A Naive Algorithm for Feedback Vertex Set*

Yixin Cao

Department of Computing, Hong Kong Polytechnic University, Hong Kong, China
yixin.cao@polyu.edu.hk

Abstract

Given a graph on n vertices and an integer k, the feedback vertex set problem asks for the deletion of at most k vertices to make the graph acyclic. We show that a greedy branching algorithm, which always branches on an undecided vertex with the largest degree, runs in single-exponential time, i.e., $O(c^k \cdot n^2)$ for some constant c.

1998 ACM Subject Classification F.2.2 Analysis of Algorithms and Problem Complexity, G.2.2 Graph Theory

Keywords and phrases greedy algorithm, analysis of algorithms, branching algorithm, parameterized computation, graph modification problem

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.1

1 Introduction

All graphs in this paper are undirected and simple. A graph G is given by its vertex set $V(G)$ and edge set $E(G)$, whose cardinalities will be denoted by n and m respectively. A set V_- of vertices is a *feedback vertex set* of graph G if $G - V_-$ is acyclic, i.e., being a forest. Given a graph G and an integer k, the feedback vertex set problem asks whether G has a feedback vertex set of at most k vertices.

The feedback vertex set problem was formulated from artificial intelligence, where a feedback vertex set is also called a *loop cutset*. For each instance of the constraint satisfaction problem one can define a constraint graph, and it is well known that the problem can be solved in polynomial time when the constraint graph is a forest [11]. Therefore, one way to solve the constraint satisfaction problem is to find first a minimum feedback vertex set of the constraint graph, enumerate all possible assignments on them, and then solve the remaining instance. Given an instance I of the constraint satisfaction problem on p variables, and a feedback vertex set V_- of the constraint graph, this approach can be implemented in $O(p^{|V_-|} \cdot |I|^{O(1)})$ time [7]. A similar application was found in Bayesian inference, also in artificial intelligence [18]; more updated material can be found in the thesis of Bidyuk [3].

The feedback vertex set problem is NP-hard [16]. The aforementioned approach for solving the constraint satisfaction problem only makes sense when $|V_-|$ is fairly small. This motivates the study of parameterized algorithms for the feedback vertex set problem, i.e., algorithms that find a feedback vertex set of size at most k in time $f(k) \cdot n^{O(1)}$. Since earlier 1990s, a chain of parameterized algorithms have been reported in literature; currently the best known $f(k)$ is 3.62^k [5, 15]. We refer to [5] for a complete list of published results. Instead of providing a new and improved algorithm, this paper considers a naive branching algorithm that *should have been discovered decades ago*.

* Supported in part by the National Natural Science Foundation of China (NSFC) under grant 61572414, the Hong Kong Research Grants Council (RGC) under grants 25202615 and 15201317, and the European Research Council (ERC) under grant 280152.

A trivial branching algorithm will work as follows. It picks a vertex and branches on either including it in the solution V_- (i.e., deleting it from G), or marking it "undeletable," until the remaining graph is already a forest. This algorithm however takes $O(2^n)$ time. A (rather informal) observation is that a vertex of a larger degree has a larger chance to be in a minimum feedback vertex set, thereby inspiring a two-phase greedy algorithm for the problem. If there are undecided vertices of degree larger than two after some preprocessing, then it always branches on an undecided vertex with the largest degree. Believe it or not, this greedy algorithm already beats most previous algorithms for this problem.

▶ **Theorem 1.1.** *The greedy algorithm can be implemented in $O(8^k \cdot n^2)$ time.*

The use of the observations on degrees in solving the feedback vertex set problem is quite natural. Indeed, the research on parameterized algorithms and that on approximation algorithms for the feedback vertex set problem have undergone a similar process. Early work used the cycle packing-covering duality, and hence ended with $O((\log k)^{O(k)} \cdot n^{O(1)})$-time parameterized algorithms [19] and $O(\log n)$-ratio approximation algorithms [8], respectively, while the first 2-approximation algorithm uses a similar greedy approach on high-degree vertices [1]. Indeed, all the four slightly different 2-approximation algorithms for this problem are based on similar degree observations [6, 12]. So is the quadratic kernel of Thomassé [20]. There is also an $O(4^k \cdot n)$-time randomized algorithm [2] based on this idea. Our greedy branching algorithm can be viewed as the de-randomization of this randomized algorithm.

