

# Experimental Evaluation of Parameterized Algorithms for Feedback Vertex Set

**Krzysztof Kiljan**

Institute of Informatics, University of Warsaw, Poland  
krzysztof.kiljan@student.uw.edu.pl

**Marcin Pilipczuk**

Institute of Informatics, University of Warsaw, Poland  
malcin@mimuw.edu.pl

---

## Abstract

FEEDBACK VERTEX SET is a classic combinatorial optimization problem that asks for a minimum set of vertices in a given graph whose deletion makes the graph acyclic. From the point of view of parameterized algorithms and fixed-parameter tractability, FEEDBACK VERTEX SET is one of the landmark problems: a long line of study resulted in multiple algorithmic approaches and deep understanding of the combinatorics of the problem. Because of its central role in parameterized complexity, the first edition of the Parameterized Algorithms and Computational Experiments Challenge (PACE) in 2016 featured FEEDBACK VERTEX SET as the problem of choice in one of its tracks. The results of PACE 2016 on one hand showed large discrepancy between performance of different classic approaches to the problem, and on the other hand indicated a new approach based on half-integral relaxations of the problem as probably the most efficient approach to the problem. In this paper we provide an exhaustive experimental evaluation of fixed-parameter and branching algorithms for FEEDBACK VERTEX SET.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms, Theory of computation → Graph algorithms analysis

**Keywords and phrases** Empirical Evaluation of Algorithms, Feedback Vertex Set

**Digital Object Identifier** 10.4230/LIPIcs.SEA.2018.12

**Funding** Supported by the “Recent trends in kernelization: theory and experimental evaluation” project, carried out within the Homing programme of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund.

## 1 Introduction

The FEEDBACK VERTEX SET problem asks to delete from a given graph a minimum number of vertices to make it acyclic. It is one of the classic graph optimization problems, appearing already on Karp’s list of 21 NP-hard problems [24]. In this work, we are mostly focusing on *fixed-parameter algorithms* for the problem, that is, exact (and thus exponential-time, as we are dealing with an NP-hard problem) algorithms whose exponential blow-up in the running time bound is confined by a proper parameterization. More formally, a fixed-parameter algorithm on an instance of size  $n$  with parameter value  $k$  runs in time bounded by  $f(k) \cdot n^c$  for some computable (usually exponential) function  $f$  and a constant  $c$  independent of  $k$ .

FEEDBACK VERTEX SET is one of the most-studied problems from the point of view of parameterized algorithms. A long line of research [4, 15, 16, 29, 23, 12, 18, 9, 8, 3, 11] lead to a very good understanding of the combinatorics of the problem, multiple known algorithmic approaches, and a long “race” for the fastest parameterized algorithm under the



© Krzysztof Kiljan and Marcin Pilipczuk;  
licensed under Creative Commons License CC-BY  
17th International Symposium on Experimental Algorithms (SEA 2018).

Editor: Gianlorenzo D’Angelo; Article No. 12; pp. 12:1–12:12



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

parameterization of the solution size (i.e., the parameter  $k$  equals the size of the solution we are looking for). Among many approaches, one can find the following:

