Originally published in *Journal of Discrete Algorithms*, 8(1):76–86. Elsevier B. V., 2010.

Fixed-Parameter Tractability Results for Feedback Set Problems in Tournaments[☆]

Michael Dom[a], Jiong Guo[c,1], Falk Hüffner[b,1], Rolf Niedermeier[a], Anke Truss[a]

[a] *Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany.*
{michael.dom, rolf.niedermeier, anke.truss}@uni-jena.de
[b] *School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.*
hueffner@tau.ac.il
[c] *Universität des Saarlandes, Campus E 1.4, D-66123 Saarbrücken.*
jguo@mmci.uni-saarland.de

Abstract

Complementing recent progress on classical complexity and polynomial-time approximability of feedback set problems in (bipartite) tournaments, we extend and improve fixed-parameter tractability results for these problems. We show that Feedback Vertex Set in tournaments (FVST) is amenable to the novel iterative compression technique, and we provide a depth-bounded search tree for Feedback Arc Set in bipartite tournaments based on a new forbidden subgraph characterization. Moreover, we apply the iterative compression technique to d-Hitting Set, which generalizes Feedback Vertex Set in tournaments, and obtain improved upper bounds for the time needed to solve 4-Hitting Set and 5-Hitting Set. Using our parameterized algorithm for Feedback Vertex Set in tournaments, we also give an exact (not parameterized) algorithm for it running in $O(1.709^n)$ time, where n is the number of input graph vertices, answering a question of Woeginger [Discrete Appl. Math. 156(3):397–405, 2008].

Key words: Feedback arc set, Feedback vertex set, Tournament, Bipartite tournament, Fixed-parameter tractability, Iterative compression

1. Introduction

Feedback set problems deal with destroying cycles in graphs using a minimum number of vertex deletions or edge deletions [23]. When considering directed graphs, there are basically two problems: Feedback Arc Set (FAS) asks for a minimum number of arcs to be deleted in order to obtain a cycle-free

[☆] An extended abstract of this paper appeared in the Proceedings of the 6th Conference on Algorithms and Complexity (CIAC 2006), May 29–31, 2006, Rome, Italy, volume 3998 in Lecture Notes in Computer Science, pages 320–331, Springer, 2006. Besides improved running times and an exact (non-parameterized) algorithm, this long version now presents, based on the PhD thesis of the third author [33], new material on Hitting Set problems that has not been part of the conference version. Moreover, a section concerning problem kernels has been omitted, because its results have been improved by a publication of Abu-Khzam [1]. The main work was done while all authors were with the Friedrich-Schiller-Universität Jena.

[1] Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

Originally published in *Journal of Discrete Algorithms*, 8(1):76–86. Elsevier B. V., 2010.

Table 1: Complexity results for feedback set problems in tournaments. Herein, n denotes the number of vertices and k denotes the size of the desired feedback solution set. The entry "poly" in the running times of the approximation algorithms means that the authors did not specify the running times of their polynomial-time approximations.

		Approximation			Fixed-parameter tractability			
	Complexity	factor		running time	running time		kernel	
FVST	NP-c [47]	2.5	[8]	$O(n^3)$	$O(2^k \cdot n^2(\log\log n + k))$	[§4]	$O(k^2)$	[1]
FVSBT	NP-c [9]	2	[50]	poly	$O(3^k \cdot n^2 + n^3)$	[44]	$O(k^3)$	[1]
FAST	NP-c [5, 11, 16]	PTAS	[36]	poly	$2^{O(\sqrt{k}+\log^2 k)} + n^{O(1)}$	[6]	$O(k^2)$	
FASBT	NP-c [29]	4	[50]	poly	$O(3.373^k \cdot n^6)$	[§5]	?	

directed graph, whereas Feedback Vertex Set (FVS) asks for a minimum number of vertex deletions. Although feedback set problems usually are NP-hard for undirected as well as for directed graphs, the algorithmic treatment by means of approximation, exact, or parameterized algorithms seems to be significantly easier in the undirected case where more and better results are known. For directed graphs, most results so far concern the class of *tournaments*. A tournament is a directed graph where between any two distinct vertices there is exactly one arc. Motivated by applications such as voting systems and rank aggregation [4, 12, 36], there has recently been much interest in feedback set problems in tournaments. For instance, the NP-hardness of Feedback Arc Set in tournaments has recently been addressed by at least four independent groups of researchers [4, 5, 11, 16]. Here, we contribute new results concerning the algorithmic tractability of Feedback Arc Set and Feedback Vertex Set in tournaments and bipartite tournaments.

Table 1 surveys known and new complexity results for feedback set problems in (bipartite) tournaments.

Approximation algorithms. Concerning polynomial-time approximability, the following results are known. For FVS in tournaments (FVST), the trivial factor 3 has been improved to 2.5 [8] whereas for FVS in bipartite tournaments (FVSBT) the trivial factor 4 has been improved to 3.5 [9], 3 [45], and lastly 2 [50]. Note that the approximation-preserving reduction from Vertex Cover to FVST [47] together with the inapproximability result for Vertex Cover [18] shows that it is NP-hard to approximate FVST better than by a factor of 1.360 (see also [37] for the conjectured hardness of a factor-$(2 - \epsilon)$ approximation). In contrast, for FAS in tournaments (FAST), a PTAS is known[2] [36], that is, for any constant ϵ, a polynomial-time factor-$(1 + \epsilon)$ approximation algorithm can be given, albeit with exponential dependency on $1/\epsilon$. Finally, a factor-4 approximation for FASBT has been shown by van Zuylen [50] using techniques similar to those by Ailon et al. [4], correcting a previous approach by Gupta [31].

Parameterized algorithmics. As an alternative to approximations, it is reasonable to study feedback set problems from a parameterized point of view [19, 24, 40] (see also Gutin and Yeo [32] for a survey on parameterized problems on directed graphs). For instance, in undirected graphs, using iterative compression [30], it has been shown that a feedback vertex set of size at most k can be

[2]The corresponding algorithm is impractical and only of theoretical interest.

found in $37.7^k \cdot n^{O(1)}$ time [28] and $10.57^k \cdot n^{O(1)}$ time [17], where n is the number of graph vertices. The running time has now been lowered to $5^k \cdot kn^2$ [14]. The question whether Feedback Vertex Set on general directed graphs is fixed-parameter tractable had been famously open for a long time and has only recently been resolved positively, also using iterative compression [15]; however, the given algorithm running in $4^k k! \cdot n^{O(1)}$ time incurs a much worse combinatorial explosion with respect to the parameter k than those algorithms specialized to tournaments. Restricting the consideration to the class of tournaments, Raman and Saurabh [42] have given the first positive result by giving fixed-parameter algorithms for weighted FVST and weighted FAST running in $2.415^k \cdot n^{O(1)}$ time. For the unweighted case of FVST, the previously fastest algorithm is obtained by a reduction to 3-Hitting Set and runs in $2.076^k \cdot n^{O(1)}$ time [48]. In a recent manuscript, Alon, Lokshtanov, and Saurabh [6] gave for FAST one of the rare examples of a subexponential time algorithm with a running time of $2^{O(\sqrt{k}+\log^2 k)} + n^{O(1)}$. An algorithm for FVSBT with a running time of $O(3.116^k \cdot n^4)$ can also be derived using a 4-Hitting Set algorithm by Fernau [22]; with the 4-Hitting Set algorithm from Theorem 3.1, we get a running time of $O(3.076^k + n^4)$. Recently, an algorithm for FVSBT running in $O(3^k \cdot n^2 + n^3)$ time was given by Sasatte [44].