For a reader familiar with parameterized algorithms of the feedback vertex set problem, Theorem 1.1 may sound somewhat surprising. Deterministic single-exponential algorithms for the feedback vertex set problem had been sedulously sought, before finally discovered in 2005. With so many different techniques, some very complicated, having been tried toward this end,[1] it is rather interesting that the goal can be achieved in such a naive way.

The significance of single-exponential algorithms for the feedback vertex set problem lies also in the theoretical interest, for which let us put it into context.

Together with the vertex cover problem (finding a set V_- of at most k vertices of a graph G such that $G - V_-$ is edgeless), the feedback vertex set problem is arguably the most studied parameterized problem. However, a simple $O(2^k \cdot (m + n))$-time algorithm for vertex cover was already known in 1980s [17]. For this difference there is a quick and easy explanation from the aspect of graph modification problems [16, 4]. Vertex deletion problems ask for the deletion of a minimum set of vertices from a graph to make it satisfy specific properties. The vertex cover problem and the feedback vertex set problem are precisely vertex deletion problems to, respectively, the edgeless graphs and acyclic graphs, i.e., forests. The obstruction (forbidden induced subgraph) for the edgeless graphs is an single edge, the simplest one that is nontrivial. On the other hand, the obstructions for forests are all cycles, which may be considered the simplest of all those infinite obstructions, for most of which single-exponential algorithms are quite nontrivial, if possible at all.

The problems vertex cover and feedback vertex set are also known as planar-$\mathcal{F}$-deletion problems, which, given a graph G, a set $\mathcal{F}$ of graphs of which at least one is planar, ask for a minimum set of vertices whose deletion make the graph H-minor-free for every $H \in \mathcal{F}$ [9]. They correspond to the cases with $\mathcal{F} = \{K_2\}$ and $\mathcal{F} = \{K_3\}$ respectively. Recently, Fomin et al. [10] and Kim et al. [14] showed that all planar-$\mathcal{F}$-deletion problems can be solved in single-exponential time. Yet another way to connect the vertex cover problem and the

[1] To date, the number of parameterized algorithms for feedback vertex set published in literature exceeds any other single problem, including the more famous vertex cover problem.

Algorithm naive-fvs(G, k, F)
INPUT: a graph G, an integer k, and a set $F \subseteq V(G)$ inducing a forest.
OUTPUT: a feedback vertex set $V_- \subseteq V(G) \setminus F$ of size $\leq k$ or "NO."

0. **if** $k < 0$ **then return "NO"**; **if** $V(G) = \emptyset$ **then return** $\emptyset$;
1. **if** a vertex v has degree less than two **then**
   return naive-fvs$(G - \{v\}, k, F \setminus \{v\})$;
2. **if** a vertex $v \in V(G) \setminus F$ has two neighbors in the same component of $G[F]$ **then**
   $X \leftarrow$ **naive-fvs**$(G - \{v\}, k - 1, F)$;
   return $X \cup \{v\}$;
3. pick a vertex v from $V(G) \setminus F$ with the maximum degree;
4. **if** $d(v) = 2$ **then**
4.1. $X \leftarrow \emptyset$;
4.2. **while** there is a cycle C in G **then**
   take any vertex x in $C \setminus F$;
   add x to X and delete it from G;
4.3. **if** $|X| \leq k$ **then return** X; **else return** "NO";
5. $X \leftarrow$ **naive-fvs**$(G - \{v\}, k - 1, F)$; \\ *case 1:* $v \in V_-$.
   if X is not "NO" **then return** $X \cup \{v\}$;
6. **return naive-fvs**$(G, k, F \cup \{v\})$. \\ *case 2:* $v \notin V_-$.

Figure 1 A simple algorithm for feedback vertex set branching in a greedy manner. As we will see in the end of this section, we may terminate the recursive calls after certain number of steps. This leads to a search tree of bounded depth.

feedback vertex set problem is that a graph has treewidth zero if and only if it is edgeless, and treewidth at most one if and only if it is a forest.