1. A classic randomized algorithm of Becker et al. [3] with expected running time  $4^k n^{\mathcal{O}(1)}$ . This algorithm is based on the following observation: after performing a set of simple reductions that reduce the graph to minimum degree at least 3, at least half of the edges of the graph are incident with solution vertices, so a random endpoint of an edge chosen uniformly at random is in the solution with probability at least  $1/4$ .
2. A line of research of branching algorithms based on iterative compression and an intricate measure to bound the size of the branching tree [9, 8, 26], leading to a simple  $3.62^k n^{\mathcal{O}(1)}$ -time deterministic algorithm [26]. These algorithms solve a “disjoint” version of the problem where, given a set  $U \subseteq V(G)$  such that  $G - U$  is a forest, one seeks for a solution *disjoint* with  $U$  of size at most  $k$ . A branching algorithm picks a vertex  $v \in V(G) \setminus U$  and includes it in the solution (deletes and decreases  $k$  by one) or moves to  $U$ . The crucial observation is that here, apart from  $k$ , the number of connected components of  $G[U]$  is a useful potential to measure the progress of a branching algorithm: if the branching pivot  $v$  has many neighbors in  $U$ , then moving it to  $U$  decreases the number of connected components significantly.
3. A polynomial-time algorithm for FEEDBACK VERTEX SET in subcubic graphs [8] (see also [26] for a simpler proof), used heavily in other approaches as a subroutine. The algorithm build on a reduction from FEEDBACK VERTEX SET in cubic graphs to the matroid matching problem in graphic matroids, which is polynomial-time solvable.
4. A Monte Carlo algorithm running in time  $3^k n^{\mathcal{O}(1)}$  via the Cut&Count technique [11]. The Cut&Count technique is a generic framework for turning algorithms with running time  $2^{\mathcal{O}(t \log t)} n^{\mathcal{O}(1)}$  for connectivity problems in graphs of treewidth bounded by  $t$  into Monte-Carlo algorithms with running time bounds  $c^t n^{\mathcal{O}(1)}$  by changing the “connectivity” requirement into a modulo-2 counting requirement. For FEEDBACK VERTEX SET, the base of the exponent has been optimized to  $c = 3$ , and has been proven to be optimal under the Strong Exponential Time Hypothesis.
5. A surprisingly simple branching algorithm by Cao [7] with a running time bound  $8^k n^{\mathcal{O}(1)}$ . Cao [7] shown that the following algorithm has running time bounded by  $8^k n^{\mathcal{O}(1)}$ : after applying simple reduction rules, one either branch on maximum-degree vertex or, if the maximum degree is at most 3, solve the instance in polynomial time.
6. A branching algorithm based on intricate half-integral relaxation due to Imanishi and Iwata [19, 20] with running time bound  $4^k k^{\mathcal{O}(1)} n$ . The work of Iwata, Wahlström, and Yoshida [21] showed a generic framework for branching algorithms for various graph separation problems. In the case of FEEDBACK VERTEX SET, [21] shows that the following local problem has a polynomial-time solvable half-integral relaxation: given a root vertex  $s \in V(G)$ , find a minimum-size set  $X \subseteq V(G) \setminus \{s\}$  such that the connected component of  $G - X$  that contains  $s$  is a tree. Imanishi and Iwata [19, 20] showed a linear-time combinatorial algorithm that solves this half-integral relaxation. The final algorithm follows the framework of [21]: it branches on vertices for which the half-integral relaxation solution was undecided (i.e., gave it the non-integral value of  $1/2$ ).

Another line of research concerns so-called polynomial kernels for the problem [6, 5, 30, 20].

Parameterized Algorithms and Computational Experiments challenge is an annual programming challenge started in 2016 that aims to “investigate the applicability of algorithmic ideas studied and developed in the subfields of multivariate, fine-grained, parameterized, or fixed-parameter tractable algorithms”. With two successful editions so far [13, 14] and the third one currently being conducted, PACE continues to bring together theory and practice in parameterized algorithms community.

The first edition of PACE in 2016 featured FEEDBACK VERTEX SET as the problem of choice in Track B. The winning entry by Imanishi and Iwata, implementing the aforementioned algorithm based on half-integral relaxation, turned out to outperform the second entry by the second author of this work [28], based on the algorithm of Cao [7]. The winning margin has been substantial: out of 130 test cases, the winning entry solved 84, while the second entry solved 66.

These results indicated the branching algorithms based on half-integral relaxation of the problem as potentially most efficient approach to FEEDBACK VERTEX SET in practice. Furthermore, experimental results of Akiba and Iwata [2] showed big potential in a branching algorithm based on the same principle for the VERTEX COVER problem.

In the light of the above, we see the need to rigorously experimentally evaluate different approaches to FEEDBACK VERTEX SET. While the results of PACE 2016 indicate algorithms based on half-integral relaxation as potentially fastest, a lot of differences may come from the use of different preprocessing routines, different choice of lower bounding or pruning techniques, or even simply different data structures handling basic graph operations.<sup>1</sup>

In this work, we offer such a comparison. We implement a number of branching strategies mentioned above. Our implementations use the same data structures for handling graphs, the same implementations of basic graph operations, the same basic reduction rules such as suppressing degree-2 vertices, and the same branching framework. Most of the tested approaches differ only at a small fraction of code: they usually differ by the choice of the branching pivot, and some use one or more approach-specific reduction rules.

In our experiments, we follow up the set up of PACE 2016: we take their 230 instances (100 public and 130 hidden, on which evaluation took place) as our benchmark set, allow each algorithm to run for 30 minutes on each test instance.

