AN EXACT METHOD FOR THE MINIMUM FEEDBACK ARC SET PROBLEM

ALI BAHAREV, HERMANN SCHICHL, ARNOLD NEUMAIER, TOBIAS ACHTERBERG

Abstract. Given a directed graph G, a feedback arc set of G is a subset of its edges containing at least one edge of every cycle in G. Finding a feedback arc set of minimum cardinality is the minimum feedback arc set problem. The minimum set cover formulation of the minimum feedback arc set problem is appropriate as long as all the simple cycles in G can be enumerated. Unfortunately, even sparse graphs can have $\Omega(2^n)$ simple cycles, and such graphs appear in practice. An exact method is proposed for sparse graphs that enumerates simple cycles in a lazy fashion, and extends an incomplete cycle matrix iteratively. In all cases encountered so far, only a tractable number of cycles has to be enumerated until a minimum feedback arc set is found. Numerical results are given on a test set containing computationally challenging sparse graphs, relevant for industrial applications.

Key words. minimum feedback arc set, maximum acyclic subgraph, minimum feedback vertex set, linear ordering problem, tearing

1. Introduction.

A **directed graph** G is a pair (V,E) of finite sets, the vertices V and the edges $E \subseteq V \times V$. It is a **simple graph** if it has no multiple edges or self-loops (edges of the form (v,v)). Let $G = (V,E)$ denote a simple directed graph, and let $n = |V|$ denote the number of vertices (nodes), and $m = |E|$ denote the number of edges. A subgraph H of the graph G is said to be **induced** if, for every pair of vertices u and v of H, (u,v) is an edge of H if and only if (u,v) is an edge of G.

A directed **path** from vertex v to w is an alternating sequence of vertices and edges of G leading from v to w. A path that starts and ends at the same vertex is called a **cycle**. A **simple cycle** is a cycle with no repeating edges and no repeating nodes in the cycle. Two simple cycles are **distinct** if one is not a cyclic permutation of the other. Throughout this paper, whenever simple cycles are mentioned, distinct simple cycles are meant.

A **topological order** of G is a linear ordering of all its nodes such that if G contains an edge (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** [20, Sec. 14.8]. The **topological sort algorithm** of [17, 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 strongly connected components of a directed graph can be found in linear time, that is, in $\Theta(n+m)$ time, see [87] and [17, 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 **feedback vertex set** of a simple directed graph G is a set of vertices whose removal 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 edge set** is a subset of edges containing at least one edge of every cycle in a directed graph. In other words, removing the edges in the feedback edge set from the graph makes the remaining graph a directed acyclic graph. A feedback edge set S is **minimal** if reinsertion of any edge $s \in S$ to the directed acyclic graph induces a cycle. If the edges in a minimal feedback edge 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 edge set problem** is to either construct a feedback edge set

of at most k edges for G, or to prove that no such edge set exists. This problem is called the feedback arc set problem, and it was item 8 on the list of Richard M. Karp's 21 NP-complete problems [53]. We prefer the term feedback edge set to the term feedback arc set. The feedback edge set problem is **fixed-parameter tractable (FPT)**: an $O(n^4 4^k k^3 k!)$ time algorithm is given in [13], that is, this algorithm runs in polynomial time if k is bounded above by a constant. The reader is referred to [22] regarding further details on parameterized complexity and FPT.

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

Finding a feedback edge set of minimum cardinality is the **minimum feedback arc set problem**; we will refer to it as the **minimum feedback edge set problem** hereafter. Similarly, finding a feedback vertex set of minimum cardinality is the **minimum (directed) feedback vertex set problem**. The reductions between the minimum feedback edge 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 [27]. Hereafter we focus on the minimum feedback edge set problem but we wanted to indicate that results for the minimum (directed) feedback vertex set problem are also directly relevant.

The minimum feedback edge set problem is **APX-hard** [52]: Unless $P = NP$, the minimum feedback edge set problem does not have a polynomial-time approximation scheme (PTAS). The minimum feedback edge set problem is **approximation resistant**: Conditioned on the Unique Games Conjecture (UGC) [56], for every $C > 0$, it is NP-hard to find a C-approximation to the minimum feedback edge set problem, see Corollary 1.2. in [39]. One can construct a feedback edge set with cardinality of at most $m/2$ by taking either the forward or backward edges (whichever has smaller cardinality) in an arbitrary ordering of the vertices of G. A better upper bound, $m/2 - n/6$, was derived in [25] by taking into account that edges 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 [82] in his proof; the corresponding algorithm was explicitly given in [27].

The minimum feedback edge set problem is solvable in polynomial time for planar graphs [62, 63], and for reducible flow graphs [73]. A **tournament** is a directed graph without self-loops such that for every two distinct nodes u and v there is exactly one edge with end-nodes u and v. A polynomial-time approximation scheme for minimum weighted feedback edge sets on tournaments is presented in [55].

The complementary problem to the minimum feedback edge set problem is the **maximum acyclic subgraph problem**. The problem was proved to be APX-complete in [70]. The algorithm of [8] finds an acyclic subgraph with $(1/2 + \Omega(1/\sqrt{d_{max}}))m$ edges, 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 edges is sharp in the sense that an infinite class of directed graphs is exhibited in [8] realizing this bound. The previously cited algorithm of [25] provides an acyclic graph with at least $m/2 + n/6$ edges 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 [8]. A polynomial time approximation scheme (running in $n^{O(1/\varepsilon^2)}$ time) was given in [3] 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 [12], which

is a slight improvement over $1/2+\Omega(1/(\log n \log\log n))$ that follows from [27, 82]. The problem is approximation resistant: Conditioned on the UGC, it is NP-hard to approximate the maximum acyclic subgraph problem within $1/2+\varepsilon$ for every $\varepsilon > 0$ [39, 40]. Without assuming the UGC and subject only to $P \neq NP$, the best known inapproximability result is $14/15+\varepsilon$, derived in [5].

The **linear ordering problem** can be defined as searching in a complete weighted directed graph for an acyclic tournament with a maximal sum of edge weights, see e.g. [65] for further details. The maximum acyclic subgraph problem and the linear ordering problem can be transformed into each other by a simple construction [33]; furthermore, the minimum feedback edge set problem is complementary to the maximum acyclic subgraph problem. Therefore, a good algorithm for one problem usually yields a good algorithm for the other.