An important tool from the FPT toolchest are kernelizations [27]. A kernelization replaces, in polynomial time, an instance by a decision equivalent instance (the *kernel*) whose size can be bounded by a function of the parameter k, that is, it will not depend on the original problem size n anymore. Using kernels for d-Hitting Set [1], one can derive a kernel of $O(k^2)$ vertices and $O(k^3)$ edges for FVST, and a kernel of $O(k^3)$ vertices and $O(k^4)$ edges for FVSBT. A kernel for FAST is also easy to achieve: If an arc occurs in more than k triangles, it needs to be deleted. After doing this exhaustively, at most $O(k^2)$ vertices can be left, or the instance is unsolvable.

Feedback Vertex Set in tournaments can also be seen as the problem of making an antisymmetric relation transitive by omitting a minimum number of elements. For the related problem of making a symmetric relation transitive by omitting a minimum number of elements, known as Cluster Vertex Deletion, fixed-parameter tractability results have recently been given [34].

We mention in passing that Raman et al. [43] provided exact (not parameterized with respect to k) algorithms solving FAST in $O(1.555^m)$ time (where m is the number of arcs).

Our contributions. We start, in Section 3, with considering the problem d-Hitting Set, where one has to delete k vertices from a hypergraph that has m size-d hyperedges, such that all hyperedges are destroyed. d-Hitting Set is a generalization of FVST and FVSBT, because FVST can be reduced to 3-Hitting Set by mapping the vertices of the input tournament to vertices of the hypergraph and mapping triangles to hyperedges. Similarly, FVSBT can be reduced to 4-Hitting Set. By using iterative compression [30], we show that solving an instance of d-Hitting Set reduces to solving several instances of $(d-1)$-Hitting Set. This approach results in parameterized algorithms for 4-Hitting Set and 5-Hitting Set that run in $O(3.076^k + m)$ time and $O(4.076^k + m)$ time, respectively, and, thus, are faster than the previously fastest known algorithms for these problems.[3]

In Section 4, we improve the time bound of exactly solving (parameterized) unweighted FVST to $O(2^k \cdot n^2(\log\log n + k))$. This also demonstrates the applicability of the elegant iterative compression method in contrast to the more standard case-distinction based search tree approaches employed by Raman and Saurabh [42] and Wahlström [48]. Further, this allows us to give an exact (not parameterized) algorithm for FVST running in $O(1.709^n)$ time, answering a question of Woeginger [49].

For FASBT, iterative compression could so far not be applied. Therefore, in Section 5 we provide a $3.373^k \cdot n^{O(1)}$-time algorithm for FASBT which is based on a novel characterization by forbidden subgraphs.

2. Preliminaries

In this paper we deal with fixed-parameter algorithms that emerge from the field of parameterized complexity analysis [19, 24, 40]. An instance of a parameterized problem consists of a problem instance I and a parameter k. A parameterized problem is *fixed-parameter tractable* if it can be solved in $f(k) \cdot |I|^{O(1)}$ time, where f is a computable function solely depending on the parameter k, not on the input size $|I|$.

A *directed graph* or *digraph* D consists of a vertex set V and an arc set E with $n := |V|$ and $m := |E|$ [7]. Each arc is an ordered pair of vertices. We consider only digraphs without loops, that is, $(v, v) \notin E$ for all $v \in V$. We call a digraph $D' = (V', E')$ an *induced subgraph* of $D = (V, E)$ if $V' \subseteq V$ and $E' = \{(u, v) \mid u, v \in V' \text{ and } (u, v) \in E\}$. The subgraph of D induced by a vertex subset V' is denoted by $D[V']$. With *reversing* an arc (u, v) we mean that we delete the arc (u, v) from E and insert (v, u) into E. A *tournament* $T = (V, E)$ is a digraph where there is exactly one arc between each pair of vertices. A digraph is a *bipartite tournament* if its vertex set is the union of two disjoint sets V_1 and V_2 such that each arc consists of one vertex from each of V_1 and V_2 and between each vertex from V_1 and each vertex from V_2 there is exactly one arc. A *cycle* is a sequence of distinct vertices $v_1, \ldots, v_s$ with $(v_i, v_{i+1}) \in E$ for all $1 \leq i < s$ and $(v_s, v_1) \in E$. A *triangle* is a cycle of length 3, a *chord* is an arc that connects two vertices of a cycle that are not consecutive in the cycle. A *topological sort* of a digraph $D = (V, E)$ is a sequence $v_1, v_2, \ldots, v_n$ of the vertices in V in which each vertex appears exactly once and $i < j$ for each arc $(v_i, v_j) \in E$. Clearly, a digraph has a topological sort iff it is acyclic, that is, it does not contain a cycle.

A *hypergraph* consists of a vertex set V and a hyperedge set E, where each hyperedge is a nonempty subset of V.

The problem Feedback Vertex Set (Feedback Arc Set) in tournaments, FVST (FAST) for short, is defined as follows:

> **Input**: A tournament T and a nonnegative integer k.
> **Task**: Find a set F of at most k vertices (arcs) whose deletion results in an acyclic digraph.

[3]Similar ideas have been used independently by Fomin et al. [25] for obtaining non-parameterized exact algorithms for d-Hitting Set.

Originally published in *Journal of Discrete Algorithms*, 8(1):76–86. Elsevier B. V., 2010.

The set F is called a *feedback vertex set* (*feedback arc set*). When the input digraph is restricted to bipartite tournaments instead of tournaments, we call the problem Feedback Vertex Set (Feedback Arc Set) in bipartite tournaments, FVSBT (FASBT) for short.

Since any cycle of length at least four in a tournament has at least one chord and since any chord of a cycle together with some arcs of the cycle forms a shorter cycle, we get the following lemma [7].

Lemma 2.1. *A tournament is acyclic iff it contains no triangles.*

Due to the following lemma from folklore, we can reverse arcs instead of deleting them when dealing with FAST and FASBT. This is useful because it allows us to apply feedback arc sets without leaving the class of (bipartite) tournaments.

Lemma 2.2. *Let F be a minimal feedback arc set of a digraph D. Then the graph D' formed from D by reversing the arcs in F is acyclic.*

Proof. Let D'' be the graph resulting by deleting the arcs of F from D, and let $v_1, \ldots, v_n$ be the topological sort of the vertices of D''. Due to the minimality of F, every edge $(v_i, v_j) \in F$ has $i > j$—otherwise, inserting (v_i, v_j) into D'' would not create a cycle, contradicting the minimality of F. Inserting all edges (v_j, v_i) with $(v_i, v_j) \in F$ into D'', therefore, results in a cycle-free graph D'. □

3. Iterative Compression for Hitting Set

We now show how to solve Hitting Set, which generalizes FVST and FVSBT, by using iterative compression. This also serves to introduce the technique (see also [30] for a recent survey on iterative compression) and further produces the currently fastest algorithms for 4-Hitting Set and 5-Hitting Set. As introductory example, we use 3-Hitting Set. To emphasize the similarity to the graph problems, we formulate it as a hypergraph modification problem.