The paper is organized as follows. In Section 2 we discuss the studied approaches and some technical details of the implementations. Section 3 discusses the setup of the experiment. Section 4 presents results, while Section 5 concludes the paper.

## 2 Studied algorithms

Most of the known branching algorithm for FEEDBACK VERTEX SET, in particular all algorithms studied in our work, follow the following general framework.

Every instance to be solved by a recursive branching algorithm consists of a multigraph  $G$ , a set  $U \subseteq V(G)$  of *undeletable* vertices and allowed budget  $k$  for solution size. The goal is to find a set  $X \subseteq V(G) \setminus U$  of size at most  $k$  such that  $G - X$  is a forest. Each branching step consists of picking a vertex  $v \in V(G) \setminus U$  and branching into two cases: either  $v$  gets picked to a solution (and the algorithm recurses on the instance  $(G - \{v\}, U, k - 1)$ ) or moved to set  $U$  (and the algorithm recurses on the instance  $(G, U \cup \{v\}, k)$ ).

The intuition of the progress of the algorithm is as follows. In the first branch, the budget  $k$  gets decreased. For the second branch, note that the algorithm can safely terminate for instances with  $G[U]$  not being acyclic. Thus, if the branching pivot  $v$  has  $d$  neighbors in  $U$ ,

<sup>1</sup> A good example here is as follows. In FEEDBACK VERTEX SET, it is natural to keep the graph in the form of adjacency lists, as the considered graphs are usually of constant average degree. However, given an edge  $uv$ , it is not clear whether the vertex  $u$  in its adjacency list should only store the vertex  $v$ , or also a pointer to the position where  $v$  keeps  $u$  in its adjacency list. On one hand, such pointers greatly simplify the operations of deleting a vertex or contracting an edge. On the other hand, they effectively double the size of the graph data structure, increasing the cost of copying the graph in the branching step.

then the number of connected components of  $G[U \cup \{v\}]$  decreases by  $(d - 1)$  as compared to  $G[U]$ .

Between branching step, the algorithm is allowed to perform a number of reduction (preprocessing) steps. In the literature, a number of simple reduction steps are known that are performed by all our algorithms:

1. If  $k$  gets negative or  $G[U]$  is not acyclic, stop.
2. Remove all vertices of degree at most 1.
3. If two vertices are connected by more than 2 parallel edges, reduce their multiplicity to 2.
4. If there exists a vertex  $v$  that has a self-loop, or there exists a single connected component  $D$  of  $G[U]$  more than one edge incident with  $v$  and a vertex of  $D$  (i.e., there exists a cycle  $C$  in  $G$  with  $v$  being the only vertex of  $V(C) \setminus U$ ), then greedily include it in the solution (i.e., delete it and decrease  $k$  by one).
5. Suppress vertices of degree 2. That is, if a vertex  $v$  is of degree 2 with incident edges  $vu$  and  $vw$  is present, delete  $v$  and replace it with an edge  $uw$ .
6. If there exists a vertex  $v$  with two neighbors  $u$  and  $w$ , such that  $vu$  is a single edge and  $vw$  is a double edge, greedily include  $w$  in the solution.

For efficiency, the above reduction rules are implemented in the form of a queue of vertices to reduce: when the number of distinct neighbors of a vertex drop to two or less, or a vertex gets a self-loop, it is enqueued, and preprocessing routines start by clearing up the queue.

Other preprocessing steps used by some of our algorithms are:

**Split into connected components.** If the instance becomes disconnected, solve each connected component independently.

**Solving subcubic instances.** As proven by [8], if every vertex  $v \in V(G) \setminus U$  is of degree at most three, then the corresponding instance is polynomial-time solvable. Kociumaka and Pilipczuk [26] provided a simpler proof of this result via a reduction to the matroid parity problem in graphic matroids, which proceeds as follows. First, apply the known reduction rules that reduce the problem to the case when every  $v \in V(G) \setminus U$  is of degree exactly three. Second, subdivide every edge  $uv \in E(G - U)$  with a new vertex  $x_{uv} \in U$ , so that  $V(G) \setminus U$  is an independent set. Third, for every  $v \in V(G) \setminus U$ , pick two out of the three edges incident to  $v$  as a pair  $P_v$ . Let  $Q = E(G) \setminus \bigcup_{v \in V(G) \setminus U} P_v$  be the remaining edges; note that every  $v \in V(G) \setminus U$  is of degree exactly 1 in the graph  $(V(G), Q)$ . Then, it is easy to observe that the FEEDBACK VERTEX SET problem is equivalent to picking a set  $Y \subseteq V(G) \setminus U$  of maximum cardinality such that  $(V(G), Q \cup \bigcup_{v \in Y} P_v)$  is a forest. This condition is the same as requiring  $Y$  to be a maximum matching in the graphic matroid of the graph  $G/Q$  ( $G$  with edges of  $Q$  contracted) with pairs  $(P_v)_{v \in V(G) \setminus U}$ .