1.2. 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 edge 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 edge 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 (1) exclusively for the (weighted) minimum feedback edge set problem, e.g., [7, 21, 31, 36, 57, 68, 72, 74, 88, 89, 92], and [9, Ch. 8], (2) for both the minimum feedback edge set problem and for tearing as in our terminology as defined above, see e.g. [43, 64, 67, 78], and (3) primarily in our sense, e.g., [10, 15, 37, 42, 58, 84–86, 90]. This issue seems to be specific to the chemical engineering literature: For example, in the electrical engineering literature, tearing is used in our sense.

The reason why the (weighted) minimum feedback edge 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 can be used for initializing equation-oriented models, see e.g. [4].

2. Heuristics. The literature on the various heuristics for the minimum feedback edge 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 the 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 edge set problem is approximation resistant. In practice, it usually means that the difference between the solution found by a heuristic and the optimal solution can be as large as $O(n)$.

The minimum set cover problem approach. The greedy heuristic of [57] 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 [49]. We gradually build the feedback edge set by always picking that edge 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 edge 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 [16, 48, 61, 76]. Simplification rules can be applied

to reduce the graph in each iteration step, before removing an edge or edges, see [30, 57, 72], [9, p. 279], or [22, p. 114]. Sophisticated tie-breaking rules are also proposed in [57].

Unfortunately, the weakness of this heuristic is that even sparse graphs can have $\Omega(2^n)$ simple cycles [81], and such graphs appear in practice, see Section 5.4.

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 that, e.g., 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 edge set is built up iteratively. The input graph is simplified in each step before removing an edge or edges; this simplification can include splitting into SCCs, and then dropping the trivial SCCs (a trivial SCC consists of a single node), breaking two-cycles appropriately, etc. 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 edges of that node are removed. For example, a node in an SCC with a single in-edge but with many out-edges is a good candidate: Removing its single in-edge breaks all the cycles that pass through that node, and the number of destroyed simple cycles is at least the out-degree of that node (each out-edge 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), then all of its in- or its out-edges removed, whichever edge set is of smaller cardinality. The algorithm continues with the simplification. The heuristic terminates when there are no edges left. This pattern can be recognized, for example, in [24, 25, 36, 77], but this list is by no means complete.

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

The sorting heuristics view the minimum feedback edge 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 in [11]. Numerical results are reported on both sparse and dense random graphs where n ranges from 100 to 1000, 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 above mentioned ones is given in [71]. 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 is performed on each component to identify a (hopefully large) acyclic subgraph D; the edges not in D form a feedback edge 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 [65].

3. Exact methods. The published exact methods include (a) dynamic programming, e.g., [80, 88], (b) custom branch and bound methods (or smart enumeration with special exclusion rules), e.g., [28, 34, 50, 69, 72], and (c) integer programming formulations. The latter will be reviewed in the following subsections in detail, since the present paper 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 edge $(i, j) \in E$, and let $c_{i,j} = 0$ if $(i, j) \notin E$. If the cardinality of the feedback edge set is to be minimized, then for each $(i, j) \in E$ we have $c_{i,j} = 1$. If the weighted minimum feedback edge set problem is to be solved, then all $c_{i,j}$ associated with a directed edge equal the weight of the corresponding edge (i, j). Furthermore, 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:

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

Any y that satisfies the **triangle inequalities** (3) must correspond to an ordering [33, 59, 66]. 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., [33, 66], and [65, Ch. 5].

3.2. Integer programming formulation as minimum set cover. An alternative to the formulation of the previous section is the minimum set cover formulation, see for example [72, Eq. (1)] or [9, Sec. 8.4].

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

Here, m denotes the number of edges; w_j are nonnegative weights (often integer); y_j is 1 if edge j is in the feedback edge set, and 0 otherwise; a_{ij} is 1 if edge j participates in cycle i, and 0 otherwise; 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 presolve phase [30, 57, 72], [9, p. 279], or [22, p. 114] (e.g., 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 [38] or SCIP [2] implement these simplifications (and other simplifications as well). After the presolve 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 [81], and such graphs appear in practice, see Section 5.4.

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 [49] 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 [38] 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. In all practical cases encountered so far, only a tractable number of simple cycles has to be enumerated until a minimum feedback edge set is found. 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 call Algorithm 2 with *all* the edges of G as feedback edge set, and with an empty cycle matrix; the algorithm returns the first cycle matrix for $\tilde{P}^{(1)}$. (Other initialization procedures are also possible.)

In iteration k, the optimal solution to the relaxed problem $\tilde{P}^{(k)}$ gives a feedback edge set, and we remove all the edges in this feedback edge 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 edge set $F^{(k)}$ of $G^{(k)}$ using an appropriate heuristic, see Section 2. The proposed algorithm is guaranteed to make progress with *any* feedback edge set but the algorithm is likely to make better progress with an $F^{(k)}$ of small cardinality. Removing the edges in $F^{(k)}$ makes $G^{(k)}$ acyclic, and therefore the associated y yields a feasible solution to P. 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 edge set $F^{(k)}$ contains at least one edge of every cycle in $G^{(k)}$ by definition; therefore, there must be at least one edge $e \in F^{(k)}$ that participates in a cycle. For each edge $e \in F^{(k)}$ we compute the shortest path from the head of e to the tail of e with breadth-first search (BFS). (Although it has not been observed, this procedure based on BFS can potentially lead to poor performance if the edge weight distribution is pathological. It is subject to future research to improve this procedure in such pathological and not yet seen cases.) 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 the edges 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 edge 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 6.5.1 [38] 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 at [6].

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 [6].

Algorithm 1: Finding a minimum feedback edge set based on integer programming and lazy constraint generation

Input: G, a directed graph with m edges and nonnegative edge weights w_j $(j = 1,2,\ldots,m)$
Output: A minimum weight feedback edge 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 edge 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 edge 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 the edges 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 edge 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 edge 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 $A^{(i)}$

4.3. Novelties. The idea of building up an integer program incrementally, by adding constraints to it in a lazy fashion, is not new, see for example Dantzig et al. [19] from 1954. The well-known column generation approach corresponds to this idea but works on the dual