> 3-Hitting Set
> **Instance:** A hypergraph $G = (V, E)$ with $|e| = 3$ for all $e \in E$ and an integer $k \geq 0$.
> **Question:** Is there a *hitting set* $X \subseteq V$ with $|X| \leq k$, that is, a set of vertices whose deletion removes all hyperedges?

Here, deleting a vertex implies also removing all hyperedges that contain this vertex. 3-Hitting Set is NP-complete [26]. There is a simple 3-approximation for the minimization version of the problem (repeatedly take all three vertices of a hyperedge); it has been conjectured that this approximation factor cannot be improved [37]. Note that the variant 2-Hitting Set is equivalent to the NP-complete Vertex Cover problem. 3-Hitting Set can be solved in $O(3^k m)$ time by a simple search tree algorithm: choose any hyperedge $\{v_1, v_2, v_3\} \in E$ and branch into the three cases $v_1 \in X$, $v_2 \in X$, and v_3 in X. By case distinction and careful analysis, this has been improved in a series of results to $O(2.270^k + m)$ [41], then $O(2.179^k + m)$ [21, 22], and finally $O(2.076^k + m)$ [48]. A kernel of

```
IterativeCompression($G = (V, E)$)
1   $V' \leftarrow \emptyset$
2   $X \leftarrow \emptyset$
3   for each $v \in V$:
4       $V' \leftarrow V' \cup \{v\}$
5       $X \leftarrow X \cup \{v\}$
6       $X \leftarrow$ Compress($G[V'], X$)
7   return $X$
```

Figure 1: Pseudo-code for iterative compression, using the compression routine Compress. The function call Compress($G[V'], X$) returns a hitting set for $G[V']$ that is smaller than the hitting set X, if possible.

size $O(k^3)$ is known [41], which has recently been improved to $O(k^2)$ vertices and $O(k^3)$ edges [1].

The central idea of iterative compression is to use a *compression routine*, that is, an algorithm that, given a problem instance and a solution, either calculates a smaller solution or proves that the given solution is of minimum size. The most obvious way to employ a compression routine is to start with an approximate solution and then use the compression routine until no further compression is possible. However, since the running time of the compression routine depends exponentially on the size of the solution to compress, it is faster to build up the graph vertex-by-vertex while always keeping a minimal solution. This is illustrated in the pseudo-code in Figure 1.

We start with $V' = \emptyset$ and $X = \emptyset$; clearly, X is a minimum hitting set for $G[V']$. In lines 4 and 5, we add one vertex $v \notin V'$ from V to both V' and X. Then X is still a hitting set for $G[V']$, although possibly not a minimum one. We can, however, obtain a minimum one by applying our compression routine. Here, the compression routine Compress takes a hypergraph G and a hitting set X for G, and returns a smaller hitting set for G if there is one; otherwise, it returns X unchanged. Therefore, it is a loop invariant that X is a minimum-size hitting set for $G[V']$. Since eventually $V' = V$, we obtain an optimal solution for G once the algorithm returns X.

Note that we defined a compression routine as a function that returns a *smaller* solution, but not necessarily a minimum one. This suffices here, because the hitting set $X \cup \{v\}$ to be compressed can be larger by at most one than an optimal hitting set X' for $G[V' \cup \{v\}]$; this is because X' is also a hitting set for $G[V']$, and cannot be smaller than the minimum hitting set X.

It remains to describe the compression routine. The basic idea, which is shared with most other known iterative compression algorithms [30], is to reduce the compression problem to a *disjoint* compression problem:

Definition 3.1. *A* disjoint compression routine *is an algorithm that, given a problem instance and a solution S, either calculates a smaller solution that is disjoint from S or proves that this is not possible.*

The reason for working with a disjoint compression routine is that it gives us extra structure to work with: Not only do we know that $G[V \setminus S]$ is hyperedge-

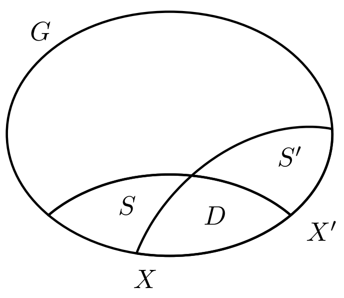

Figure 2: Partition of X

```
COMPRESS($G, X$)
1  for each $S \subseteq X$:
2      $D \leftarrow X \setminus S$
3      if $G[S]$ is a hyperedge-free graph:
4          $G' \leftarrow G[V \setminus D]$
5          $S' \leftarrow$ COMPRESSDISJOINT($G', S$)
6          if $|S'| < |S|$:
7              return $(X \setminus S) \cup S'$
8  return $X$
```

Figure 3: Pseudo-code for COMPRESS. The function call COMPRESSDISJOINT(G', S) returns a hitting set for G' that is smaller than the hitting set S, if possible, and disjoint from S.

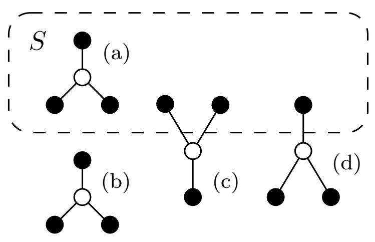

Figure 4: Hyperedges in disjoint compression for 3-HITTING SET. Black circles are vertices, white circles connected to three vertices are hyperedges.

free, but also that $G[S]$ is hyperedge-free, because otherwise we can immediately claim that no compression is possible, since we are not allowed to delete vertices from S.

For the transformation from compression to disjoint compression, consider a smaller solution X' as a modification of the known solution X. It will retain some vertices D from X and replace the other vertices S with fewer vertices S' (Figure 2). The idea (see Figure 3) is to try by brute force all $2^{|X|}$ possibilities to partition X into S and D (line 1). If $G[S]$ still has hyperedges, then there is no solution disjoint from S, and we can skip this partition (line 3). Since we decided to keep all vertices of X in the solution except for those in S, we can immediately get rid of the other vertices, that is, the vertices in D (line 4). We have thus gained the disjointness assumption at a cost of a factor of $2^{|X|} = O(2^k)$ in the running time. It now remains to find in $G[V \setminus D]$ an optimal hitting set that is disjoint from S, which is done by the function COMPRESSDISJOINT.

To implement COMPRESSDISJOINT, we examine possible configurations of hyperedges (Figure 4). Configuration (a) is not possible because of the check in line 3. Configuration (b) is not possible either, because S is a hitting set for G'. If we encounter configuration (c), we can immediately delete the single vertex that is not in S, since there is no other way to get rid of such a hyperedge. So the only remaining possibility is (d): each remaining hyperedge has exactly one vertex in S and two vertices in $V \setminus S$. Since we are not allowed to delete any vertex in S, we might as well omit them. This leaves us with a number of 2-element edges, the task still being to delete vertices to get rid of all edges. This

Originally published in *Journal of Discrete Algorithms*, 8(1):76–86. Elsevier B. V., 2010.

is exactly the Vertex Cover problem. For Vertex Cover, many fast parameterized algorithms exist, which we can use to solve the remaining instance. We arrive at the following theorem.