In some of our algorithms, we use the approach of [26] to solve such instances. As the underlying solver to the matroid matching problem, we use the augmenting path algorithm of Gabow and Stallmann [17].

**Solution lower bound.** Consider an instance  $(G, U, k)$  and let  $v_1, v_2, \dots$  be the vertices of  $V(G) \setminus U$  in the nonincreasing order of degrees in  $G$ . If  $(G, U, k)$  admits a solution  $X$  of size  $j$ , then  $G - X$  has at least  $|E(G)| - \sum_{i=1}^j \deg_G(v_i)$  edges. On the other hand, if  $G - X$  is a forest, it has less than  $|V(G)| - j$  edges. Consequently, we can safely stop if for all  $0 \leq j \leq k$  we have that

$$|E(G)| - \sum_{i=1}^j \deg_G(v_i) \geq |V(G)| - j.$$

The above pruning strategy has been used in the entry of Imanishi and Iwata [19].

Unless otherwise noted, all our implementations split instances into connected components. We also compare a number of selected approaches without this preprocessing step to see its impact on performance.

The algorithm of Cao [7] uses all aforementioned simple reduction rules as well as the solver of subcubic instances. On a branching step, it simply chooses the vertex of highest degree. As shown by Cao, such an algorithm has running time bound  $8^k n^{\mathcal{O}(1)}$ . We also test a variant of the algorithm of Cao that first branches on vertices incident with double edges, a variant that does not use the subcubic instance solver, and a variant that prunes the search tree via the aforementioned lower bound.

## 2.1 Approximation and iterative compression

The algorithms of [9, 8, 26] operate in the framework of iterative compression. That is, their central subroutine solves a seemingly simpler problem, where additionally a slightly too large solution  $Y$  is given, and the algorithm first branches on the vertices of  $Y$  (putting each  $y \in Y$  into the solution or into set  $U$ ). Since at the beginning  $G - Y$  is a forest, and every vertex of  $Y$  is either deleted or put into  $U$ , we obtain the property that  $G - U$  is a forest. This greatly helps in the analysis.

In the literature, the set  $Y$  is traditionally taken from the iterative compression step. One picks an order  $V(G) = \{v_1, v_2, \dots, v_n\}$  and solves iteratively FEEDBACK VERTEX SET on graphs  $G_i = G[\{v_1, v_2, \dots, v_i\}]$ . Given a solution  $X_{i-1}$  to  $G_{i-1}$ , one can set  $Y = X_{i-1} \cup \{v_i\}$  for  $G_i$ .

However, such an approach leads to multiple invocation of the same algorithm, and a substantial multiplicative overhead in the running time bound. In our algorithms, we instead find  $Y$  via a simple heuristic: reduce the graph via simple reductions as long as possible and, when impossible, delete the vertex of highest degree. In Section 4 we discuss the performance of this heuristic on our test data.

Our branching framework keeps a queue of **branching hints** and, if nonempty, the algorithm always branches on a vertex from the queue. For algorithms based on iterative compression, the queue is initiated by an approximate solution found by our heuristic. This corresponds to passing the set  $Y$  to the algorithms based on iterative compression, but allows to reduce some of the vertices of the set  $Y$  by reductions after a number of branching steps. If an algorithm does not use iterative compression, the queue is empty through the entire run of the algorithm.

The algorithm of [9] implements the iterative compression framework and branches on a vertex of  $V(G) \setminus U$  that is incident to maximum number of edges leading to  $U$ . As shown in [9], such an algorithm has  $5^k n^{\mathcal{O}(1)}$  time bound guarantee.

