



Dynamic Parameterized Feedback Problems in Tournaments

Anna Zych-Pawlewicz  

Institute of Informatics, University of Warsaw, Poland

Marek Żochowski 

Institute of Informatics, University of Warsaw, Poland

Abstract

In this paper we present the first dynamic algorithms for the problem of K-FEEDBACK ARC SET IN TOURNAMENTS (K-FAST) and the problem of K-FEEDBACK VERTEX SET IN TOURNAMENTS (K-FVST). Our algorithms maintain a dynamic tournament on n vertices altered by redirecting the arcs, and answer if the tournament admits a feedback arc set (or respectively feedback vertex set) of size at most K , for some chosen parameter K . For dynamic K-FAST we offer two algorithms. In the promise model, where we are guaranteed, that the size of the solution does not exceed $g(K)$ for some computable function g , we give an $O(\sqrt{g(K)})$ update and $O(3^K K \sqrt{K})$ query algorithm. In the general setting without any promise, we offer an $O(\log^2 n)$ update and $O(3^K K \log^2 n)$ query time algorithm for dynamic K-FAST. For dynamic K-FVST we offer an algorithm working in the promise model, which admits $O(g^5(K))$ update and $O(3^K K^3 g(K))$ query time.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases dynamic algorithms, parameterized algorithms, feedback arc set, feedback vertex set, tournaments

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.22

Related Version *Full Version:* <https://arxiv.org/abs/2404.12907> [28]

Funding National Science Centre, Poland, Grant 2017/26/D/ST6/00264

1 Introduction and Related Work

In this paper we study feedback set problems in dynamic tournaments. A tournament is a directed graph where every pair of vertices is connected by exactly one arc. The feedback arc (resp. vertex) set of a given directed graph is a set of arcs (resp. vertices) whose removal makes the graph acyclic. The problems we focus on here are the FEEDBACK ARC SET IN TOURNAMENTS and the FEEDBACK VERTEX SET IN TOURNAMENTS. In the classical static setting, these problems are very well known and are defined as follows:

- K-FEEDBACK ARC SET IN TOURNAMENTS (K-FAST): Given a tournament $T = (V, E)$ and a positive integer K , does there exist a subset $F_E \subseteq E$ of at most K arcs whose removal makes T acyclic.
- K-FEEDBACK VERTEX SET IN TOURNAMENTS (K-FVST): Given a tournament $T = (V, E)$ and a positive integer K , does there exist a subset $F_V \subseteq V$ of at most K vertices whose removal makes T acyclic.

Both the above problems are flag problems in the area of parameterized complexity and textbook examples for the branching technique. Feedback arc sets in tournaments have applications in voting systems and rank aggregation, and are well studied from the combinatorial [9, 11, 25] and algorithmic [27, 16, 6, 3, 10, 12] points of view. Unfortunately, the K-FAST problem is NP-hard [2]. The fastest parameterized algorithm achieves $2^{O(\sqrt{K})} n^{O(1)}$ running time [12], where n is the size of the input tournament. In this paper, however, we



© Anna Zych-Pawlewicz and Marek Żochowski;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 22; pp. 22:1–22:15

Leibniz International Proceedings in Informatics



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

explore a textbook branching algorithm for this problem running in $3^K n^{O(1)}$ time [24]. The K-FVST problem also has many interesting applications, for instance in social choice theory where it is essential to the definition of a certain type of election winners [4]. It is also NP-hard [26] and has been studied from various angles [19, 20, 13, 22]. The fastest currently known parameterized algorithm for this problem runs in $O(1.6181^K + n^{O(1)})$ time [19].

Recently, there is a growing interest in studying parameterized problems in a dynamic setting. In this context, we typically consider an instance I of a problem of interest with an associated parameter K . The instance is dynamic, i.e., it is updated over time by problem specific updates, while for simplicity we assume that the problem parameter K does not change through the entire process. The goal is to provide a data structure, that allows for efficient updates to I , and upon a query efficiently provides the answer to the problem in question. The update/query running time may depend in any computable way on the parameter K , but it should be sublinear in terms of the size of I . The typical update/query running times in this setting are $f(K)$, $f(K)(\log n)^{O(1)}$, or sometimes even $f(K)n^{o(1)}$, where f is some computable function and n is the size of I . Since we allow f to be exponential, this setting applies to NP-hard problems as long as they are fixed-parameter tractable in the static setting.