Proposition 3.1. 3-Hitting Set *can be solved in* $O(2.274^k \cdot kn^2)$ *time by using iterative compression.*

Proof. The data reduction in CompressDisjoint (removal of edges with two vertices in S) can be executed in $O(kn)$ time, if we do it incrementally and enumerate subsets of X in a way such that at each step only membership of one vertex changes; this can be done using a Gray code [38, Section 7.2.1.1]. More precisely, to remove all edges with two vertices in S, we just modify the output of the data reduction performed in the previous call of CompressDisjoint: Since, compared to the previous call of CompressDisjoint, only one vertex v has moved from $V \setminus S$ to S or from S to $V \setminus S$, we check for each of the at most k neighbors $u \in S$ of v and each of the at most n neighbors $w \in V \setminus S$ of v whether the hyperedge $\{u, v, w\}$ can be eliminated or whether it was eliminated in the previous call of CompressDisjoint and has to be re-inserted now.

Then, the remaining task is to solve a Vertex Cover instance with at most n vertices and m edges. Vertex Cover with a cover size of at most k' can be solved in $O(1.274^{k'} + k'n)$ time [13]. We thus can execute Compress in $O(\sum_{S \subseteq X}(1.274^{|S|} + |S|n))$ time. Using $\sum_{i=0}^{k}\binom{k}{i}c^i = (c+1)^k$ for any c and the fact that $|S|$ is bounded by $k+1$, this gives an $O(\sum_{i=0}^{k+1}\binom{k+1}{i}(1.274^i + in)) = O(2.274^k \cdot kn)$ time bound. The compression routine is called at most n times, giving an overall running time of $O(2.274^k \cdot kn^2)$ as claimed. □

Using a kernelization [1, 41] and the fact that the rounding of the exponential base allows us to omit polynomial factors of k, we can even claim a running time of $O(2.274^k + m)$ (although this borders on abuse of the Big O notation).

The running time of this iterative compression algorithm is already competitive with that of the algorithm of Niedermeier and Rossmanith [41], which runs in $O(2.270^k + m)$ time; however, it is not as fast as the best known 3-Hitting Set algorithm by Wahlström [48] running in $O(2.076^k + m)$ time. Still, it might be a useful approach to solving 3-Hitting Set in practice, in particular since except for the Vertex Cover subroutine, it is very simple, and high-performance Vertex Cover implementations have been presented (see, for instance, Abu-Khzam et al. [3] and Felner et al. [20]).

Furthermore, we can in the same way use iterative compression to solve 4-Hitting Set using a 3-Hitting Set algorithm, or more generally d-Hitting Set using iterative compression and a $(d-1)$-Hitting Set algorithm. If we use the 3-Hitting Set algorithm by Wahlström [48], we obtain the following theorem.

Theorem 3.1. 4-Hitting Set *can be solved in* $O(3.076^k + m)$ *time, and* 5-Hitting Set *can be solved in* $O(4.076^k + m)$ *time.*

These algorithms are slightly faster than the previously fastest known by Fernau [22] running in $O(3.116^k + m)$ and $O(4.079^k + m)$ time, respectively. For d-Hitting Set with $d > 5$, this approach does not yield new records anymore; further, we have an increasing polynomial overhead with growing d, since the running time for d-Hitting Set is more precisely $O((d-0.924)^k n^{d-3} + (d-2)^k n^{d-3} m$.

4. Iterative Compression for Feedback Vertex Set in Tournaments

We use the same overall scheme as for 3-Hitting Set. To make the task of looking for a smaller feedback vertex set for a tournament T easier, we would like to restrict our search to feedback vertex sets that are disjoint from a given one. This is the same approach as used in Section 3 and most other iterative compression algorithms. We can achieve this in the same way as for 3-Hitting Set (see Figure 3), that is, by a brute-force enumeration of all $O(2^{|X|})$ possibilities to partition the given feedback vertex set X into two vertex sets S and $X \setminus S$. For each partition, we then look only for solutions that contain all of $X \setminus S$ (they can immediately be deleted from the tournament), but none of S. Further, we can omit all partitions where $T[S]$ is not cycle-free, since we determined none of the vertices in S would be deleted. Therefore, all that remains is to deal with the following problem.

FVST Disjoint Compression
Instance: A tournament $T = (V, A)$ and a subset $S \subseteq V$ such that $T[S]$ and $T[V \setminus S]$ are acyclic.
Task: Find a set $S' \subseteq V \setminus S$ with $|S'| < |S|$ such that $T[V \setminus S']$ is acyclic.

Up to this point, the algorithm is analogous to the iterative compression algorithms for general directed Feedback Vertex Set [15] and undirected Feedback Vertex Set [17, 28]. The core part of the compression routine, however, is completely different; in particular, we will be able to solve the remaining task of finding a smaller feedback vertex set that is disjoint from the given one S in polynomial time, whereas Chen et al. [15] still require exponential (in k) time for this task in the case of Feedback Vertex Set on general directed graphs, as well as Dehne et al. [17] and Guo et al. [28] when solving Feedback Vertex Set on undirected graphs.

Consider a FVST Disjoint Compression instance (T, S). As mentioned, both $T[S]$ and $T[V \setminus S]$ are acyclic and thus have a topological sort. Then, the topological sort of a maximum acyclic subtournament of T containing all of S can be thought of as resulting from inserting a subset of $V \setminus S$ into the topological sort of S. On the one hand, the order of the inserted subset must not violate the topological sort of $T[V \setminus S]$. On the other hand, we can achieve by a data reduction rule that for every $v \in V \setminus S$, the subtournament $T[S \cup \{v\}]$ is acyclic and therefore v has a "natural" position within the topological sort of S. We then obtain the maximum acyclic subtournament as the longest common subsequence of the topological sort of $T[V \setminus S]$ and $V \setminus S$ sorted by natural position within S. The details follow.

Our approach is based on the subroutine displayed in Figure 5. First we apply data reduction to the instance: whenever there is a triangle with two vertices in S, we can only get rid of this triangle by deleting the third vertex (lines 3–5). After applying this reduction rule exhaustively, for any $v \in V \setminus S$, the subtournament $T[S \cup \{v\}]$ clearly does not contain triangles anymore and therefore is acyclic by Lemma 2.1. This means that we can insert v at some point in the topological sort $s_1, \ldots, s_{|S|}$ of S without introducing back arcs (that is, arcs pointing from a higher indexed vertex to a lower indexed vertex in the sort). Since T is a tournament, there is thus some integer $p[v]$ such that for $i < p[v]$,

Input: Tournament $T = (V, A)$ and a feedback vertex set S for T.
Output: A minimum feedback vertex set F for T with $F \cap S = \emptyset$.

```
1  $s_1, \dots, s_{|S|} \leftarrow$ topological sort of $T[S]$
2  $R \leftarrow \emptyset$
3  while there is a triangle $u, v, w$ with $u, v \in S$ and $w \in V \setminus S$:
4      $R \leftarrow R \cup \{w\}$
5      $T \leftarrow T[V \setminus \{w\}]$
6  for each $v \in V \setminus S$:
7      $p[v] \leftarrow \min(\{i \mid (v, s_i) \in A\} \cup \{|S| + 1\})$
8  $L \leftarrow$ topological sort of $T[V \setminus S]$
9  $P \leftarrow V \setminus S$ sorted by $p$, with position in $L$ as tie-breaker
10 $Y \leftarrow$ vertices in a longest common subsequence of $L$ and $P$
11 return $R \cup ((V \setminus S) \setminus Y)$
```