Algorithm 2: Extending the cycle matrix given an arbitrary feedback edge set

Input: G, a directed graph; F, a feedback edge 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 breadth-first search (BFS) in G
    if such a path p exists then
        Turn the path p into a simple cycle s by adding the edge e to p
        Add a new row r to the cycle matrix corresponding to s if r is not already in the matrix
```

problem. Probably the first published paper applying column generation is from 1958 by Ford and Fulkerson [29]. Not surprisingly, state-of-the-art integer programming solvers have high-level API support for implementing such algorithms, see for example `LazyConstraints` in Gurobi 6.5.1 [38].

We are not aware of any published algorithm that would apply lazy constraint generation in the context of the minimum feedback edge set problem. However, the real novelty is in the way *how* the lazy constraint generation is carried out: (1) We apply a greedy heuristic to find a feedback edge set, (2) we then find the tightest simple cycles with breadth-first search that contain this feedback edge set, (3) and finally 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.

5. Computational results.

Pre-solve phase. In the pre-solve phase, we attempt to generate an equivalent but simpler graph than the input. Only the following procedures were applied: splitting into nontrivial SCCs, then iteratively removing runs and 3-edge bypasses, see Hand-coded procedures for common patterns in Appendix A.3 and also Figure 4.

Hardware and software environment. The computations were carried out with the following hardware and software configuration. Processor: Intel(R) Core(TM) i5-4670S CPU at 3.10GHz; 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 [38] was called through its API from Python 2.7.11; the graph library `NetworkX` [41] 1.9.1 was used.

5.1. Cross-checking correctness. This set primarily serves for cross-checking the correctness of our implementation against both the published results and the integer programming approaches of Section 3.1 and 3.2. The properties of these test graphs are given in Table 1.

TABLE 1
Properties of the test graphs for cross-checking correctness. These graphs once used to be a benchmark.

ID	Nodes	Edges	SCCs	Cycles	Optimum	Original source
1	6	30	1	409	15	Complete graph
2	12	21	1	22	2	Pho and Lapidus [72]
3	15	35	3	27	6	Barkley and Motard [7]
4	19	31	1	20	6	Sargent and Westerberg [79]
5	25	32	1	10	3	Christensen and Rudd [14] (‘first’)
6	29	37	1	11	5	Jain and Eakman [47] (HF-alkylation)
7	30	42	1	31	3	Christensen and Rudd [14] (‘second’)
8	41	61	1	103	5	Shannon (Sulfuric acid), see [36]
9	50	79	1	22	8	Jain and Eakman [47] (Vegetable oil)
10	109	163	1	13746	12	Gundersen [35] (Heavy water)
11	32	52	1	187	6	See Appendix A.4

The graphs were taken from Gundersen and Hertzberg [36]: The test problems with ID=1..10 correspond to the problem with the same ID in [36]. (The self-loops were removed from Problem 1 of [36].) Problem 11 was obtained by running the ILP-based edge 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. Although these graphs once used to be a benchmark, they can be solved without any significant difficulties with today's computational power.

5.2. Sparse random graphs. In the $G(n,p)$ Erdős–Rényi random graph model, each edge of an order-n graph is included with probability p independently from every other edge [26]. This model was introduced independently by Edgar Gilbert [32]. 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 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 $c = pn$.

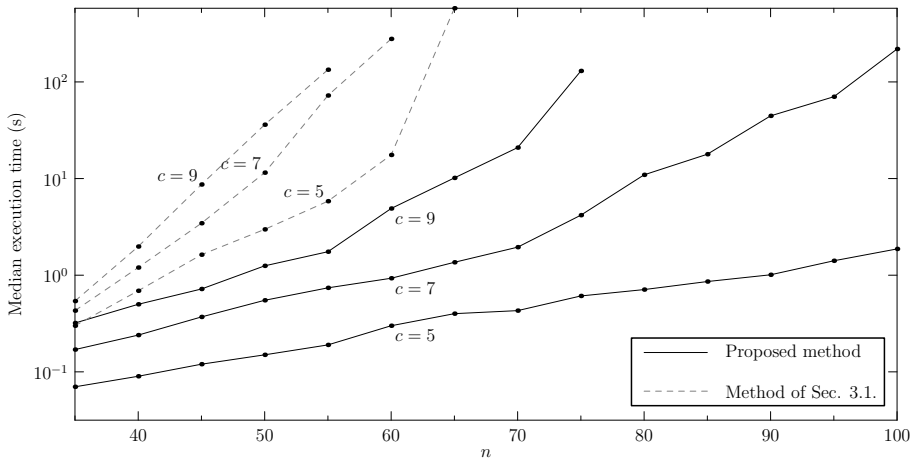

FIG. 1. *Computing the minimum feedback edge set of random graphs, generated according to the Erdős-Rényi model $G(n,p)$ with edge probability $p = \frac{c}{n}$, where the number of nodes n and the parameter c are shown in the figure. Each dot represents the median of the execution times over 101 random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

5.3. 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 polynomial-time approximation scheme (see Sec. 1.1) and custom-tailored cutting plane algorithms have been developed to solve tournaments efficiently (see Sec. 3.1). The sole reason why 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 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 $n < 33$. As expected, the method

of Section 3.1 eventually outperforms the proposed method for random tournaments of size $n \geq 33$, since the method of Section 3.1 was tailored for tournaments.

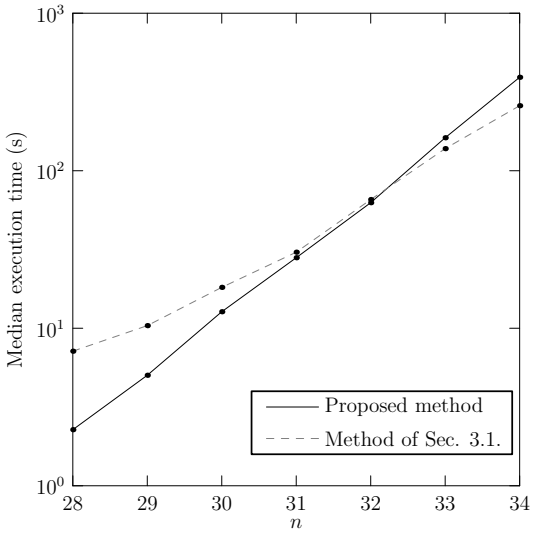

FIG. 2. *Computing the minimum feedback edge set of random tournaments. Each dot represents the median of the execution times over 101 random tournaments. The number of nodes is denoted by n. Solid line: proposed method; dashed line: integer programming formulation of Section 3.1 using triangle inequalities.*

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 Sec. 3.2 for $n > 10$ (it has more than 10^7 simple cycles), and the proposed method extends the minimum set cover approach. Therefore it is reasonable to add the complete directed graph to the test set. However, both the proposed method and the integer programming approach of Sec. 3.1 solve the complete graphs already in the presolve phase even for $n > 10$.