Parameterized dynamic data structures were first systematically investigated by Iwata and Oka [15], followed by Alman et al. [1]. These works provided data structures with update times $f(K)$ or $f(K) \cdot (\log n)^{O(1)}$ for several classic problems such as VERTEX COVER, CLUSTER VERTEX DELETION, HITTING SET, FEEDBACK VERTEX SET, or LONGEST PATH. Other recent advances include data structures for maintaining various graph decompositions together with runs of dynamic programming procedures [5, 7, 8, 17, 21] and treatment of parameterized string problems from the dynamic perspective [23].

Alman et al. in their work [1] study the problem of K-FEEDBACK VERTEX SET in dynamic undirected graphs, where the graph is altered by edge additions and removals and the goal is to test if the graph has a feedback vertex set of size at most K . For this problem, Alman et al. propose a dynamic algorithm with $2^{O(K \log K)} \log^{O(1)} n$ amortized update time and $O(K)$ query time. It is worth mentioning, that while the K-FEEDBACK VERTEX SET problem is NP-hard on undirected graphs, the K-FEEDBACK ARC SET problem is polynomial in this class of graphs and can be efficiently maintained dynamically using dynamic connectivity algorithms [14]. So an interesting question is whether these two problems admit efficient dynamic algorithms in the class of directed graphs. In this regard Alman et al. [1] show lower bounds for the K-FEEDBACK VERTEX SET problem, which extend also to the K-FEEDBACK ARC SET problem. To be more precise, they show that in this case the dynamic algorithm requires $\Omega(f(K)m^\delta)$ update/query time for some fixed $\delta > 0$, assuming RO hypothesis (see [1]). Thus, a natural question is whether we can have more efficient dynamic algorithms at least in tournaments. In this paper we give positive answers to this question.

Our precise setting is the following. With regard to dynamic K-FAST, the goal is to design a data structure supporting the following operations:

- **Initialize**(T, n) - initialize the data structure with a given tournament T on n vertices
- **Reverse**(u, v) - reverse an arc in T between two vertices u and v of T
- **FindFAST**() - answer if there is a feedback arc set of size at most K in T .

With regard to dynamic K-FVST, the goal is to design a data structure supporting the analogous operations:

- **Initialize**(T, n) - initialize the data structure with a given tournament T on n vertices
- **Reverse**(u, v) - reverse an arc in T between two vertices u and v of T
- **FindFVST**() - answer if there is a feedback vertex set of size at most K in T .

■ **Table 1** The known results for dynamic K-FEEDBACK ARC SET. and dynamic K-FEEDBACK VERTEX SET for different classes of graphs.

	Undirected Graphs	Directed Graphs	Tournaments
K-FEEDBACK ARC SET	$\log^{\mathcal{O}(1)} n$ update $\log^{\mathcal{O}(1)} n$ query [14]	no $f(K)m^{\mathcal{O}(1)}$ algorithm assuming RO hypothesis [1]	full model: $\mathcal{O}(\log^2 n)$ update $\mathcal{O}(3^K K \log^2 n)$ query promise model: $\mathcal{O}(\sqrt{g(K)})$ update $\mathcal{O}(3^K K \sqrt{K})$ query
K-FEEDBACK VERTEX SET	$2^{\mathcal{O}(K \log K)} \log^{\mathcal{O}(1)} n$ amortized update $\mathcal{O}(K)$ query [1]	no $f(K)m^{\mathcal{O}(1)}$ algorithm assuming RO hypothesis [1]	promise model: $\mathcal{O}(g^5(K))$ update $\mathcal{O}(3^K K^3 g(K))$ query

The above setting is referred to as the *full* model. A popular restriction of this setting, introduced in generality by Alman et. al. [1], is called the *promise* model. The promise model applied to our setting provides the data structure with a guarantee, that the feedback arc set (or respectively feedback vertex set) remains of size bounded by $g(K)$ for some computable function g for the entire process. Some algorithms provided by Alman et al. [1] work only in this restricted setting.

Our results

For the dynamic K-FAST problem we offer two data structures. In the promise model we propose an $\mathcal{O}(\sqrt{g(K)})$ update and $\mathcal{O}(3^K K \sqrt{K})$ query data structure which does not need to know $g(K)$. In the full model, we offer an $\mathcal{O}(\log^2 n)$ update and $\mathcal{O}(3^K K \log^2 n)$ query data structure. For the dynamic K-FVST problem we offer a data structure which works in the promise model, with $\mathcal{O}(g^5(K))$ update and $\mathcal{O}(3^K K^3 g(K))$ query time. This data structure does need to know $g(K)$. Our results for dynamic feedback set problems, compared to the related results of Alman et al. [1] are shown in Table 1. All our running times are worst case (i.e., not amortized).