The algorithm of [26] is arguably a simplification of the arguments of [8], so we implement only the first one. It modifies the algorithm of [9] in the following way:

- It leaves alone vertices  $v \in V(G) \setminus U$  that are of degree 3 and all their incident edges lead to  $U$  (such vertices are called henceforth *tents*). The crux is that if every vertex  $v \in V(G) \setminus U$  is a tent, then we can apply the polynomial-time algorithm of [8, 26] to the instance. In other words, tents form a “polynomial-time solvable” part of the instance.
- Given a vertex  $v \in V(G) \setminus U$  of degree 3 with exactly one neighbor  $u$  in  $V(G) \setminus U$  (and other 2 neighbors in  $U$ ), it proceeds as follows:
  - subdivide the edge  $uv$  with a new vertex  $w \in U$ ; note that this does not change the set of feasible solutions to the FEEDBACK VERTEX SET problem;
  - marks  $w$  irreducible for the reduction suppressing degree-2 vertices.

Note that this operation turns  $v$  into a tent, while reducing the number of vertices of  $V(G) \setminus U$  that are not tents.

- It applies the solver for subcubic instances if every vertex of  $V(G) \setminus U$  is a tent.

As shown in [26], such an algorithm has  $3.62^k n^{\mathcal{O}(1)}$  time bound guarantee.

Inspired by the methods of choice of branching pivots of the algorithms [9, 8, 26], we also test a variant of Cao’s algorithm where the choice of the branching pivot is as in [9]: vertex with maximum number of neighbors in  $U$  (but, contrary to [9], no iterative compression).

Additionally, we check how much the algorithms can be sped up by adding pruning via the aforementioned lower bound and if the Cao’s algorithm can benefit from the use of iterative compression.

## 2.2 Branching based on half-integral relaxation

Iwata, Wahlström, and Yoshida showed a generic approach to numerous transversal problems via half-integral relaxations [21]. They are all based on the following principle: a half-integral relaxation of a variant of the problem is defined and shown to be polynomial-time solvable. Furthermore, the solution to the half-integral relaxation has some persistency property: it either indicates some greedy choice for the integral problem, or indicates a good branching pivot. In this approach, the time needed to find a solution of the half-integral relaxation is critical.

The algorithms of [21] run in linear time for edge deletion problems, but unfortunately for vertex-deletion problems the subroutine that finds the half-integral relaxation requires solving linear programs. The main contribution of Iwata [20] (implemented in the PACE 2016 entry by Iminishi and Iwata [19]) is a combinatorial linear-time solver for the half-integral relaxation in the special case of FEEDBACK VERTEX SET. We reimplement this solver in our branching framework.

Iwata [20] also observed that the half-integral relaxation can be used to find a polynomial kernel for the problem, improving the previous seminal kernel of Thomassé [30]. Apart from the reduction rules mentioned before, this kernel employs another involved reduction rule, applicable on vertices of degree more than  $2k$ . All our implementations based on a half-integral relaxation implement this preprocessing step as well.

The half-integral relaxation of [21, 20] does not solve FEEDBACK VERTEX SET directly, but rather given an undeletable vertex  $u \in U$ , tries to separate an acyclic connected component (i.e., a tree) containing  $u$  from the rest of the graph. On high level, the branching strategy of the algorithm is as follows. If  $U = \emptyset$ , we branch on any vertex; we choose one of highest degree here for efficiency. Otherwise, we use a vertex  $u \in U$  as a root for the half-integral relaxation. The persistence properties of the relaxation ensure that we can perform a greedy step unless the relaxation put values 0.5 on all neighbors of  $u$ . If this is the case, we branch on a neighbor of  $u$ ; we choose highest-degree neighbor here for efficiency. Note that once a tree component with  $u$  gets separated, simple reduction rules delete it from the graph.

Since the kernelization routine of [20] is computationally expensive, it is not obvious if one should apply it at every step. We experiment with two variants: when we run the kernelization step at every step, or only at steps with  $U = \emptyset$ . Furthermore, we also check how much pruning with the lower bound heuristic or solver of subcubic instances helps.

■ **Table 1** Comparison of the size of the approximate and exact solution found on 127 instances.

difference approximation minus optimum	0	1	2	>2	> 10% · optimum
number of instances	89	30	4	4	5

### 3 Experiment setup

#### 3.1 Hardware and code

The experiments have been performed on a cluster of 16 computers at the Institute of Informatics, University of Warsaw. Each machine was equipped with Intel Xeon E3-1240v6 3.70GHz processor and 16 GB RAM. All machines shared the same NFS drive. Since the size of the inputs and outputs to the programs is small, the network communication was negligible during the process.