Figure 5: Algorithm for FVST Disjoint Compression

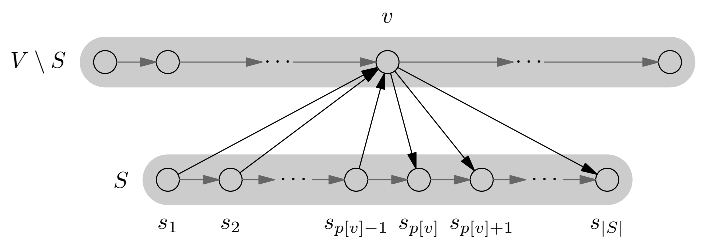

Figure 6: Illustration of equivalence (1). For clarity, only some of the arcs within the acyclic subtournaments $T[S]$ and $T[V \setminus S]$ are shown.

there is an arc from s_i to v, and for $i \geq p[v]$, there is an arc from v to s_i (Figure 6):

$$(v, s_i) \in A \iff i \geq p[v]. \tag{1}$$

We calculate p in lines 6–7: when we encounter the first s_i in the topological sort of S where $(v, s_i) \in A$, we can insert v before s_i; if there is no such s_i, we set $p[v]$ to $|S| + 1$, and (1) still holds.

We now construct a sequence P from p (line 9), where vertices from $V \setminus S$ that are positioned by p between the same two vertices of S are ordered according to their relative position in the topological sort of $T[V \setminus S]$. Clearly, any acyclic subtournament of T containing all of S must have a topological sort where the vertices from $V \setminus S$ occur in the same order as in P. The same holds for the topological sort L of $T[V \setminus S]$, which is calculated in line 8. This leads to the following lemma.

Lemma 4.1. *After line 9 of the algorithm in Figure 5, T is acyclic iff the sequences L and P are equal.*

Proof. "$\Rightarrow$": If L and P are not equal, then there are $v, w \in V \setminus S$ with $(v, w) \in A$ but $p[v] > p[w]$. Then, by (1), we have $(w, s_{p[w]}) \in A$ and $(v, s_{p[w]}) \notin A$. Since T is a tournament, $(v, s_{p[w]}) \notin A$ implies $(s_{p[w]}, v) \in A$, and, therefore, T contains the cycle $v, w, s_{p[w]}$.

Originally published in *Journal of Discrete Algorithms*, 8(1):76–86. Elsevier B. V., 2010.

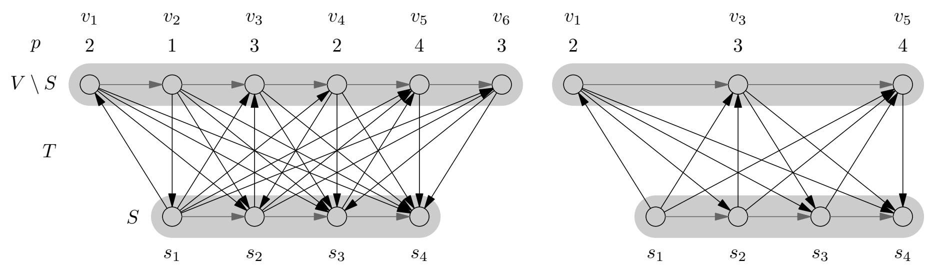

Figure 7: Example for the subroutine in Figure 5. For clarity, only some of the arcs within the acyclic subtournaments $T[S]$ and $T[V \setminus S]$ are shown. Left: Tournament T after data reduction with $L = v_1, v_2, v_3, v_4, v_5, v_6$ and $P = v_2, v_1, v_4, v_3, v_6, v_5$. A longest common subsequence is v_1, v_3, v_5, yielding the acyclic graph shown on the right.

"$\Leftarrow$": By Lemma 2.1, it suffices to look for triangles to decide whether T is acyclic. Since $T[S]$ and $T[V \setminus S]$ are acyclic and we destroyed all triangles with two vertices in S, there can only be triangles with exactly two vertices in $V \setminus S$. If L and P are equal, then for all $v, w \in V \setminus S$ with $(v, w) \in A$ we have $p[v] \leq p[w]$. Then by (1), there cannot be any s_i with $(w, s_i) \in A$ and $(s_i, v) \in A$, and there can be no triangle in T. $\square$

With the same justification, the statement of Lemma 4.1 holds for induced subgraphs of T and the corresponding sequences L and P:

Corollary 4.1. *After line 9 of the algorithm in Figure 5, for any subset Y of T's vertices it holds that $T[Y]$ is acyclic iff the vertices of Y appear in a common subsequence of L and P.*

Proof. The corollary follows immediately from the proof of Lemma 4.1, because deleting a vertex $v \in V \setminus S$ from T affects L and P only insofar as v disappears from L and P. $\square$

According to Corollary 4.1, the cheapest way to make T acyclic by vertex deletions can be obtained by finding the cheapest way to make L and P equal by vertex deletions; this is exactly the complement of the longest common subsequence of L and P. We then obtain the desired feedback vertex set for T by adding the vertices of this complement to those of the set R, which contains the vertices that were determined to be in any feedback vertex set in the reduction step (lines 10–11). Figure 7 shows an example for the execution of the subroutine from Figure 5.

In summary, the subroutine from Figure 5 is correct and can be used to solve Feedback Vertex Set in tournaments by iterative compression as described at the beginning of this section.

Theorem 4.1. *Using iterative compression,* Feedback Vertex Set *in tournaments can be solved in $O(2^k \cdot n^2(\log\log n + k))$ time.*

Proof. We have shown how to solve Feedback Vertex Set in tournaments using iterative compression. It remains to analyze the running time. First we examine the subroutine from Figure 5. Topological sort (line 1) can be easily done in $O(|S|) = O(k)$ time. Finding triangles in line 3 can be done in $O(nk)$ time: for every $v \in V \setminus S$, we iterate over the topological sort of S; if we encounter a vertex s_i with $(v, s_i) \in A$ and later a vertex s_j with $(s_j, v) \in A$, we have a triangle as desired. Line 8 needs $O(n)$ time, and by using bucket sort, line 9 can also be done in $O(n)$ time. Since L and P are permutations of each other, finding a longest common subsequence reduces to finding a longest increasing subsequence, which can be done in $O(n \log\log n)$ time [35]. In summary, the subroutine can be executed in $O(n(\log\log n + k))$ time. In the compression routine, the subroutine is called $O(2^k)$ times, once for each partition of X into two subsets. The compression routine itself is called n times when inductively building up the graph structure. In total, we have a running time of $O(2^k \cdot n^2(\log\log n + k))$. □

Using the $O(k^2)$-vertex 3-Hitting Set kernelization [1], we arrive at the following running time.