As a side result, we propose two dynamic data structures that can efficiently find triangles (i.e., directed cycles of length three) in dynamic tournaments. The efficiency of these data structures depends on parameter $\text{ADT}(T)$, which is the maximum number of arc-disjoint triangles in T . The first data structure admits $\mathcal{O}(\sqrt{\text{ADT}(T)})$ update time and $\mathcal{O}(\text{ADT}(T)\sqrt{\text{ADT}(T)})$ query time. The second data structure admits $\mathcal{O}(\log^2 n)$ update time and $\mathcal{O}(\text{ADT}(T) \log^2 n)$ query time. The data structures do not need to know $\text{ADT}(T)$. We believe that the triangle detection data structures may be of independent interest.

In the next sections we provide an illustrative overview of our techniques and ideas used to obtain our results, with the main focus on the dynamic K-FAST problem in the promise model. The rigorous proofs, as well as details for the dynamic K-FAST problem in the full model and dynamic K-FVST problem in the promise model are moved to the full version [28] due to space limitations.

2 Triangle Detection Data Structures

For both the dynamic K-FAST and the dynamic K-FVST problem, in order to answer the query efficiently, our plan is to run the standard (static) branching algorithms (see Algorithm 1 and Algorithm 2 respectively) upon every query. The branching algorithm for K-FAST relies

on the folklore knowledge that the minimum feedback arc set in a tournament is equivalent to the minimum set of arcs whose reversal makes the tournament acyclic. It is also a folklore fact that a tournament has a cycle if and only if it has a cycle of length three (that we refer to as a triangle). These folklore facts together with proofs can be found in [24]. For both K-FAST and K-FVST the branching algorithm executes at most 3^K recursive calls. In each recursive call the algorithm finds a triangle in the tournament and tries to reverse each of its edges (for the K-FAST problem) or remove each of its vertices (for the K-FVST problem). Thus, to implement the K-FAST query efficiently in the dynamic setting, we need a data structure that can quickly find a triangle in a dynamic tournament altered by arc reversals. For the dynamic K-FVST problem, the data structure also needs to allow removing a limited number of vertices.

Hence, the basis for our algorithms are the triangle detection data structures, that might be of independent interest. The data structures we provide are stated independently of the feedback set problems. They rely on a different parameter, which is the maximum number of arc-disjoint triangles in the tournament. Nevertheless, our later results rely on a close connection between the maximum number of arc-disjoint triangles and the minimum feedback arc set of a tournament. Throughout the paper, for a given tournament T , we denote by $\text{FAST}(T)$ the size of the minimum feedback arc set in T , by $\text{ADT}(T)$ the maximum number of arc-disjoint triangles in T , and by $\text{FVST}(T)$ the size of the minimum feedback vertex set in T . The following fact holds.

► **Fact 1.** $\text{ADT}(T) \leq \text{FAST}(T) \leq 6(\text{ADT}(T) + 1)$.

Proof. It is easy to see that $\text{ADT}(T) \leq \text{FAST}(T)$, as in each of the $\text{ADT}(T)$ arc-disjoint triangles one arc must be taken to the feedback arc set of T . The proof of the second inequality can be found in [18] (Theorem 4). ◀

The triangle detection data structures maintain the dynamic tournament altered by reversing arcs, and allow queries for a triangle in the maintained tournament. The first data structure is given by Theorem 2 below. It is later used for dynamic K-FAST in the promise model.

► **Theorem 2** (Theorem 15 in Appendix A in [28]). *For any integer $n \in \mathbb{N}$ there exists a data structure $\mathbb{DTP}[n]$, that maintains a dynamically changing tournament T on n vertices¹ by supporting the following operations:*

1. **Initialize**(T, n) – initializes the data structure with a given tournament T on n vertices, in time $\mathcal{O}(n^2)$
2. **Reverse**(v, u) – reverses an arc between vertices v and u in T , in $\mathcal{O}(\sqrt{\text{ADT}(T)})$ time
3. **FindTriangle**() – returns a triangle from T or reports that there are no triangles, in time $\mathcal{O}(\text{ADT}(T)\sqrt{\text{ADT}(T)})$

We note here that the above data structure does not need to know the value of $\text{ADT}(T)$. The same holds for the second data structure presented next. The second triangle detection data structure gets rid of the dependence on the parameter in the update operation. This later allows us to use it for dynamic K-FAST in the full model at the cost of introducing factors poly-logarithmic in the size of the tournament.