The code has been written in C++ and is available at [25] or at project’s webpage [1].

#### 3.2 Test cases

As discussed in the introduction, we repeat the setup of the PACE 2016 challenge [13]. At PACE 2016, the organizers gathered 230 graphs from different sources [27]. A subset of 100 of them has been made public prior to the competition deadline, and the final evaluation has been made on the hidden 130 instances. We run every tested algorithm on each of the 230 instances with 30 minutes timeout.

Our of the test instances, we gathered two subsets to compare actual running times of the algorithm. The first set, *A*, consists of test cases solved by all algorithms, but with substantial running time of some of them. The second set, *B*, is defined similarly, but with regards only to the algorithms that use pruning via the lower bound. More precisely, set *A* consists of the following 14 tests:

```
hidden_001  hidden_007  hidden_012  hidden_056  hidden_065  hidden_083  hidden_099
hidden_106  public_011  public_014  public_037  public_069  public_076  public_086
```

The set *B* consists of the following 7 tests:

```
hidden_022  hidden_041  hidden_068  hidden_088  public_035  public_066  public_067
```

We also gather sizes of approximate solution found by our heuristic and compare it with the optimal size found by the algorithm.

### 4 Results

A full table with running times of each program at each test can be found at project’s website [1].

#### 4.1 Performance of the approximation heuristic

We compared the performance of the approximation heuristic discussed in Section 2.1 (i.e., greedily take the largest-degree vertex after applying the simple reduction rules) with the size of the optimum solution that was known to us on 127 instances. The results are in Table 1. On only 8 instances, the approximate solution was more than one vertex larger than the optimum one. Consequently, the approximate solution can serve well as the basis for iterative compression.

■ **Table 2** Comparison of different algorithms. The first column indicates the base of the algorithm: Cao’s [7], CFLLV for Chen et al. [9], KP for Kociumaka and Pilipczuk [26], and II for Imanishi and Iwata [19, 20]. II/kernel stands for the algorithm of [19, 20] that runs the kernelization step more often, namely at every branching step. Cao/double stands for the algorithm of [7] that first branches on double edges. Cao/undel stands for the algorithm of [7] that chooses branching pivot with regards to maximum degree to undeletable vertices. In the optimizations columns, CC stands for splitting into connected components, deg3 for the use of solver of subcubic instances, LB for the use of pruning with the lower bound, and IC stands for iterative compression.

algorithm	optimizations				solved instances			total time (MM:SS.ms)	
	CC	deg3	LB	IC	all	public	hidden	set <i>A</i>	set <i>B</i>
Cao		+			100	48	52	35:17.21	-
CFLLV				+	91	44	47	35:16.06	-
KP		+		+	101	49	52	36:22.48	-
Cao	+	+			101	48	53	37:31.83	-
CFLLV	+			+	91	44	47	31:23.63	-
KP	+	+		+	101	49	52	32:00.49	-
Cao	+				91	43	48	38:00.78	-
II	+				92	46	46	34:51.17	-
II/kernel	+				90	45	45	75:11.16	-
Cao	+	+	+		123	62	61	0:19.28	10:38.88
Cao/double	+	+	+		122	62	60	3:22.52	21:31.54
Cao/undel	+	+	+		118	58	60	0:35.67	8:39.05
Cao	+	+	+	+	123	62	61	0:19.99	10:37.88
Cao/double	+	+	+	+	122	62	60	3:13.71	14:31.66
Cao/undel	+	+	+	+	118	58	60	0:36.88	8:12.80
CFLLV	+		+	+	117	57	60	0:37.61	8:12.88
KP	+	+	+	+	118	58	60	0:38.95	8:28.46
Cao	+		+		122	61	61	0:22.94	10:28.32
II	+		+		117	58	59	1:36.79	25:31.95
II	+	+	+		118	59	59	1:37.08	25:33.76
II/kernel	+		+		109	55	54	7:24.42	-

## 4.2 Comparison

We have run 21 different algorithms on the whole test data. A CSV file with full results is available at the project’s webpage [1]. Table 2 contains aggregated values: number of solved test instances within the time limit (30 minutes per instance) and the total running time on sets *A* and *B*. Please see the caption of Table 2 for description of the notation used in the table.

