ph.D. Dissertation. College of Science and Engineering, School of Informatics, University of Edinburgh. http://hdl.handle.net/1842/860
[106] Mark A. Stadtherr. 1979. Maintaining sparsity in process design calculations. *AIChE Journal* 25, 4 (1979), 609–615.
[107] M. A. Stadtherr, W. A. Gifford, and L. E. Scriven. 1974. Efficient solution of sparse sets of design equations. *Chemical Engineering Science* 29, 4 (1974), 1025–1034.
[108] M. A. Stadtherr and E. S. Wood. 1984. Sparse matrix methods for equation-based chemical process flowsheeting–I: Reordering phase. *Computers & Chemical Engineering* 8, 1 (1984), 9–18.
[109] K. Sugiyama, S. Tagawa, and M. Toda. 1981. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics* 11, 2 (Feb. 1981), 109–125. DOI: https://doi.org/10.1109/TSMC.1981.4308636
[110] R. E. Tarjan. 1972. Depth first search and linear graph algorithms. *SIAM Journal on Computing* 1 (1972), 146–160.
[111] E. W. Thiele and R. L. Geddes. 1933. Computation of distillation apparatus for hydrocarbon mixtures. *Industrial & Engineering Chemistry* 25 (1933), 289–295.
[112] R. S. Upadhye and E. A. Grens. 1972. An efficient algorithm for optimum decomposition of recycle systems. *AIChE Journal* 18 (1972), 533–539.
[113] Leslie G. Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM Journal on Computing* 8, 3 (1979), 410–421. DOI: https://doi.org/10.1137/0208032
[114] G. V. Varma, K. H. Lau, and D. L. Ulrichson. 1993. A new tearing algorithm for process flowsheeting. *Computers & Chemical Engineering* 17, 4 (1993), 355–360.
[115] A. W. Westerberg and F. C. Edie. 1971. Computer-aided design, part 2: An approach to convergence and tearing in the solution of sparse equation sets. *Chemical Engineering Journal* 2, 1 (1971), 17–25.
[116] Daniel R. Zerbino and Ewan Birney. 2008. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research* 18 (2008), 821–829.
[117] L. Zhou, Z. Han, and K. Yu. 1988. A new strategy of net decomposition in process simulation. *Computers & Chemical Engineering* 12, 6 (1988), 581–588.

Received March 2019; revised October 2020; accepted December 2020