► **Theorem 3** (Theorem 16 in Appendix A in [28]). *For any integer $n \in \mathbb{N}$ there exists a data structure $\mathbb{DT}[n]$, that maintains a dynamically changing tournament T on n vertices¹ by supporting the following operations:*

1. **Initialize**(T, n) – initializes the data structure with a given tournament T on n vertices, in time $\mathcal{O}(n^2)$
2. **Reverse**(v, u) – reverses an arc between vertices v and u in T , in time $\mathcal{O}(\log^2 n)$
3. **FindTriangle**() – returns a triangle from T or reports that there are no triangles, in time $\mathcal{O}(\text{ADT}(T) \log^2 n)$

Both data structures are described in detail in Appendix A in [28]. In this overview, we mainly focus on describing the first data structure $\mathbb{DTTP}[n]$ of Theorem 2, which also sheds some light on the data structure $\mathbb{DT}[n]$ of Theorem 3, as both data structures are based on similar main ideas.

In particular, both data structures maintain the same basic information related to the dynamic tournament T . First and foremost, both data structures maintain n sets called *indegree buckets*, which partition the vertices of T according to their indegrees in T . The indegree bucket $\text{Db}_T[d]$ stores vertices of indegree d . The indegree buckets alone let us easily determine if the tournament is acyclic due to the following fact (which can be found for instance in [10]).

► **Fact 4** (Fact 19 in Appendix A.1 in [28]). *A tournament is acyclic if and only if all its indegree buckets have size one.*

By $\max_{\text{Db}}(T)$ we denote the maximum size of an indegree bucket for the maintained tournament T . It is easy to see, that $\max_{\text{Db}}(T) \leq 2\text{FAST}(T) + 1 \in \mathcal{O}(\text{ADT}(T))$, because one arc reversal can remove at most two vertices from the indegree bucket of maximum size, and we have to remove all but one to make the tournament acyclic. With a bit more care one can show the following bound.

► **Fact 5** (Lemma 25 in Appendix A.1 in [28]). $\max_{\text{Db}}(T) \leq 8\sqrt{\text{ADT}(T) + 1} + 8$.

Both data structures also maintain a set **Empty** of indices of indegree buckets that are empty, i.e., $\text{Empty} = \{d \in \{0, \dots, n-1\} : \text{Db}_T[d] = \emptyset\}$. Since each arc reversal can place up to two vertices in the empty buckets, and since by Fact 4 there are no empty buckets after reversing the arcs of a minimum feedback arc set, the following bound holds.

► **Fact 6** (Lemma 27 in Appendix A.1 in [28]). $|\text{Empty}| \leq 2\text{FAST}(T) \in \mathcal{O}(\text{ADT}(T))$

Both the indegree buckets and the set **Empty** are straightforward to maintain in a constant time per arc reversal, since every arc reversal affects the indegrees of exactly two vertices (see Lemma 35 in [28] for the details). The data structures also rely on some other information that is straightforward to maintain, such as adjacency matrix of the tournament and similar basic structures. These are not crucial enough to be mentioned in this short description, but they are detailed in Appendix A in [28] instead.

The promise data structure $\mathbb{DTTP}[n]$ additionally maintains a set **Back** of *back arcs*. An arc uv is called a back arc if $d_T(u) \geq d_T(v)$, where $d_T(x)$ stands for the indegree of x in T . The back arcs are another natural obstacle to T being acyclic, as every back arc belongs to some triangle in T (see Lemma 21 in Appendix A in [28]). Reversing the arcs of a minimum feedback arc set of T gets rid of all cycles and thus also gets rid of all back arcs in T .