The first nine algorithms do not use pruning via the lower bounding technique, and the first three do not use splitting into connected components. They are mostly meant to compare basic approaches.

Without the lower bound pruning, the best approaches are Cao’s [7] and Kociumaka-Pilipczuk [26], and they seem to be rather incomparable. The other algorithm based on iterative compression of Chen et al. [9] is clearly outperformed by the other two, and the same holds for the branching algorithm based on half-integral relaxation [19, 20].

The first three rows differ from rows 4-6 by the usage of splitting into connected components. They show that the effect of this improvement is small, and even hurt a bit Cao’s algorithm [7].



The 7th row treats Cao’s algorithm [7] without the solver of the subcubic instances. It indicates that this solver is essential for the performance of Cao’s algorithm.

The 9th row treats the Imanishi-Iwata algorithm [19, 20] with more often application of the kernelization step, namely at every branching step (not only at the ones with  $U = \emptyset$ ). It shows that the step is too expensive to execute it that often.

Let us now discuss the algorithms with the lower bound pruning step. First, the results show that the pruning step greatly improves performance for all algorithms.

With regards to the variants of Cao’s algorithm [7], the results show that any mutation of the branching pivot rule here is undesirable. Also, adding the iterative compression step does not seem to have any particular impact on the performance. Interestingly, the negative effect of dropping the solver of the subcubic instances mostly disappear if one adds the pruning step.

For the branching algorithm based on the half-integral relaxation [19, 20], the last three rows of Table 2 again confirm the corollary that more often application of the kernelization step is undesirable. There is little difference with addition of the solver of subcubic instances (the main difference comes from the fact that the test set contains one huge cubic graph, `public_84`, which is not amenable to any branching technique we tested).

Finally, in our experiments the best variant of Cao’s algorithm [7] with the pruning slightly outperforms the best variant of the branching algorithm based on half-integral relaxations [19, 20]. However, the difference (5 tests more and roughly  $3\times$  speed-up on sets  $A$  and  $B$ ) is not big enough to decisively conjecture its advantage.

First, it is possible that a  $3\times$  speed-up can be gained by low-level optimizations of the solver of the half-integral relaxations. Arguably, Cao’s algorithm [7] is much simpler and thus easier to optimize. Second, it may be also an artifact of the chosen test data: there are 5 tests in the data set that were solved by the penultimate algorithm in Table 2, but not by the 10th (and 10 tests vice-versa). That is, there are types of instances solved significantly faster by one of the approaches but not by another.

We conclude with a remark about comparison with PACE’16 results [13], where the entry of Imanishi and Iwata [19] solved 84 hidden instances while the entry of the second author [28], based on Cao’s algorithm [7], solved 66. While this data seemingly stands in contradiction with the results in Table 2, one should note that the two entries differ significantly in other internals. Most importantly, the first entry used pruning with the lower bound, while the second one did not. Other difference includes: removal of bridges in the first entry vs splitting into 2-connected components in the second, and the use of bounded treewidth subroutine in the second entry. In other words, our results indicate that the big difference in the performance of the first two entries at PACE 2016 were mainly caused by the difference in preprocessing routines and pruning heuristics (and possibly low-level optimizations) rather than in the underlying base branching algorithm.

## 5 Conclusions

We have conducted a thorough experimental evaluation of various parameterized algorithms for the FEEDBACK VERTEX SET problem, following the setup of Parameterized Algorithms and Computational Experiments Challenge from 2016 [13]. Our results does not confirm greater advantage of the approach based on half-integral relaxations that was suggested by PACE’16 results, but rather suggest that lower bounding techniques and low-level optimizations were decisive at PACE’16.

On the other hand, the approach via half-integral relaxation turned out to be almost on par with the best variant of Cao’s algorithm [7]. This still indicates big potential in this approach, in particular in the light of the recent (theoretical) generalization of this approach to other problems [22]. In particular, it would be interesting to see an experimental evaluation of parameterized algorithms for MULTIWAY CUT with the comparison of the approach of [22] with the more classic ones via so-called important separators (see Chapter 8 of [10]). Other interesting problem to study is ODD CYCLE TRANSVERSAL and its edge-deletion variant EDGE BIPARTIZATION, where again both classic and half-integral relaxation-based approaches are known.