Theorem 4.2. *Using iterative compression,* Feedback Vertex Set *in tournaments can be solved in $O(2^k \cdot k^5 + n^3)$ time.*

Proof. All that the 3-Hitting Set kernelization of Abu-Khzam [1] does is to delete vertices that have to be deleted in any case (in other words, it yields an *induced* problem kernel, see [2]); therefore, we can easily transform the kernelized 3-Hitting Set instance that represents the set of triangles back into a Feedback Vertex Set instance. The kernelization takes $O(n^3)$ time and leaves $O(k^2)$ vertices, giving a running time of $O(n^3)+O(2^k \cdot k^4(\log\log k^2+k)) = O(2^k \cdot k^5 + n^3)$. □

We now show how to extend our results to the weighted case with rational weights $\omega \geq 1$:

Weighted Feedback Vertex Set in tournaments
Instance: A tournament $T = (V, A)$, a vertex weight function $\omega : V \to [1, \infty)$, and a number $t \geq 0$.
Question: Is there a subset $X \subseteq V$ with $\sum_{v \in X} \omega(x) \leq t$ such that deleting all vertices in X from G results in a directed acyclic graph?

Note that for arbitrary weights, in particular, with weights below 1 allowed, the problem is not fixed-parameter tractable unless P = NP, since otherwise we could solve FVST in polynomial time by scaling down the weights sufficiently.

We modify our algorithm solving unweighted FVST only in the last iteration of the iterative compression, where we have a feedback vertex set X of size at most $k + 1$ for T. Clearly, we can still enumerate all $O(2^k)$ possibilities of which part S to keep and which part to omit from X to get a minimum-weight solution X'. The data reduction (Figure 5 lines 3–5) is also still correct, and Lemma 4.1 and Corollary 4.1 hold. Therefore, again, the cheapest way to make T acyclic by vertex deletions can be obtained by finding the cheapest way to make L and P equal by vertex deletions; therefore, we need a *maximum-weight* common subsequence of L and P. Since L and P are permutations of each other, this reduces to finding a maximum-weight increasing subsequence, which in turn reduces to finding a maximum-weight independent set in a permutation graph: A permutation graph is a graph that has an intersection model consisting of straight lines (one per vertex) between two parallels. In our case, the permutation graph to be constructed has one vertex for every element of L; its edges are obtained by writing the two sequences L and P on two parallel

Originally published in *Journal of Discrete Algorithms*, 8(1):76–86. Elsevier B. V., 2010.

horizontal lines, one below the other, and connecting every element of L with its counterpart in P by a straight line. If two lines cross, the corresponding vertices in the graph are connected by an edge. A maximum-weight independent set in a permutation graph can be found in $O(n \log\log n)$ time [10]. Further, since a weighted optimal solution needs at least as many vertices as an unweighted optimal solution, and each vertex weighs at least 1, we have $t \geq k$. We arrive at the following result.

Theorem 4.3. *Weighted* Feedback Vertex Set *in tournaments can be solved in* $O(2^t \cdot n^2(\log\log n + t))$ *time.*

Woeginger [49] noted that by combining an algorithm by Schwikowski and Speckenmeyer [46] that enumerates all (inclusion-)minimal feedback vertex sets in a directed graph with polynomial delay with the fact that a tournament has at most 1.717^n minimal feedback vertex sets [39], one obtains an algorithm that solves FVST in $O(1.717^n)$ time. He asks whether this bound can be improved. One can sometimes gain fast exact algorithms by using an FPT algorithm for small parameter values and brute force only for large parameter values [43]. This approach can be applied here. We try all possible parameter values $k = 0, \dots, n$; if $k \leq \lambda n$, we use the $2^k \cdot n^{O(1)}$ algorithm of Theorem 4.1, and otherwise, we try by brute force all $\binom{n}{k}$ possible solutions. The running time of the brute force approach is maximum for $\lambda = 1/2$, since $\binom{n}{n/2} \approx 2^n$; therefore, we can improve the trivial 2^n bound if $\lambda > 1/2$. The optimal λ is attained when $2^{\lambda n} = \binom{n}{\lambda n}$, which gives (asymptotically) $\lambda \approx 0.773$. Thus, we can answer Woeginger's question affirmatively.

Theorem 4.4. Feedback Vertex Set *in tournaments can be solved in* $O(1.709^n)$ *time.*

5. Search Tree for Feedback Arc Set in Bipartite Tournaments

Raman and Saurabh [42] have shown that if a tournament T does not contain a particular four-vertex tournament denoted by F_1, then the cycles in T are pairwise vertex-disjoint. Using this, their $O(2.415^k \cdot n^{2.376})$-time algorithm solves FAST in a two-phase manner: First, it uses a depth-bounded search tree approach to get rid of all cycles contained in subtournaments F_1 appearing in T by reversing at most k arcs; this also destroys all subtournaments F_1 in T. In the second phase, in each tournament output by the search tree it destroys in polynomial time all remaining, pairwise disjoint triangles by reversing an arbitrary arc in each triangle. If after these two phases there is an acyclic tournament with at most k arcs reversed, then T has a feedback arc set with size at most k.

Following the same approach, we derive a fixed-parameter algorithm for Feedback Arc Set in bipartite tournaments (FASBT). In contrast to the algorithm for FAST [42], which needs only one simple-structured subtournament F_1 for characterizing instances that can be solved in polynomial time, here we need two subtournaments and a more involved branching strategy. We use the following lemma, which can immediately be seen by arguing analogously as in the case of Lemma 2.1.

Lemma 5.1. *A bipartite tournament is acyclic iff it contains no cycle of length four.*

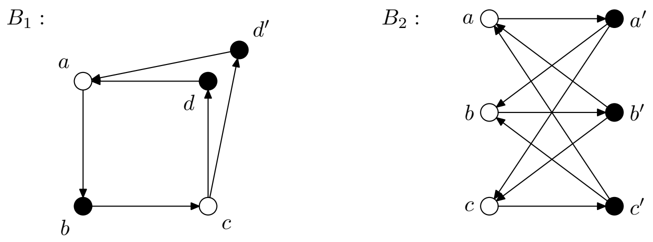

Figure 8: Forbidden subgraphs for bipartite tournaments where all cycles of length four are disjoint. The color of the vertices describes the bipartition.

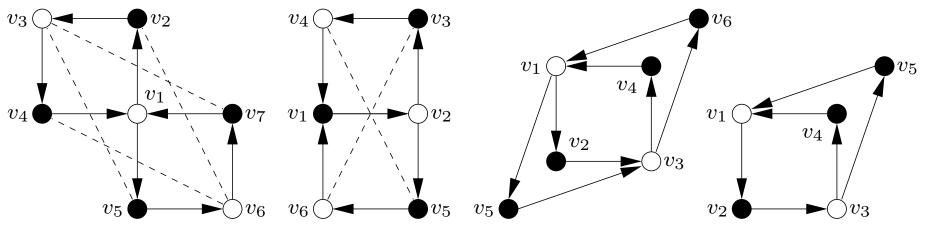

Figure 9: All possibilities for how two length-four cycles in a bipartite tournament can share at least one vertex. Left: Two cycles v_1, v_2, v_3, v_4 and v_1, v_5, v_6, v_7 sharing one vertex. Independent from the directions of the dashed arcs, one can always find an induced B_1. Second from left: Two cycles v_1, v_2, v_3, v_4 and v_1, v_2, v_5, v_6 sharing two consecutive vertices. Depending on the directions of the dashed arcs, one can find an induced B_1 or B_2. Second from right: Two cycles v_1, v_2, v_3, v_4 and v_1, v_5, v_3, v_6 sharing two non-consecutive vertices. One can find an induced B_1. Right: Two cycles v_1, v_2, v_3, v_4 and v_1, v_2, v_3, v_5 sharing three vertices. One can find an induced B_1.