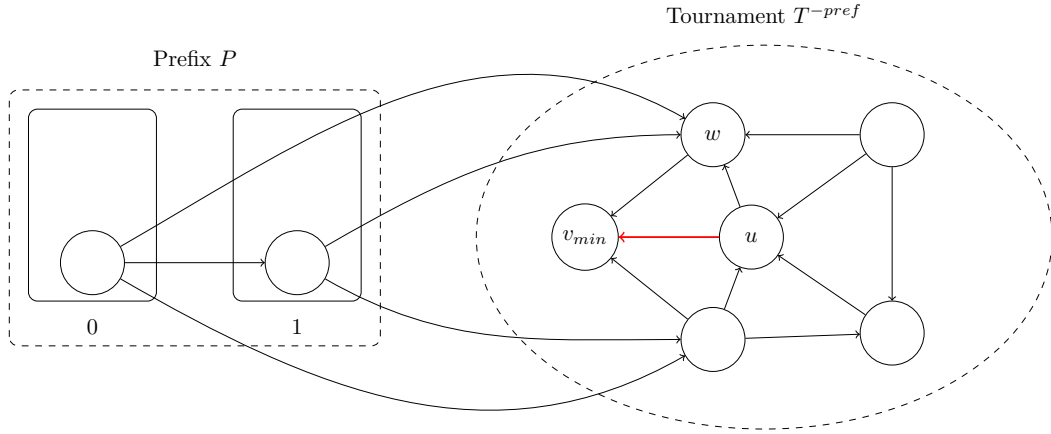
The set **Back** changes its size by at most $\mathcal{O}(\sqrt{\text{ADT}(T)})$ upon arc reversal for the following reason. Consider reversing arc uv . Any arc that after the reversal becomes a back arc or stops being a back arc has one endpoint in either u or v . The candidates for the other endpoint of

¹ All our data structures assume that the vertices of T are indexed with numbers from 0 to $n-1$.

such arc have indegrees differing by at most one from $d_T(u)$ or $d_T(v)$ (see Lemma 31 in [28] for more details). This gives (by Fact 5) $O(\sqrt{\text{ADT}(T)})$ candidate arcs for altering the set **Back**. This observation has two important consequences. Firstly, maintaining the set of back arcs **Back** takes $O(\sqrt{\text{ADT}(T)})$ time per update. Secondly, we get the bound for the size of the set of back arcs **Back**.

► **Fact 7** (Lemma 33 in Appendix A.1 in [28]). $|\text{Back}| \in O(\text{ADT}(T)\sqrt{\text{ADT}(T)})$

Fact 7 follows for the following reason. Reversing $\text{FAST}(T)$ arcs gets rid of all back arcs, and one arc reversal gets rid of at most $O(\sqrt{\text{ADT}(T)})$ back arcs. Thus, $|\text{Back}| \in O(\text{FAST}(T)\sqrt{\text{ADT}(T)})$. By Fact 1 the bound of Fact 7 follows.



■ **Figure 1** Tournament T , its prefix P and tournament $T^{-\text{pref}}$. Arcs between all pairs of vertices are present in T , however some arcs are not drawn for the sake of readability.

We now move on to describing how our data structures detect triangles. To support this operation, we first define the *prefix* P of a tournament T as the set of vertices in the maximum prefix of indegree buckets of size one (see Figure 1 for illustration and Definition 22 in [28] for a more formal definition). The subtournament $T[P]$ of T induced by its prefix P is acyclic by Fact 4. Since we cannot find triangles inside the prefix P , we are more interested in tournament $T^{-\text{pref}} = T[V(T) \setminus P]$ which stands for tournament T induced by all its vertices excluding prefix. Both the prefix P of a tournament T and the remaining tournament $T^{-\text{pref}}$ are illustrated in Figure 1.

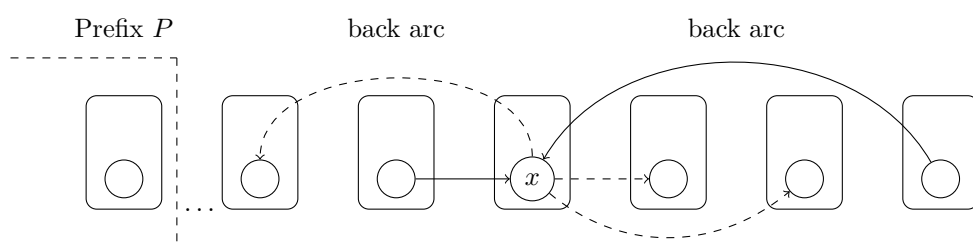
The data structures $\mathbb{DTP}[n]$ and $\mathbb{DT}[n]$ follow the same general approach. In order to find a triangle, they first search for a vertex v_{\min} of minimum indegree in $T^{-\text{pref}}$ (see Figure 1). One can find the vertex v_{\min} in $O(|\text{Empty}|) = O(\text{ADT}(T))$ time by iterating through the set **Empty**. Clearly, the indegree buckets whose indices range from 0 to $|P| - 1$ are not empty, by the definition of prefix P . Since we want to skip vertices of P , we are interested in the first index larger than $|P| - 1$ which is not in **Empty**. This is the index of the bucket containing v_{\min} . We have the following bound on $d_{T^{-\text{pref}}}(v_{\min})$, which is the minimum indegree in $T^{-\text{pref}}$.