5.4. Challenging sparse graphs. The sparse graphs considered in this section are **intractable** with the integer programming approaches of Section 3.1 and 3.2 due to the sheer number of constraints in the integer programs. In particular, the formulation of Section 3.1, when applied to a graph of n nodes, yields an integer program with $\binom{n}{2}$ binary variables and $2\binom{n}{3}$ constraints. We consider this approach intractable for $n \geq 100$. For each graph we verified that there are more than 10^7 simple cycles in it: $10^7 + 1$ simple cycles were enumerated with Johnson's algorithm [49] before giving up on enumerating all of them. (It is very likely that each of them has several orders of magnitude more simple cycles.) This rules out the integer programming approach of Section 3.2. Finally, exploiting fixed-parameter tractability, the $O(n^4 4^k k^3 k!)$ time algorithm of [13] (Sec. 1.1), is not an option either due to the size of the minimum feedback edge set.

The **generalized de Bruijn graph** $B(n,d)$ is defined as follows [23]. The nodes are labeled $0, 1, \ldots, n-1$, and the directed edge set consists of

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

where d is the degree of the graph. The generalized de Bruijn graphs were first proposed independently by S. M. Reddy and Kuhl [75] and Imase and Itoh [44]. Certain grid network topologies are de Bruijn graphs [83, Sec. 5.6.1.2 and 5.6.2.2]. The distributed hash table

protocol Koorde uses de Bruijn graphs [51]. In bioinformatics, de Bruijn graphs are used for de novo assembly of (short) read sequences into a genome [91].

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 [54]. A node is labeled with a word of length k, $(x_1,\dots,x_k)$, on the alphabet $\Sigma = \{0,\dots,d\}$, $|\Sigma| = d+1$, in which $x_i \neq x_{i+1}$ for $1 \leq i \leq k-1$. There is an edge from a node $x = (x_1,\dots,x_k)$ to all vertices y such that $y = (x_2,\dots,x_k,z)$, $z \in \Sigma$, $z \neq x_k$. The Kautz graph is a good static topology to construct distributed hash table (DHT) schemes [60], for fault-tolerant processor interconnection networks [46], and for multi-OPS optical network topologies [18].

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 [45]: The nodes are labeled $0,1,\dots,n-1$, and there is a directed edge from node i to node j iff:

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

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 [45].

TABLE 2

Computational results on de Bruijn graphs with the proposed method. These graphs are considered intractable with the methods of Sec. 3.1 and Sec. 3.2; exploiting fixed-parameter tractability is not an option either. Except the parameter d, all the data are given for the graphs after the presolve phase, i.e., after removing self-loops.

Nodes	Edges	Parameter d	Optimum	Time (s)
100	296	3	58	22.39
100	396	4	91	1.08
100	492	5	116	1.22
100	590	6	158	8.39
110	326	3	63	1.11
110	436	4	97	34.96
110	544	5	134	3.77
110	650	6	172	2224.37
120	356	3	66	1.8
120	474	4	108	3.27
120	592	5	150	1.19
120	710	6	180	1479.63

6. Conclusions. We have proposed a novel method for solving the minimum feedback edge set problem. In Section 5.2, the proposed method was compared to the alternative integer programming approach of Section 3.1 on 3.7k sparse random graphs of varying size and sparsity. These graphs, although still tractable, push the compared methods to their limits. The proposed method shows significantly better scaling on these sparse random graphs than the alternative approach.

Tournaments trigger, in some sense, the worst-case behavior of the proposed method since the algorithm was meant to be used with *sparse* graphs, whereas tournaments are the orientation of the undirected complete graph. Since the method of Section 3.1 was tailored for tournaments, it eventually outperforms the proposed method for random tournaments of

TABLE 3

Computational results on graphs of Imase and Itoh with the proposed method. These graphs are considered intractable with the methods of Sec. 3.1 and Sec. 3.2; exploiting fixed-parameter tractability is not an option either. Except the parameter d, all the data are given for the graphs after the presolve phase, i.e., after removing self-loops.

Nodes	Edges	Parameter d	Optimum	Time (s)
100	300	3	66	0.36
100	400	4	90	1.30
100	496	5	126	1.54
100	594	6	156	35.18
100	696	7	192	45.42
110	328	3	62	1.14
110	440	4	100	3.79
110	546	5	135	6.91
110	654	6	172	37.56
110	764	7	210	4785.70
120	360	3	72	0.98
120	480	4	114	3428.02

size $n \geq 33$. Nevertheless, for random tournaments of size $n < 33$, the proposed method is faster on average. Let us emphasize again that the proposed method is not meant to be used with tournaments or dense graphs: As discussed in Section 1.1, specialized methods are available for tournaments and dense graphs. The sole reason for testing on tournaments was to study the worst-case behavior of the proposed method, and we consider this worst-case performance satisfactory.