Note that we cannot use Lemma 5.1 solve FASBT by first reducing the problem to 4-Hitting Set (with hyperedges corresponding to length-four cycles and vertices corresponding to arcs) and then using the algorithm for 4-Hitting Set from Section 3. The reason is that reversing an arc could create new length-four cycles.

By Lemma 5.1, in order to derive a forbidden subgraph characterization for bipartite tournaments where all cycles of length four are disjoint, we consider two length-four cycles in a bipartite tournament. If they are not vertex-disjoint, then they have one, two, or three common vertices. These three possibilities lead to bipartite tournaments which contain B_1 or B_2 shown in Figure 8 as induced subgraph. The following lemma strengthens this finding.

Lemma 5.2. *If a bipartite tournament B contains neither B_1 nor B_2 (shown in Figure 8) as an induced subgraph, then all cycles in B have length four and are pairwise vertex-disjoint.*

Proof. With Lemma 5.1, we first consider length-four cycles. By distinguishing three cases, namely two length-four cycles sharing one, two, and three vertices, respectively, and several subcases, one can show that a $\{B_1, B_2\}$-free bipartite tournament contains no two length-four cycles having a common vertex, see Figure 9. For example, if a cycle v_1, v_2, v_3, v_4 shares two vertices v_1, v_3 with a cycle v_1, v_5, v_3, v_6, then there is a B_1 with $a = v_1, b = v_2, c = v_3, d = v_4, d' = v_6$. Moreover, observe that in a bipartite tournament B, a subgraph of B induced by the vertices lying on a cycle with length greater than four contains several

Originally published in *Journal of Discrete Algorithms*, 8(1):76–86. Elsevier B. V., 2010.

Branching for B_1:	**Branching for B_2:**		
$\{(a,b)\}$,	$\{(a,a'),(b,b')\}$,	$\{(b,b'),(c',a)\}$,	$\{(a',b),(c',a),(c',b)\}$,
$\{(b,c)\}$,	$\{(a,a'),(c,c')\}$,	$\{(c,c'),(a',b)\}$,	$\{(b',a),(a',c),(b',c)\}$,
$\{(c,d),(c,d')\}$,	$\{(b,b'),(c,c')\}$,	$\{(c,c'),(b',a)\}$,	$\{(b',a),(a',c),(c',b)\}$,
$\{(c,d),(d',a)\}$,	$\{(a,a'),(b',c)\}$,	$\{(a',b),(a',c),(b',c)\}$,	$\{(b',a),(c',a),(b',c)\}$,
$\{(d,a),(c,d')\}$,	$\{(a,a'),(c',b)\}$,	$\{(a',b),(a',c),(c',b)\}$,	$\{(b',a),(c',a),(c',b)\}$
$\{(d,a),(d',a)\}$	$\{(b,b'),(a',c)\}$,	$\{(a',b),(c',a),(b',c)\}$,	

Figure 10: The branching for destroying induced subgraphs B_1 and B_2. In each subcase of the branching, all edges of one of the edge sets displayed in the figure are reversed.

length-four cycles which are not vertex-disjoint. Thus, a $\{B_1, B_2\}$-free bipartite tournament contains no cycle with a length greater than four. This completes the proof. □

Based on Lemma 5.2, our algorithm solving FASBT has the same two phases as the algorithm by Raman and Saurabh [42], namely a search tree algorithm destroying all cycles contained in the induced subgraphs B_1 and B_2 from Figure 8 and a polynomial-time second phase getting rid of the remaining, vertex-disjoint cycles.

Theorem 5.1. Feedback Arc Set *in bipartite tournaments of n vertices with k arc deletions can be solved in $O(3.373^k \cdot n^3)$ time.*

Proof. For destroying the cycles in B_1, the search tree algorithm makes a branching into six subcases, namely, reversing (a,b), reversing (b,c), reversing (c,d) and (c,d'), reversing (c,d) and (d',a), reversing (d,a) and (c,d'), and reversing (d,a) and (d',a). For each reversed arc, the parameter k is decreased by one. The size of depth-bounded search trees can be estimated using *branching vectors* [40]. The branching vector, which indicates how many arcs are reversed in every branch, here is $(1,1,2,2,2,2)$, corresponding to a search tree size of $O(3.24^k)$. Dealing with B_2, one branches into 17 subcases and, in each subcase, reverses two or three arcs (see Figure 10). The observation yielding the correctness of the branching is that a B_1 contains two length-four cycles, namely a,b,c,d and a,b,c,d', and that a B_2 contains three length-four cycles, namely a,a',b,b' and a,a',c,c' and b,b',c,c', which all have to be destroyed.

The worst-case running time is determined by the branching for B_2, with a search tree size of $O(3.373^k)$.

Finding one of B_1 and B_2 can be done in $O(n^3)$ time by searching for two non-disjoint cycles of length four; the subgraph induced by the vertices of these cycles contains a B_1 or B_2, which can be found in constant time in this subgraph. The two non-disjoint cycles can be found as follows: First, find a cycle C of length 4 in $O(n^2)$ time: start with a cycle C of arbitrary length, which can be found in $O(n+m)$ time with a depth first search, and repeatedly decrease its length by taking an arbitrary chord of C and considering the cycle formed by this chord and some of the edges of C. Second, search for a length-4 cycle that has at least one vertex in common with the length-4 cycle C. To find a length-4 cycle that has exactly one vertex in common with C, construct, for every vertex $v \in C$, a graph B' by inserting a copy v' of v into B, and search in $O(n+m)$ time for a path from v to v' in $B'[(V \setminus C) \cup \{v, v'\}]$. To find a length-4 cycle that has two or three vertices in common with C, try for $i = 2, 3$ every combination of i vertices from C and $4 - i$ vertices from $V \setminus C$ and check in constant time whether these four vertices form a cycle. If no cycle of length 4

Originally published in *Journal of Discrete Algorithms*, 8(1):76–86. Elsevier B. V., 2010.

can be found that is non-disjoint from C, the search for a B_1 or B_2 is continued in $B[V \setminus C]$; hence, at most $\lfloor n/4 \rfloor$ cycles C have to be considered to find a B_1 or a B_2.

When destroying vertex-disjoint cycles in the second phase of the search tree algorithm, reversing arcs on cycles does not generate new cycles because, due to Lemma 2.2, reversing an arc is "equivalent" to deleting an arc and because, due to Lemma 5.2, there is no cycle of length greater than four after the first phase of the search-tree algorithm. Thus, the second phase of the algorithm can be executed in $O(n^3)$ time by repeatedly searching for a length-4 cycle and reversing an arbitrary arc of each found cycle. Note that there can be at most $\lfloor n/4 \rfloor$ vertex-disjoint cycles. □

6. Conclusion

In this work, we presented improved fixed-parameter algorithms for Feedback Vertex Set in tournaments, 4-Hitting Set, and 5-Hitting Set. The algorithm for FVST also implies an exact algorithm for this problem running in $O(1.709^n)$, answering a question of Woeginger [49]. Herein, the iterative compression technique plays a central role. Finally, we gave a size-$O(3.373^k)$ search tree algorithm for Feedback Arc Set in bipartite tournaments, based on a new forbidden subgraph characterization for bipartite tournaments.