2 The algorithm

There is no secret in our algorithm, which is presented in Figure 1, except the recursive form and an extra input F, the set of "undeletable" vertices. We say that (G, k, F), where $F \subseteq V(G)$ and the solution can only be picked from $V(G) \setminus F$, is an *extended instance*; note that to make such an extended instance nontrivial, F needs to induce a forest. (Indeed, the solution in the original loop cutset problem has to be selected from "allowed" vertices, which is exactly the case F comprising all vertices that are not allowed.) The algorithm can be viewed as two parts, the first (steps 1–4) applying some simple operations when the situation is simple and clear, while the second (steps 5 and 6) trying both possibilities on whether a vertex v is in a solution. The operations in the first part are called reductions in the parlance of parameterized algorithms. The three we use here are standard and well-known,[2] and their correctness is straightforward; see, e.g., [5].

[2] For the reader familiar with related algorithms, our reduction steps may seem slightly different from those in literature. First of all, unlike most algorithms for the problem, our algorithm does not involve multiple edges. We believe it is simpler to keep the graph simple. As a result, we are not able to eliminate all vertices of degree two: The common way to dispose of a vertex v of degree-2 is to delete v and add an edge between its two neighbors, so called *smoothening*. Smoothening a vertex whose two neighbors were already adjacent would introduce parallel edges. Noting that there always exists an optimal solution avoiding v, one may move v into F [5], but we prefer the current form because it is simpler and easier to analyze. It is also easier to be extended in Section 3.

▶ **Lemma 2.1.** *Calling algorithm* `naive-fvs` *with* $(G, k, \emptyset)$ *solves the instance* (G, k) *of the feedback vertex set problem.*

Proof. The two termination conditions in step 0 are clearly correct. For each recursive call in steps 1 and 2, we show that the original instance is a yes-instance if and only if the new instance is a yes-instance. Note that no vertex is moved to F in these two steps. In step 1, the vertex v is not in any cycle, and hence it can be avoided by any solution. In step 2, there is a cycle consisting of the vertex v and vertices in F (any path connecting these two vertices in $G[F]$), and hence any solution has to contain v.

To argue the correctness of step 4, we show that the solution found in step 4 is optimal. Let c be the number of components in G and ℓ the size of optimal solutions. Note that every vertex in a solution has degree two, and hence after deleting ℓ vertices the graph has $n - \ell$ vertices and at least $m - 2\ell$ edges. Moreover, deleting vertices from an optimal solution will not decrease the number of components of the graph, we have $(n - \ell) - (m - 2\ell) \geq c$. Hence, $\ell \geq m - n + c$, and showing $|X| = m - n + c$ would finish the task. Deleting a vertex of degree 2 from a cycle never increases the number of components. Also note that a graph of c components contains a cycle if and only if it has more than $n - c$ edges. Therefore, the while loop in step 4 would be run exactly $m - n + c$ iterations: After deleting $m - n + c$ vertices, each of degree two when deleted, the remaining graph has $2n - m - c$ vertices and $2n - m - 2c$ edges, which has to be a forest of c trees.

The last two steps are trivial: If there is a solution containing v, then it is found in step 5; otherwise, step 6 always gives the correct answer. ◀

We now analyze the running time of the algorithm, which is simple but nontrivial. The execution of the algorithm can be described as a search tree in which each node corresponds to two extended instances of the problem, the *entry instance* and the *exit instance*. The entry instance of the root node is $(G, k, \emptyset)$. The exit instance of a node is the one after steps 0–2 have been exhaustively applied on the entry instance. If step 5 is further called, then two children nodes are generated, with entry instances $(G - \{v\}, k - 1, F)$ and $(G, k, F \cup \{v\})$ respectively. (Note that the second child may not be explored by the algorithm, but this is not of our concern.) A leaf node of the search tree returns either a solution or "NO."

It is clear that each node can be processed in polynomial time, and thus the focus of our analysis is to bound the number of nodes in the search tree. Since the tree is binary, it suffices to bound its depth. We say that a path from the root of the search tree to a leaf node is an *execution path*. Let us fix an arbitrary execution path in the search tree of which the leaf node returns a solution V_-, and let F' denote all the vertices moved into F by step 6 in this execution path. The length of this execution path is at most $|V_-| + |F'|$: Each non-leaf node puts at least one vertex to V_- or F. We are allowed to put at most k vertices into V_-, i.e., $|V_-| \leq k$, and hence our task in the rest of this section is to bound $|F'|$.