---

## References

- 1 Recent trends in kernelization: theory and experimental evaluation — project website, 2018. URL: <http://kernelization-experiments.mimuw.edu.pl>.
- 2 Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.*, 609:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.
- 3 Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *J. Artif. Intell. Res. (JAIR)*, 12:219–234, 2000. URL: <http://www.cs.washington.edu/research/jair/abstracts/becker00a.html>.
- 4 Hans L. Bodlaender. On disjoint cycles. *Int. J. Found. Comput. Sci.*, 5(1):59–68, 1994.
- 5 Hans L. Bodlaender and Thomas C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory Comput. Syst.*, 46(3):566–597, 2010. doi:10.1007/s00224-009-9234-2.
- 6 Kevin Burrage, Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, Shev Mac, and Frances A. Rosamond. The undirected feedback vertex set problem has a  $poly(k)$  kernel. In Hans L. Bodlaender and Michael A. Langston, editors, *IWPEC*, volume 4169 of *Lecture Notes in Computer Science*, pages 192–202. Springer, 2006. doi:10.1007/11847250\_18.
- 7 Yixin Cao. A naive algorithm for feedback vertex set. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, volume 61 of *OASICS*, pages 1:1–1:9. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/OASICS.SOSA.2018.1.
- 8 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set new measure and new structures. In Haim Kaplan, editor, *SWAT*, volume 6139 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2010. doi:10.1007/978-3-642-13731-0\_10.
- 9 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008. doi:10.1016/j.jcss.2008.05.002.
- 10 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.
- 11 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *FOCS*, pages 150–159. IEEE, 2011. URL: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6108120>, doi:10.1109/FOCS.2011.23.
- 12 Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An  $O(2^{O(k)}n^3)$  FPT algorithm for the undirected feedback vertex set problem. *Theory Comput. Syst.*, 41(3):479–492, 2007. doi:10.1007/s00224-007-1345-z.

- 13 Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. The First Parameterized Algorithms and Computational Experiments Challenge. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPICs*, pages 30:1–30:9. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.30.
- 14 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 30:1–30:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPICs.IPEC.2017.30.
- 15 Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness. In *Complexity Theory: Current Research*, pages 191–225, 1992.
- 16 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.
- 17 Harold N. Gabow and Matthias F. M. Stallmann. An augmenting path algorithm for linear matroid parity. *Combinatorica*, 6(2):123–150, 1986. doi:10.1007/BF02579169.
- 18 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006.
- 19 Kensuke Imanishi and Yoichi Iwata. Feedback Vertex Set solver, entry to PACE 2016, 2016. <http://github.com/wata-orz/fvs>.
- 20 Yoichi Iwata. Linear-time kernelization for feedback vertex set. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 68:1–68:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.68.
- 21 Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, LP-branching, and FPT algorithms. *SIAM J. Comput.*, 45(4):1377–1411, 2016. doi:10.1137/140962838.
- 22 Yoichi Iwata, Yutaro Yamaguchi, and Yuichi Yoshida. Linear-time FPT algorithms via half-integral non-returning a-path packing. *CoRR*, abs/1704.02700, 2017. arXiv:1704.02700.
- 23 Iyad A. Kanj, Michael J. Pelsmayer, and Marcus Schaefer. Parameterized algorithms for feedback vertex set. In Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors, *IWPEC*, volume 3162 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2004. doi:10.1007/978-3-540-28639-4\_21.
- 24 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. URL: <http://www.cs.berkeley.edu/~rsimluca/cs172/karp.pdf>.
- 25 Krzysztof Kiljan and Marcin Pilipczuk. Experimental evaluation of parameterized algorithms for Feedback Vertex Set: code repository, 2018. [https://bitbucket.org/marcin\\_pilipczuk/fvs-experiments](https://bitbucket.org/marcin_pilipczuk/fvs-experiments).
- 26 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014. doi:10.1016/j.ipl.2014.05.001.
- 27 Christian Komusiewicz. PACE 2016: tests for Feedback Vertex Set, 2016. <https://github.com/ckomus/PACE-fvs>.
- 28 Marcin Pilipczuk. Feedback Vertex Set solver, entry to PACE 2016, 2016. [http://bitbucket.com/marcin\\_pilipczuk/fvs-pace-challenge](http://bitbucket.com/marcin_pilipczuk/fvs-pace-challenge).

## 12:12 Experimental Evaluation of Parameterized Algorithms for FVS

- 29 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms*, 2(3):403–415, 2006.
- 30 Stéphan Thomassé. A  $4k^2$  kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.