The highly structured sparse graphs of Section 5.4 are intractable with the alternative integer programming approaches of Sections 3.1 and 3.2 due to the sheer number of constraints in the integer program. Exploiting fixed-parameter tractability is not an option either due to the size of the minimum feedback edge set of these challenging graphs. However, it is still tractable to find the minimum feedback edge set of these graphs with the proposed method.

The 4468 test graphs used in this study, together with their minimum feedback edge set, are available in electronic form at [6] as plain text files. 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 at [6]. This contribution aims at establishing a benchmark for future exact algorithms.

Acknowledgement. The research was funded by the Austrian Science Fund (FWF): P27891-N32. Support by the Austrian Research Promotion Agency (FFG) under project number 846920 is gratefully acknowledged.

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

The goal of the algorithm is to find edges that can be safely removed. If the algorithm fails to find such an edge set, no simplification takes place, 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 edges 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 edge 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-edge and v has an additional out-edge, see 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 (i) the edge (u,v), or (ii) the edge (v,u), or (iii) both. The third option is obviously not an optimal solution to break the two-cycle. We now discuss the first two options.

The edge (v,u) cannot participate in any simple cycle other than the two-cycle shown in Figure 3, because u has a single out-edge and that points to v, or alternatively, because v has a single in-edge and that comes from u. However, the edge (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 edge (v,u) cannot yield a strictly lower cost solution than removing (u,v) since removing (u,v) breaks both the two-cycle and all the other cycles in C (if any). The edge (u,v) can be safely removed; the global optimum remains unchanged.

A.2. Rule to identify edges that are safe remove. Our observations made in the previous subsection 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,
- the cost c_2 of making both G' acyclic and breaking also all those cycles of G that can have an edge in G' by removing edges in G' only. Let F' denote such an edge set; this edge 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 edge 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* those cycles of G that have an edge 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 edge 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-edge bypasses, and their corresponding simplified forms are hand-coded, see Figure 4. We assume throughout this paper 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 edge that can break a self-loop; this edge is always removed and added to the feedback edge 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 the 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 right below.

Selecting the induced subgraphs G'. In order 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 edge 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'. Depth first search (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 edge removing algorithm, one after the other in an arbitrary sequence, and starting with $d = 1$ as depth limit for the DFS. If no safe edge set is found at any of the 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 edge set. However, whenever a safe edge set is found, it is removed and added to the feedback edge set. The remaining part of G is split into nontrivial SCCs, the runs and the 3-edge 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 it 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 edge 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 [49] for enumerating simple cycles can be implemented in a lazy fashion [41], that is, it can be aborted after a pre-defined number of simple cycles (e.g. 100 or 1000) 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 edge of G' that can possibly participate in a simple cycle in G, necessarily participates in a simple cycle in H too. 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 an out-edge 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 edge in G' but not all of their edges; 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 edges outside G'. This will ensure that any edge in G' that appears in a cycle in C, will also be involved in a simple cycle in H that has edges 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 edges. For each edge in G that has its initial node (tail) t in G' but its terminal node (head) not in G' (edges "sticking out" of G'), we add the edge (t,u) to G'. Similarly, for each edge that has its terminal node (head) h in G' but its initial node (tail) not in G', we add the edge (v,h) to G'. Finally, we add the edge (u,v). Let H denote the graph that we obtained; G' is obviously an induced subgraph of H.