► **Fact 8** (Fact 28 in Appendix A.1 in [28]). $d_{T^{-\text{pref}}}(v_{\min}) \leq |\text{Empty}|$.

Fact 8 follows, because in $T^{-\text{pref}}$ all indegree buckets $\text{Db}_{T^{-\text{pref}}}[d]$ for $d < d_{T^{-\text{pref}}}(v_{\min})$ are empty. The number of empty indegree buckets of $T^{-\text{pref}}$ is the same as the number of empty indegree buckets of T , because $d_{T^{-\text{pref}}}(w) = d_T(w) - |P|$ for any vertex w of $T^{-\text{pref}}$.

Once v_{\min} is located, the data structures find an in-neighbor u of v_{\min} in $T^{-\text{pref}}$. Next, the data structures find a set W of $d_{T^{-\text{pref}}}(v_{\min})$ in-neighbors of u in $T^{-\text{pref}}$. Since $d_{T^{-\text{pref}}}(v_{\min}) \leq d_{T^{-\text{pref}}}(u)$, vertex u has at least $d_{T^{-\text{pref}}}(v_{\min})$ in-neighbors in $T^{-\text{pref}}$. We are bound to find a triangle $uv_{\min}w$ for some $w \in W$, because if all the arcs between W and v_{\min} were directed towards v_{\min} , that would imply that the indegree of v_{\min} in $T^{-\text{pref}}$ is more than $d_{T^{-\text{pref}}}(v_{\min})$. This is illustrated in Figure 1.

Thus, in order to find a triangle, we need an *in-neighbour listing* method to list l in-neighbours in $T^{-\text{pref}}$ of a given vertex $x \in T^{-\text{pref}}$. The method responsible for finding in-neighbours is different for the two data structures. In the promise data structure $\mathbb{DTTP}[n]$, this method (provided in detail in Lemma 40 in [28]) heavily relies on the fact that we can iterate over the set of back arcs Back . The process of finding in-neighbours by $\mathbb{DTTP}[n]$ data structure is illustrated in Figure 2.



■ **Figure 2** In-neighbour listing method in the promise model.

In order to find the in-neighbours of a given vertex x , the $\mathbb{DTTP}[n]$ data structure iterates over the indegree buckets of T starting after the prefix. For each $d > |P|$ such that $d \leq d_T(x)$, we proceed as follows. If $\text{Db}_T[d]$ is empty, we charge it to $|\text{Empty}|$ and move forward. Otherwise we iterate through vertices $w \in \text{Db}_T[d]$. If w is an in-neighbour of x , then we add it to the set of found in-neighbours W , whose size is bounded by parameter l . If w is an out-neighbour of v , then xw is a back arc and we charge such situation to $|\text{Back}|$. If by the time we reach $d = d_T(v)$ the size of W is not sufficient, we iterate through the set Back to find the remaining in-neighbours of x . The whole process takes $O(|\text{Back}| + |\text{Empty}| + l)$ time (see Lemma 40 in [28] for more details). By Fact 6, Fact 7 and Fact 8 we get the desired running time of the triangle query in the promise model.

The $\mathbb{DT}[n]$ data structure takes a different approach to find in-neighbours of a given vertex, as it does not have access to the set Back . Instead, it uses balanced search trees, what implies poly logarithmic factors in the update and query times. Due to space limitations, we defer the description of the $\mathbb{DT}[n]$ data structure to Appendix A.4 in [28].

3 Dynamic K-FEEDBACK ARC SET IN TOURNAMENTS

In this section we use the triangle detection data structures for dynamic K-FAST problem. We only focus on the main ideas, while the details are presented in Appendix B in [28]. Let us consider a standard branching algorithm which verifies whether $\text{FAST}(T) \leq K$ (shown in Algorithm 1, see [24] for correctness). Algorithm 1 can be implemented using both triangle detection data structures provided in Section 2.