Let us start from some elementary facts on trees. Any tree T satisfies

$$\sum_{v \in V(T)} d(v) = 2|E(T)| = 2|V(T)| - 2 \qquad \text{and} \qquad \sum_{v \in V(T)} (d(v) - 2) = -2.$$

Let L denote the set of leaves of T, and V_3 the set of vertices of degree at least three. If $V(T) \geq 2$, then $|L| \geq 2$ and

$$-2 = \sum_{v \in L} (d(v) - 2) + \sum_{v \in V(T) \setminus L} (d(v) - 2) = \sum_{v \in L} (-1) + \sum_{v \in V_3} (d(v) - 2) = \sum_{v \in V_3} (d(v) - 2) - |L|.$$

Hence

$$\sum_{v \in V_3} (d(v) - 2) = |L| - 2. \tag{1}$$

The implication of (1) for our problem is that the more large-degree vertices (V_3) in the final forest $G - V_-$, the more leaves (L) it has. Every vertex $u \in F'$ will be in the forest. Since its original degree is at least three, either its degree is decreased to two or less, or there must be some leaves produced to "balance the equation (1)." On the other hand, however, every vertex has degree at least two when u is moved to F. Therefore, if it is the second case, the leaves have to be "produced" in later steps. The requirement of degree decrements is decided by the degree of u, and can be satisfied by vertices deleted later, whose degrees cannot be larger than that of u. This informal observation would enable us to derive the desired lower bound on $|F'|$.

The following invariants will be used in our formal analysis.

Invariant 1 : During the algorithm, the degree of no vertex can increase.

Invariant 2 : When a recursive call is made in step 5 or 6, there is no vertex of degree 0 or 1 in the graph.

This algorithm never directly deletes any edge, and thus the degree of a vertex decreases only when some of its neighbors are deleted from the graph,—we are talking about the degree in the whole graph G, so moving a vertex to F does not change the degree of any vertex. In particular, only steps 1, 2, 4, and 5 can decrease the degree of vertices. By Invariant 2, after a vertex is moved to F, step 1 cannot be called before step 2, 4, or 5. In other words, the degree of a vertex in F decreases only after some vertex put into V_-. We can *attribute* them to vertices V_- as follows.

For a vertex $v \in V_- \cup F'$, we use $d^*(v)$ to denote the degree of v at the moment it is deleted from the graph and put into V_- (step 2, 4, or 5) or moved into F (step 6). Note that $d_G(v) \geq d^*(v)$ by Invariant 1, and $d^*(v) \geq 3$ when $v \in F'$. Let x_1, x_2, …, $x_{|V_-|}$ be the vertices in V_-, in the order of them being put into V_-, and let (G_i, k_i, F_i) be the exit instance in the node of the search tree corresponding to x_i.

▶ **Definition 2.2.** We say that the decrements of the degree of a vertex $u \in F'$ from $d^*(u)$ to 2 are *effective*, and an effective decrement is *incurred by* $x_i \in V_-$ if it happens between deleting x_i and x_{i+1}, or after deleting $x_{|V_-|}$ if $i = |V_-|$. Let $\delta(u, x_i)$ denote the number of effective decrements of u incurred by x_i.

Note that $\delta(u, x_i)$ may be larger than 1. It is worth stressing that we do not count the degree decrements of u before it is moved into F. Therefore, $\delta(u, x_i)$ can be positive only when u is in F when x_i is deleted, i.e., $u \in F_i$ and hence $d_{G_i}(u) \geq 2$:

$$\delta(u, x_i) = \begin{cases} d_{G_i}(u) - \max\{d_{G_{i+1}}(u), 2\} & \text{when } u \in F_i, \\ 0 & \text{otherwise.} \end{cases}$$

▶ **Proposition 2.3.** *For any $u \in F'$ and $x_i \in V_-$, if $\delta(u, x_i) > 0$ then $d^*(u) \geq d^*(x_i)$.*

First, we bound the total number of effective decrements incurred by x_i for each $x_i \in V_-$.