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

There is a corner case in the above construction of H which 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 edges 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. Edge 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 edge removal algorithm of Appendix A.3. Although it is inefficient to solve these problems with

TABLE 4

The minimum cycle budget and minimum cutoff to solve the test problems with the edge removal algorithm only. Problem 1 is the complete graph and has no safe to remove edges. Problem 11 has no safe to remove edges as expected, see in the text.

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	13746	187	5
11	6	187	187	4

the edge removal algorithm only, the numbers nevertheless show that Problems 2–10 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 Prob-

lem 10 with the edge removal algorithm only. Accordingly, Problem 11 does not have a proper subgraph that has safe to remove edges. See also Figures 6 and 7.

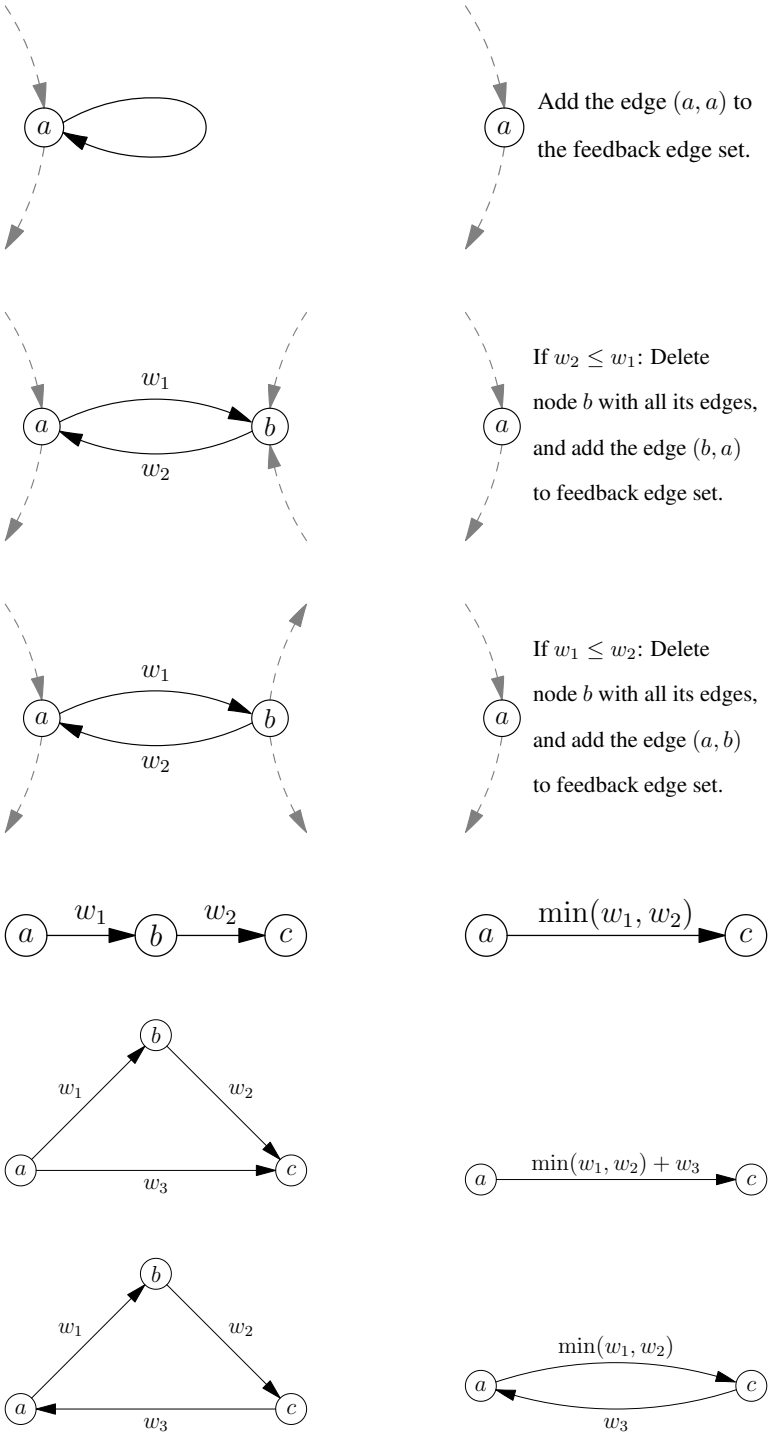

FIG. 4. *Common patterns whose simplification is hand-coded mainly for efficiency reasons. The left column shows the induced subgraphs of G, the right column shows the corresponding simplified form. From top to bottom: (1) removing self-loops, (2) breaking 2-cycles where b has out-degree* 1 *in G, (3)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

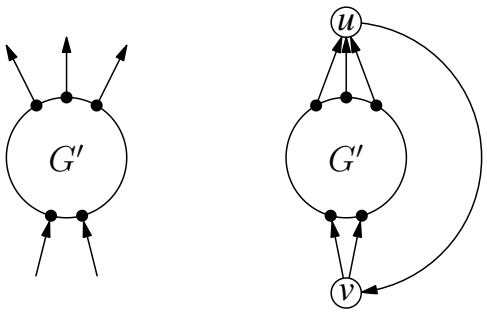

FIG. 5. *Left: The nodes on the boundary of the induced subgraph G' are shown as black dots, together with their edges not in G'. Right: The extended G', the H graph. The nodes u and v, and their edges are fake (not in G).*

References.

[1] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, July 2007.

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

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

[4] Aspen Technology, Inc. Aspen Simulation Workbook, Version Number: V7.1, 2009. Burlington, MA, USA. EO and SM Variables and Synchronization, p. 110.

[5] P. Austrin, R. Manokaran, and C. Wenner. On the NP-hardness of approximating ordering constraint satisfaction problems. In P. Raghavendra, S. Raskhodnikova, K. Jansen, and J. D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 8096 of *Lecture Notes in Computer Science*, pages 26–41. Springer Berlin Heidelberg, 2013.

[6] A. Baharev, 2016. URL https://sdopt-tearing.readthedocs.org. Exact and heuristic methods for tearing.

[7] R. W. Barkley and R. L. Motard. Decomposition of nets. *Chem. Eng. J.*, 3:265–275, 1972.

[8] Bonnie Berger and Peter W. Shor. Approximation alogorithms for the maximum acyclic subgraph problem. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 236–243, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[9] Lorenz T. Biegler, Ignacio E. Grossmann, and Arthur W. Westerberg. *Systematic Methods of Chemical Process Design*. Prentice Hall PTR, Upper Saddle River, NJ, 1997.

[10] N. L. Book and W. F. Ramirez. Structural analysis and solution of systems of algebraic design equations. *AIChE Journal*, 30(4):609–622, 1984.

[11] F. J. Brandenburg and K. Hanauer. Sorting heuristics for the feedback arc set problem. Technical Report MIP-1104, Department of Informatics and Mathematics, University of Passau, Germany, Feb 2011.

[12] M. Charikar, K. Makarychev, and Y. Makarychev. On the advantage over random for maximum acyclic subgraph. FOCS, pages 625–633. IEEE Computer Society, 2007.

[13] Jianer Chen, Yang Liu, Songjian Lu, Barry O'sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):1–19, 2008.

[14] J. H. Christensen and D. F. Rudd. Structuring design computations. *AIChE Journal*, 15: 94–100, 1969.

[15] James H. Christensen. The structuring of process optimization. *AIChE Journal*, 16(2): 177–184, 1970.

[16] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[17] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, USA, 3rd edition, 2009.

[18] David Coudert, Afonso Ferreira, and Xavier Muñoz. *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*, chapter OTIS-Based multi-hop multi-OPS lightwave networks, pages 897–910. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[19] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.
[20] Narsingh Deo. *Graph theory with applications to engineering and computer science*. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1974.
[21] A. C. Dimian, C. S. Bildea, and A. A. Kiss. Chapter 3: Steady-state flowsheeting. In *Integrated Design and Simulation of Chemical Processes*, volume 35 of *Computer-Aided Chemical Engineering*. Elsevier Amsterdam, 2nd edition, 2014.
[22] R. G. Downey and M. R. Fellows. In D. Gries and F. B. Schneider, editors, *Fundamentals of Parameterized Complexity*, Texts in Computer Science. Springer-Verlag London, 2013.
[23] D. Z. Du and F. K. Hwang. Generalized de Bruijn digraphs. *Networks*, 18(1):27–38, 1988.
[24] P. Eades and X. Lin. A new heuristic for the feedback arc set problem. *Australasian Journal of Combinatorics*, pages 15–25, 1995.