We first show how to obtain the dynamic K-FAST data structure in the promise model, where $\text{FAST}(T) \leq g(K)$ at all times. Then, by Fact 1, also $\text{ADT}(T) \leq g(K)$. The procedure $\text{FindFAST}(K)$ in Algorithm 1 calls itself recursively at most 3^K times, each call employs

■ **Algorithm 1** Pseudocode for FindFAST(K).

```

Algorithm : FindFAST(K)
Output    : Verify if  $\text{FAST}(T) \leq K$ 
1 if  $T$  is acyclic then
2 |   return TRUE ;
3 if  $K = 0$  then
4 |   return FALSE ;
5  $uvw \leftarrow \text{FindTriangle}()$ ;
6 for  $xy \in \{uv, vw, wu\}$  do
7 |    $\text{Reverse}(x, y)$  ;
8 |   if FindFAST( $K - 1$ ) then
9 | |    $\text{Reverse}(y, x)$ ;
10 | |   return TRUE ;
11 |    $\text{Reverse}(y, x)$ ;
12 return FALSE ;

```

a constant number of updates and queries to a triangle detection data structure. Thus, directly by Theorem 2, employing the data structure $\mathbb{DTP}[n]$ results in a dynamic K-FAST data structure with $\mathcal{O}(3^K g(K) \sqrt{g(K)})$ query time and $\mathcal{O}(\sqrt{g(K)})$ update time.

The query time can be improved to $\mathcal{O}(3^K K \sqrt{K})$ by using procedure $\text{FindTriangle}(K)$ instead of $\text{FindTriangle}()$. The $\text{FindTriangle}(K)$ procedure works analogously to the procedure $\text{FindTriangle}()$, but has an option not to return a triangle once it detects that $\text{ADT}(T) > K$. In such case by Fact 1 also $\text{FAST}(T) > K$, and the recursive call can safely return that the solution does not exist. This is very helpful, as the procedure $\text{FindTriangle}(K)$ can stop working once any of the bounds on $|\text{Empty}|$, $|\text{Back}|$ or $\max_{\text{Db}}(T)$ (given by Fact 5, Fact 6 and Fact 7) that we have in terms of $\text{ADT}(T)$ does not hold in terms of K . For instance, if $\max_{\text{Db}}(T) > 8\sqrt{K} + 1 + 8$, then by Fact 5 the method $\text{FindTriangle}(K)$ is allowed to terminate returning that $\text{ADT}(T) > K$. In this way we obtain a better result stated below.

► **Theorem 9** (Theorem 49 in Appendix B in [28]). *The dynamic K-FAST problem admits a data structure with initialization time $\mathcal{O}(n^2)$, worst-case update time $\mathcal{O}(\sqrt{g(K)})$ and worst-case query time $\mathcal{O}(3^K K \sqrt{K})$ under the promise that there is a computable function g , such that the maintained tournament T always has a feedback arc set of size at most $g(K)$.*

Moving on from the promise model to the full model, we can employ the data structure $\mathbb{DT}[n]$ of Theorem 3 to implement the $\text{FindFAST}(K)$ method given in Algorithm 1. There, we cannot use $\text{FindTriangle}()$ method anymore, as its running time depends on $\text{ADT}(T)$, which is not bounded as in the promise model. Instead, we again use $\text{FindTriangle}(K)$ method, which again reports $\text{ADT}(T) > K$ if the necessary bounds do not hold in terms of K . Thanks to that, $\text{FindTriangle}(K)$ runs in $\mathcal{O}(K \log^2 n)$ time. If the procedure $\text{FindTriangle}(K)$ fails to find a triangle, we can safely report that no solution exists, as this means that $\text{FAST}(T) \geq \text{ADT}(T) > K$. This way, we arrive at the following result.

► **Theorem 10** (Theorem 50 in Appendix B in [28]). *The dynamic K-FAST problem admits a data structure with initialization time $\mathcal{O}(n^2)$, worst-case update time $\mathcal{O}(\log^2 n)$ and worst-case query time $\mathcal{O}(3^K K \log^2 n)$.*

4 Dynamic K-FEEDBACK VERTEX SET IN TOURNAMENTS in the promise model

In Appendix C in [28] we present the data structure for the dynamic K-FVST problem in the promise model. The main idea is the same as for the dynamic K-FAST problem in the promise model. We want to be able to quickly find a triangle in tournament T and perform a standard static branching algorithm for K-FVST, presented in Algorithm 2.

■ **Algorithm 2** Pseudocode for `FindFVST(K)`.