▶ **Lemma 2.4.** *For each $x_i \in V_-$, it holds $\sum_{u \in F'} \delta(u, x_i) \leq d^*(x_i)$.*

Proof. Recall that all effective decrements incurred by x_i happen after deleting x_i from G_i. If $d_{G_i}(v) > 2$ for every vertex $v \in N_{G_i}(x_i)$, then the deletion of x_i will not make the degree

of any vertex smaller than two. Therefore, step 1 will not be called before putting the next vertex into V_-. The degree of each vertex in $N_{G_i}(x_i)$ decreases by one, and the total number of effective decrements incurred by x_i is thus at most $d^*(x_i)$.

In the rest $d_{G_i}(v) = 2$ for some $v \in N_{G_i}(x_i)$, and it becomes 1 with the deletion of x_i. This decrement is not effective, but it will trigger step 1, which may subsequently lead to effective decrements. Let d denote the number of degree-2 neighbors of x_i in G_i. After the deletion of x_i, all of them have degree one, and there is no other vertex having degree one in $G_i - \{x_i\}$ (Invariant 2). We consider the application of step 1, and let x be the vertex deleted. If the only neighbor of x has degree two when this step is executed, then its degree becomes 1 after the deletion of x, and hence the number of degree-1 vertices is not changed. Otherwise, there is one less vertex of degree 1 but there may be one effective decrement (only when the only neighbor of x is in F and has degree at least three). Therefore, when step 1 is no longer applicable, the total number of effective decrements is at most $d^*(x_i) - d + d = d^*(x_i)$. ◀

We are now ready to bound the number of calls of step 6 made in this execution path, i.e., the size of F', by the size of V_-. This is exactly the place the greedy order of branching plays the magic.

▶ **Lemma 2.5.** *In an execution path that leads to a solution, $|F'| \leq 3|V_-|$.*

Proof. Since this execution path leads to a solution, all vertices must be deleted from the graph at the end of the path. In the algorithm, a vertex in F can only be deleted from the graph in step 1, when the degree of the vertex has to be one or zero. On the other hand, $d^*(u) \geq 3$. Thus, all the $d^*(u) - 2$ effective decrements must have happened on this vertex, i.e., $\sum_{v \in V_-} \delta(u, v) = d^*(u) - 2$. Putting everything together, we have

$$
\begin{aligned}
|V_-| = \sum_{v \in V_-} 1 &= \sum_{v \in V_-} \frac{d^*(v)}{d^*(v)} \\
&\geq \sum_{v \in V_-} \frac{1}{d^*(v)} \sum_{u \in F'} \delta(u, v) && \text{(Lemma 2.4)} \\
&= \sum_{v \in V_-} \sum_{u \in F'} \frac{\delta(u, v)}{d^*(v)} \\
&\geq \sum_{v \in V_-} \sum_{u \in F'} \frac{\delta(u, v)}{d^*(u)} && \text{(Proposition 2.3)} \\
&= \sum_{u \in F'} \frac{1}{d^*(u)} \sum_{v \in V_-} \delta(u, v) \\
&= \sum_{u \in F'} \frac{d^*(u) - 2}{d^*(u)} \\
&\geq \sum_{u \in F'} \frac{1}{3} && (d^*(u) \geq 3) \\
&= \frac{|F'|}{3},
\end{aligned}
$$

and the proof is complete. ◀

▶ **Theorem 2.6.** *Algorithm* `naive-fvs` *can be implemented in $O(16^k \cdot n^2)$ time to decide whether a graph G has a feedback vertex set of size at most k.*

Proof. If the input graph G has a feedback vertex set of size at most k, then there must be an execution path that returns a solution, and by Lemma 2.5, the length of this path is at most $4k$. Otherwise, all execution paths return "NO," disregard of their lengths. Therefore, we can terminate every execution path after it has put $3k$ vertices into F by returning "NO" directly. The new search tree would then have depth at most $4k$. Clearly, the processing in each node can be done in $O(n^2)$ time. This gives the running time $O(2^{4k+1} \cdot n^2) = O(16^k \cdot n^2)$. ◀

3 An improved running time

It is long (but *not* well) known that if the maximum degree of a graph is at most three, then a minimum feedback vertex set can be found in polynomial time [13, 21]. This can be extended to the setting that the degree bound holds only for the undecided vertices i.e., vertices in $V(G) \setminus F$.