A natural research topic resulting from our work is to examine the applicability of the iterative compression technique to Feedback Arc Set in (bipartite) tournaments, as we did here for the vertex version. The most difficult part might be to design an iteration process. The iterative buildup of the input graph by adding vertices or edges one-by-one seems infeasible for the arc version: Adding vertices cannot guarantee the size of the solution to be compressed, whereas adding edges destroys the tournament property. Another interesting research line would be to improve the problem kernel size of all these feedback set problems. Nowadays, only polynomial-size kernels are known. Linear-size kernels would be very desirable from an applied point of view.

References

[1] F. N. Abu-Khzam. Kernelization algorithms for d-hitting set problems. In *Proceedings of the 10th Workshop on Algorithms and Data Structures (WADS '07)*, volume 4619 of *LNCS*, pages 434–445. Springer, 2007.

[2] F. N. Abu-Khzam and H. Fernau. Kernels: Annotated, proper and induced. In *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC '06)*, volume 4169 of *LNCS*, pages 264–275. Springer, 2006.

[3] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX '04)*, pages 62–69. SIAM, 2004.

[4] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5), Article 23, 2008.

[5] N. Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics*, 20(1):137–142, 2006.

[6] N. Alon, D. Lokshtanov, and S. Saurabh. Fast FAST. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP '09))*, volume 5555 of *LNCS*, pages 49–58. Springer, 2009.

[7] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications.* Springer, 2002.

[8] M.-C. Cai, X. Deng, and W. Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM Journal on Computing*, 30(6): 1993–2007, 2001.

[9] M.-C. Cai, X. Deng, and W. Zang. A min-max theorem on feedback vertex sets. *Mathematics of Operations Research*, 27(2):361–371, 2002.

[10] M.-S. Chang and F.-H. Wang. Efficient algorithms for the maximum weight clique and maximum weight independent set problems on permutation graphs. *Information Processing Letters*, 43(6):293–295, 1992.

[11] P. Charbit, S. Thomassé, and A. Yeo. The minimum feedback arc set problem is NP-hard for tournaments. *Combinatorics, Probability and Computing*, 16:1–4, 2007.

[12] I. Charon and O. Hudry. A survey on the linear ordering problem for weighted or unweighted tournaments. *4OR: A Quarterly Journal of Operations Research*, 5(1):5–60, 2007.

[13] J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In *Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS '06)*, volume 4162 of *LNCS*, pages 238–249. Springer, 2006.

[14] J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008.

[15] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55(5), Article 21, 2008.

[16] V. Conitzer. Computing Slater rankings using similarities among candidates. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI '06)*, pages 613–619. AAAI Press, 2006.

[17] F. K. H. A. Dehne, M. R. Fellows, M. A. Langston, F. A. Rosamond, and K. Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory of Computing Systems*, 41(3):479–492, 2007.

[18] I. Dinur and S. Safra. On the hardness of approximating minimum vertex-cover. *Annals of Mathematics*, 162(1):439–486, 2005.

[19] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[20] A. Felner, R. E. Korf, and S. Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 21:1–39, 2004.

[21] H. Fernau. A top-down approach to search-trees: Improved algorithmics for 3-hitting set. Technical Report TR04-073, Electronic Colloquium on Computational Complexity, 2004. To appear in *Algorithmica*.

[22] H. Fernau. *Parameterized Algorithmics: A Graph-Theoretic Approach.* Habilitationsschrift, Universität Tübingen, 2005.

[23] P. Festa, P. M. Pardalos, and M. G. C. Resende. Feedback set problems. In D. Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization, Vol. A*, pages 209–258. Kluwer, 1999.

[24] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer, 2006.

[25] F. Fomin, S. Gaspers, D. Kratsch, M. Liedloff, and S. Saurabh. Iterative compression and exact algorithms. In *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS '08)*, volume 5162 of *LNCS*, pages 335–346. Springer, 2008.

[26] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[27] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

[28] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006.

[29] J. Guo, F. Hüffner, and H. Moser. Feedback arc set in bipartite tournaments is NP-complete. *Information Processing Letters*, 102(2–3):62–65, 2007.

[30] J. Guo, H. Moser, and R. Niedermeier. Iterative compression for exactly solving NP-hard minimization problems. In *Algorithmics of Large and Complex Networks*, volume 5515 of *LNCS*, pages 65–80. Springer, 2009.

[31] S. Gupta. Feedback arc set problem in bipartite tournaments. *Information Processing Letters*, 105(5):150–154, 2008.

[32] G. Gutin and A. Yeo. Some parameterized problems on digraphs. *The Computer Journal*, 51(3):363–371, 2008.

[33] F. Hüffner. *Algorithms and Experiments for Parameterized Approaches to Hard Graph Problems.* PhD thesis, Institut für Informatik, Friedrich-Schiller Universität Jena, 2007.

[34] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. In *Proceedings of the 8th Latin American Theoretical Informatics Symposium (LATIN '08)*, volume 4598 of *LNCS*, pages 711–722. Springer, 2008. To appear in *Theory of Computing Systems*.

[35] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

[36] C. Kenyon-Mathieu and W. Schudy. How to rank with few errors: A PTAS for weighted feedback arc set on tournaments. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC '07)*, pages 95–103. ACM, 2007.

[37] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2 - \epsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[38] D. E. Knuth. *The Art of Computer Programming*, volume 4. Addison–Wesley, 2004.

[39] J. W. Moon. On maximal transitive subtournaments. *Proceedings of the Edinburgh Mathematical Society*, 17:345–349, 1971.

[40] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[41] R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003.

[42] V. Raman and S. Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006.

[43] V. Raman, S. Saurabh, and S. Sikdar. Efficient exact algorithms through enumerating maximal independent sets and other techniques. *Theory of Computing Systems*, 41(3):563–587, 2007.

[44] P. Sasatte. Improved FPT algorithm for feedback vertex set problem in bipartite tournament. *Information Processing Letters*, 105(3):79–82, 2007.

[45] P. Sasatte. Improved approximation algorithm for the feedback set problem in a bipartite tournament. *Operations Research Letters*, 36(5):602–604, 2008.

[46] B. Schwikowski and E. Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics*, 117(1–3):253–265, 2002.

[47] E. Speckenmeyer. On feedback problems in digraphs. In *Proceedings of the 15th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '89)*, volume 411 of *LNCS*, pages 218–231. Springer, 1989.

[48] M. Wahlström. *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems.* PhD thesis, Department of Computer and Information Science, Linköpings universitet, Sweden, 2007.

[49] G. J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.

[50] A. van Zuylen. Linear programming based approximation algorithms for feedback set problems in bipartite tournaments. In *Proceedings of the 6th Annual Conference on Theory and Applications of Models of Computation (TAMC '09)*, volume 5532 of *LNCS*, pages 370–379. Springer, 2009.