[25] Peter Eades, Xuemin Lin, and W.F. Smyth. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323, 1993.
[26] P. Erdős and A. Rényi. On random graphs I. *Publicationes Mathematicae*, 6:290–297, 1959.
[27] G. Even, J. (Seffi) Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.
[28] F. V. Fomin and D. Kratsch. Measure & Conquer. In *Exact Exponential Algorithms*, Texts in Theoretical Computer Science. An EATCS Series, pages 101–124. Springer Berlin Heidelberg, 2010.
[29] L. R. Ford and D. R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.
[30] R. S. Garfinkel and G. L. Nemhauser. *Integer Programming*. New York: Wiley, 1972.
[31] P. L. Genna and R. L. Motard. Optimal decomposition of process networks. *AIChE Journal*, 21(4):656–663, 1975.
[32] E. N. Gilbert. Random graphs. *Ann. Math. Statist.*, 30:1141–1144, 1959.
[33] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, 1984.
[34] G. Guardabassi. A note on minimal essential sets. *Circuit Theory, IEEE Transactions on*, 18(5):557–560, 1971.
[35] T. Gundersen. *Decomposition of large scale chemical engineering systems*. PhD thesis, Dep. of Chem. Eng., Univ. of Trondheim, Norway, 1982.
[36] T. Gundersen and T. Hertzberg. Partitioning and Tearing of Networks Applied to Process Flowsheeting. *Modeling, Identification and Control*, 4(3):139–165, 1983.
[37] Prem K. Gupta, Arthur W. Westerberg, John E. Hendry, and Richard R. Hughes. Assigning output variables to equations using linear programming. *AIChE Journal*, 20(2): 397–399, 1974.
[38] Gurobi. Gurobi Optimizer Version 6.0. Houston, Texas: Gurobi Optimization, Inc., May 2015. (software program). http://www.gurobi.com, 2014.
[39] V. Guruswami, R. Manokaran, and P. Raghavendra. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *Foundations of Computer Science, 2008. FOCS '08. IEEE 49th Annual IEEE Symposium on*, pages 573–582, 2008.
[40] V. Guruswami, J. Håstad, R. Manokaran, P. Raghavendra, and M. Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM Journal on Computing*, 40(3):878–914, 2011.
[41] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure,

dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[42] R. Hernandez and R. W. H. Sargent. A new algorithm for process flowsheeting. *Computers & Chemical Engineering*, 3(14):363–371, 1979.

[43] V. Hlaváček. Analysis of a complex plant-steady state and transient behavior. *Computers & Chemical Engineering*, 1(1):75 – 100, 1977.

[44] M. Imase and M. Itoh. Design to minimize diameter on building-block network. *IEEE Transactions on Computers*, C-30(6):439–442, 1981.

[45] M. Imase and M. Itoh. A design for directed graphs with minimum diameter. *IEEE Transactions on Computers*, C-32(8):782–784, 1983.

[46] Makoto Imase, Terunao Soneoka, and Keiji Okada. Fault-tolerant processor interconnection networks. *Systems and Computers in Japan*, 17(8):21–30, 1986.

[47] Y. V. S. Jain and J. M. Eakman. Identification of process flow networks. In *AIChE 68th National Meeting, Houston, TX*, 1971.

[48] David S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 38–49, New York, NY, USA, 1973. ACM.

[49] Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[50] R. Kaas. A branch and bound algorithm for the acyclic subgraph problem. *European Journal of Operational Research*, 8(4):355–362, 1981.

[51] M. Frans Kaashoek and David R. Karger. *Peer-to-Peer Systems II: Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003. Revised Papers*, chapter Koorde: A Simple Degree-Optimal Distributed Hash Table, pages 98–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[52] Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.

[53] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[54] W. H. Kautz. Bounds on directed (d,k) graphs. theory of cellular logic networks and machines. Technical Report AFCRL-68-0668 Final report, 20-28, 1968.

[55] Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 95–103, New York, NY, USA, 2007. ACM.

[56] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 767–775, New York, NY, USA, 2002. ACM.

[57] W. Lee and D. F. Rudd. On the ordering of recycle calculations. *AIChE Journal*, 12(6): 1184–1190, 1966.

[58] W. Lee, J. H. Christensen, and D. F. Rudd. Design variable selection to simplify process calculations. *AIChE Journal*, 12(6):1104–1115, 1966.

[59] H. W. Lenstra. The acyclic subgraph problem. Technical Report BW 26/73, Faculteit der Wiskunde en Natuurwetenschappen, Mathematisch Centrum, Amsterdam, July 1973. URL http://hdl.handle.net/1887/2116.

[60] Dongsheng Li, Xicheng Lu, and Jinshu Su. *Network and Parallel Computing: IFIP International Conference, NPC 2004, Wuhan, China, October 18-20, 2004. Proceedings*, chapter Graph-Theoretic Analysis of Kautz Topology and DHT Schemes, pages 308–315. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[61] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*,

13(4):383–390, 1975.

[62] C. L. Lucchesi. *A minimax equality for directed graphs*. PhD thesis, University of Waterloo, Ontario, 1976.

[63] C. L. Lucchesi and D. H. Younger. A minimax theorem for directed graphs. *J. London Math. Soc.*, 2(17):369–374, 1978.