▶ **Lemma 3.1** ([5]). *Given a graph G and a set F of vertices such that every vertex in $V(G) \setminus F$ has degree at most three, there is a polynomial-time algorithm for finding a minimum set $V_- \subseteq V(G) \setminus F$ such that $G - V_-$ is a forest.*

Therefore, we can change step 4 of algorithm `naive-fvs` to the following:

4. **if** $d(v) \leq 3$ **then**
4.1. call Lemma 3.1 to find a minimum solution X;
4.2. **if** $|X| \leq k$ **then return** X; **else return** "NO";

Now that $d^*(u) \geq 4$ for each vertex $u \in F'$, as a result, in the last inequality in the proof of Lemma 2.5, we can use $(d^*(u) - 2)/d^*(u) \geq 2/4 = 1/2$, which implies $|F'| \leq 2|V_-|$. We can hence terminate every execution path after it has put $2k$ vertices into F by returning "NO" directly, and the algorithm would then run in $O(8^k \cdot n^{O(1)})$ time, as announced in Theorem 1.1.

We conclude this paper by pointing out that the analysis is not tight. The inequalities in the proof of Lemma 2.5 could be tight only when $d^*(v) = 4$ for every vertex $v \in V_- \cup F$, and more importantly, all the degree decrements incurred by putting a vertex to V_- are effective. If such a graph exists,—we may assume without loss of generality that it does not contains any vertex of degree two or less,—then all its vertices have degree four, and all neighbors of a vertex $x \in V_-$ are in F. But in such a graph there should be a different solution, and note that our algorithm explores the subtree rooted at the child node made by step 6 *only if* all the leaves in the other subtree (rooted at the node made by step 5) have returned "NO."

Acknowledgment. The author would like to thank O-joung Kwon and Saket Saurabh for pointing out a mistake in the introduction of a previous version.

References

1 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999. doi:10.1137/S0895480196305124.

2 Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *Journal of Artificial Intelligence Research*, 12:219–234, 2000. doi:10.1613/jair.638.

3 Bozhena Petrovna Bidyuk. *Exploiting Graph Cutsets for Sampling-Based Approximations in Bayesian Networks*. PhD thesis, University of California, Irvine, 2006.

4 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

5 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015. A preliminary version appeared in SWAT 2010. doi:10.1007/s00453-014-9904-6.

6 Fabián A. Chudak, Michel X. Goemans, Dorit S. Hochbaum, and David P. Williamson. A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Operations Research Letters*, 22(4-5):111–118, 1998. doi:10.1016/S0167-6377(98)00021-2.

7 Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1987. doi:10.1016/0004-3702(87)90002-6.

8 Paul Erdős and Lajos Pósa. On the maximal number of disjoint circuits of a graph. *Publicationes Mathematicae Debrecen*, 9:3–12, 1962.

9 Michael R. Fellows and Michael A. Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the ACM*, 35(3):727–739, 1988. doi:10.1145/44483.44491.

10 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation and optimal FPT algorithms. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.

11 Eugene C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982. doi:10.1145/322290.322292.

12 Toshihiro Fujito. A note on approximation of the vertex cover and feedback vertex set problems - unified approach. *Information Processing Letters*, 59(2):59–63, 1996. doi:10.1016/0020-0190(96)00094-4.

13 Merrick L. Furst, Jonathan L. Gross, and Lyle A. McGeoch. Finding a maximum-genus graph imbedding. *Journal of the ACM*, 35(3):523–534, 1988. doi:10.1145/44483.44485.

14 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms*, 12(2):21:1–21:41, 2015. doi:10.1145/2797140.

15 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014. doi:10.1016/j.ipl.2014.05.001.

16 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. Preliminary versions independently presented in STOC 1978. doi:10.1016/0022-0000(80)90060-4.

17 Kurt Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1984.

18 Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann, 1988.

19 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for undirected feedback vertex set. In Prosenjit Bose and Pat Morin, editors, *ISAAC*, volume 2518 of *LNCS*, pages 241–248. Springer, 2002. doi:10.1007/3-540-36136-7_22.

20 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2):32.1–32.8, 2010. A preliminary version appeared in SODA 2009. doi:10.1145/1721837.1721848.

21 Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1-3):355–360, 1988. doi:10.1016/0012-365X(88)90226-9.