[64] Richard S. H. Mah. 4 - Computation Sequence in Process Flowsheet Calculations. In Richard S. H. Mah, editor, *Chemical Process Structures and Information Flows*, pages 125–183. Butterworth-Heinemann, 1990.

[65] Rafael Martí and Gerhard Reinelt. *The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization*, volume 175 of *Applied Mathematical Sciences*. Springer-Verlag Berlin Heidelberg, 2011.

[66] J. E. Mitchell and B. Borehers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In H. Frenk, K. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, volume 33 of *Applied Optimization*, chapter 14, pages 349–366. Springer-Science+Business Media, B.V.; Dordrecht, The Netherlands, 2000.

[67] J. M. Montagna and O. A. Iribarren. Optimal computation sequence in the simulation of chemical plants. *Computers & Chemical Engineering*, 12(1):71–79, 1988.

[68] Rodolphe L. Motard and Arthur W. Westerberg. Exclusive tear sets for flowsheets. *AIChE Journal*, 27:725–732, 1981.

[69] T. Orenstein, Z. Kohavi, and I. Pomeranz. An optimal algorithm for cycle breaking in directed graphs. *Journal of Electronic Testing*, 7(1-2):71–81, 1995.

[70] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[71] S. Park and S. B. Akers. An efficient method for finding a minimal feedback arc set in directed graphs. In *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*, volume 4, pages 1863–1866, May 1992.

[72] T. K. Pho and L. Lapidus. Topics in computer-aided design: Part I. An optimum tearing algorithm for recycle systems. *AIChE Journal*, 19(6):1170–1181, 1973.

[73] Vijaya Ramachandran. Finding a minimum feedback arc set in reducible flow graphs. *Journal of Algorithms*, 9(3):299–313, 1988.

[74] J. R. Roach, B. K. O'Neill, and D. A. Hocking. A new synthetic method for stream tearing in process systems analysis. *Chemical Engineering Communications*, 161(1): 1–14, 1997.

[75] D. K. Pradhan S. M. Reddy and J. G. Kuhl. Directed graphs with minimal diameter and maximal connectivity. Technical report, School of Engineering, Oakland University, Rochester MI, USA, July 1980.

[76] R. Saket and M. Sviridenko. New and improved bounds for the minimum set cover problem. In A. Gupta, K. Jansen, J. Rolim, and R. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 7408 of *Lecture Notes in Computer Science*, pages 288–300. Springer Berlin Heidelberg, 2012.

[77] G. Sander. Graph layout for applications in compiler construction. *Theoretical Computer Science*, 217(2):175–214, 1999.

[78] R. W. H. Sargent. The decomposition of systems of procedures and algebraic equations. In G. A. Watson, editor, *Numerical Analysis*, volume 630 of *Lecture Notes in Mathematics*, pages 158–178. Springer Berlin Heidelberg, 1978.

[79] R. W. H. Sargent and A. W. Westerberg. Speed-up in chemical engineering design. *Transactions of the Institution of Chemical Engineers*, 42a:190–197, 1964.

[80] R. W. H. Sargent and A. W. Westerberg. Speed-up in chemical engineering design.

Trans. Instn. Chem. Engrs., 42:190–197, 1964.

[81] Benno Schwikowski and Ewald Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics*, 117(13):253–265, 2002.

[82] P.D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.

[83] Ljiljana Spadavecchia. *A Network-based Asynchronous Architecture for Cryptographic Devices*. PhD thesis, University of Edinburgh; College of Science and Engineering; School of Informatics, 2006. URL http://hdl.handle.net/1842/860.

[84] M. A. Stadtherr and E. S. Wood. Sparse matrix methods for equation-based chemical process flowsheeting–I: Reordering phase. *Computers & Chemical Engineering*, 8(1): 9–18, 1984.

[85] M. A. Stadtherr, W. A. Gifford, and L. E. Scriven. Efficient solution of sparse sets of design equations. *Chemical Engineering Science*, 29(4):1025–1034, 1974.

[86] Mark A. Stadtherr. Maintaining sparsity in process design calculations. *AIChE Journal*, 25(4):609–615, 1979.

[87] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1: 146–160, 1972.

[88] R. S. Upadhye and E. A. Grens. An efficient algorithm for optimum decomposition of recycle systems. *AIChE Journal*, 18:533–539, 1972.

[89] G. V. Varma, K. H. Lau, and D. L. Ulrichson. A new tearing algorithm for process flowsheeting. *Computers & Chemical Engineering*, 17(4):355–360, 1993.

[90] A. W. Westerberg and F. C. Edie. Computer-Aided Design, Part 2 An approach to convergence and tearing in the solution of sparse equation sets. *Chem. Eng. J.*, 2(1): 17–25, 1971.

[91] Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18:821–829, 2008.

[92] L. Zhou, Z. Han, and K. Yu. A new strategy of net decomposition in process simulation. *Computers & Chemical Engineering*, 12(6):581–588, 1988.

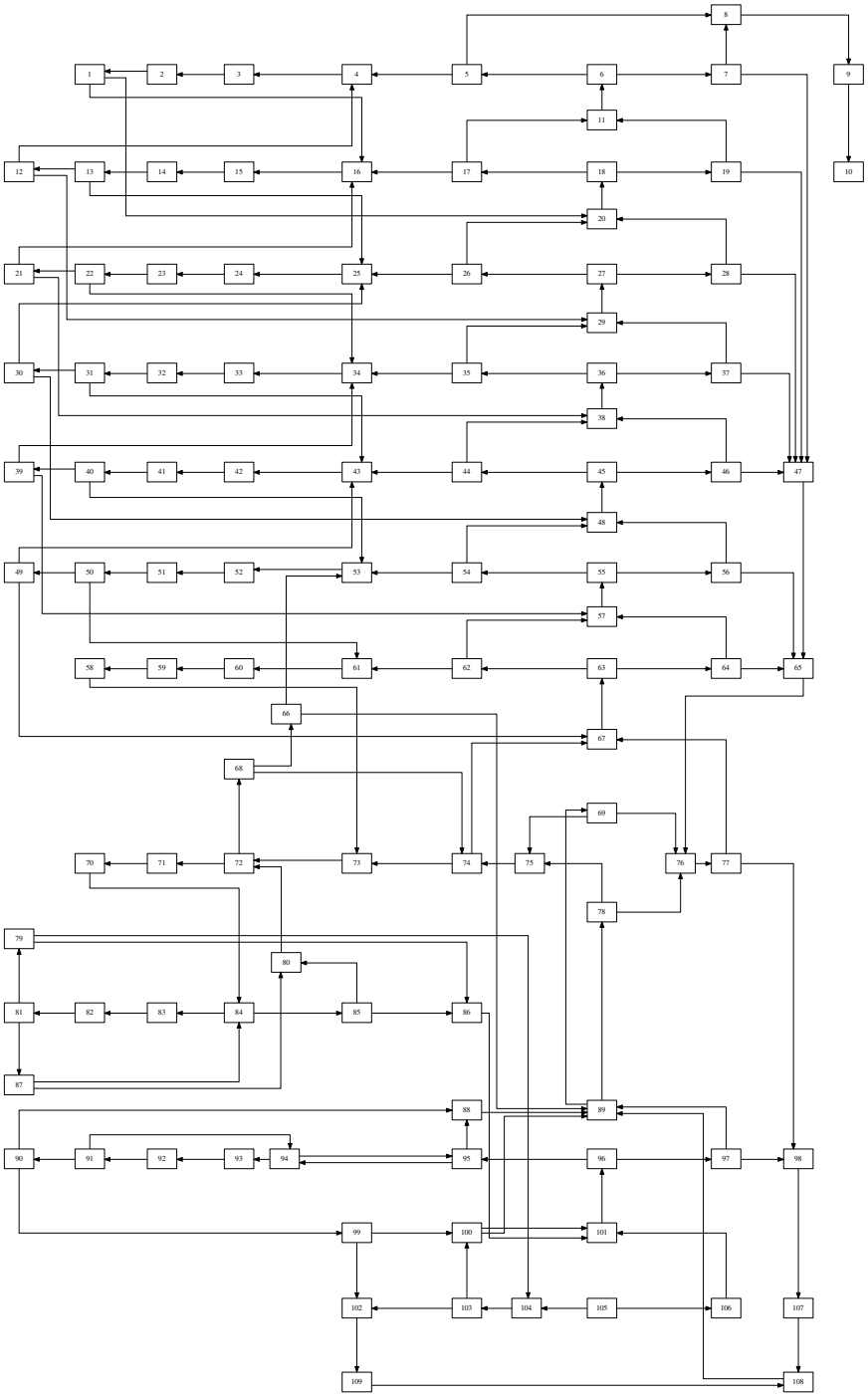

FIG. 6. *Problem 10, origin: Gundersen [35] (Heavy water). This graph has 109 nodes, 163 edges, 13746 simple cycles, and the cardinality of the minimum feedback edge set is 12.*

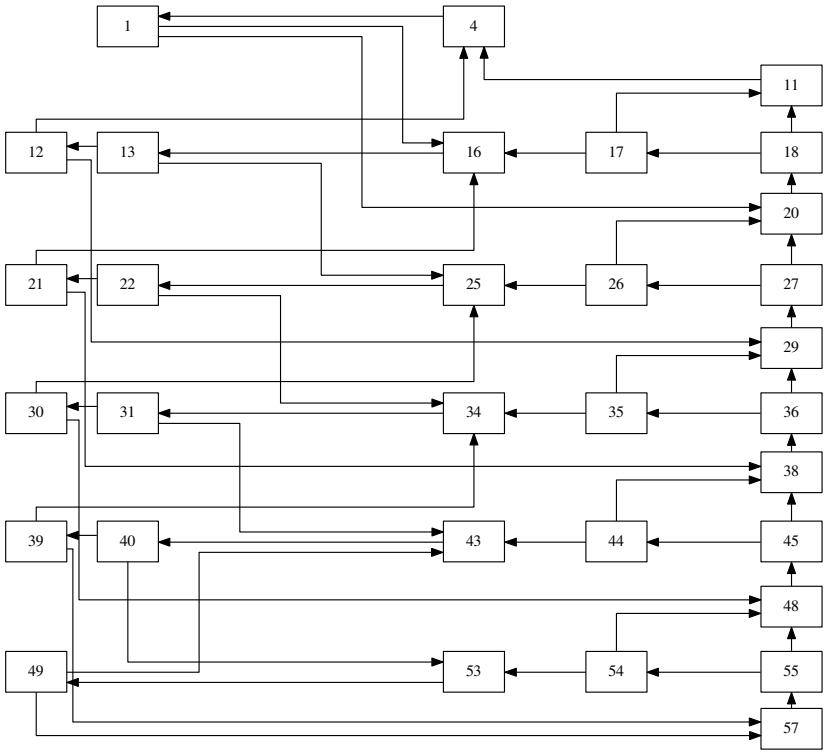

FIG. 7. *Problem 11, derived from Problem 10 as discussed in Appendix A.4. This graph has 32 nodes, 52 edges, 187 simple cycles, and the cardinality of the minimum feedback edge set is 6.*