Algorithm : `FindFVST(K)`
Output : Verify if $FVST(T) \leq K$

```

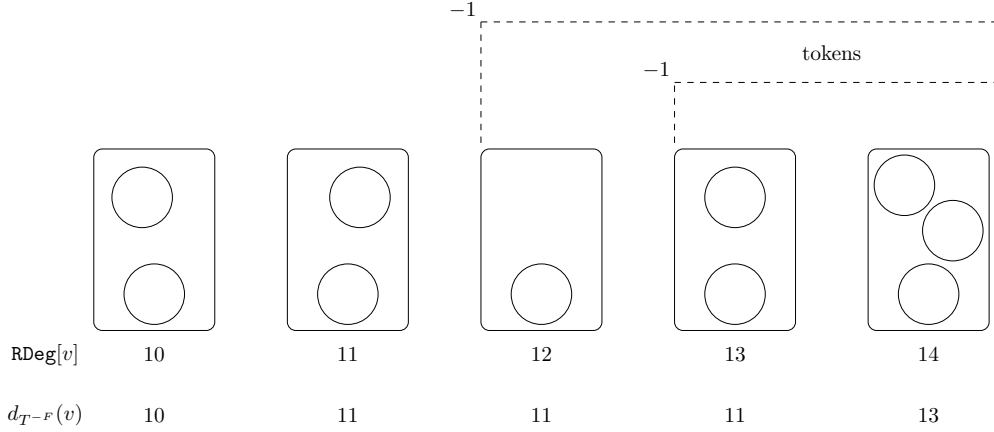
1 if  $T$  is acyclic then
2   | return TRUE ;
3 if  $K = 0$  then
4   | return FALSE ;
5  $uvw \leftarrow \text{FindTriangle}()$ ;
6 for  $x \in \{u, v, w\}$  do
7   | Remove( $x$ ) ;
8   | if FindFVST(K - 1) then
9     | Restore( $x$ );
10  | return TRUE ;
11 Restore( $x$ );
12 return FALSE ;
```

The branching algorithm (Algorithm 2) finds a triangle in the tournament, and then branches recursively with each of the triangle vertices removed. The correctness is again due to the fact that a tournament is acyclic if and only if it has no triangles. To implement this algorithm, we not only need a method to find a triangle in the maintained tournament, but we also need to support vertex removals and restorations. These are significantly more complex than edge reversals, as they change the indegree of up to $(n - 1)$ vertices. This loss of locality poses a number of problems, including maintaining indegree buckets, maintaining the set of empty indegree buckets, or even keeping track of the acyclicity of the tournament. Observe also, that our parameter is now $FVST(T)$, which behaves in a different way than $FAST(T)$. For instance, our triangle detection data structures rely on the fact, that the number of back arcs in a tournament T is bounded in terms of $FAST(T)$. This, unfortunately, stops to be the case: the number of back arcs can be actually unbounded in terms of $FVST(T)$. In the following sections, we describe how our data structure deals with all these issues.

4.1 Vertex Removals and Restorations

We first deal with the problem of removing and restoring vertices. In Appendix C.2 in [28] we introduce a new data structure called $\mathbb{DREM}[n, k]$, which essentially extends the data structure $\mathbb{DTP}[n]$ by the possibility of removing and restoring up to k vertices. This data structure is covered in detail by Lemma 59 in Appendix C.2 in [28], below we only present the main ideas.

The data structure $\mathbb{DREM}[n, k]$ maintains all the information about the tournament T that was maintained by $\mathbb{DTP}[n]$, with the exception of the set `Back` of back arcs, whose maintenance now becomes prohibitively expensive. In addition to that, $\mathbb{DREM}[n, k]$ maintains



■ **Figure 3** Implicit representation of tournament T^{-F} .

the set of removed vertices F and an implicit representation of T^{-F} . Here, T^{-F} stands for the tournament T with the set F of vertices removed, i.e, $T^{-F} = T[V(T) \setminus F]$. As mentioned before, we need to be able to quickly decrease/increase indegrees of many vertices, because removal/restoration of a vertex can pessimistically affect all other vertices. In order to do so, the $\mathbb{D}\text{REM}[n, k]$ data structure stores a set of token positions. One token corresponds to one removed vertex. Each time a vertex is removed, some token is placed at some position d . The vertices are again partitioned into buckets. A token at position d decreases by one the indegrees of vertices in all buckets whose indices are at least d . An illustration of the intended role of tokens can be found in Figure 3.

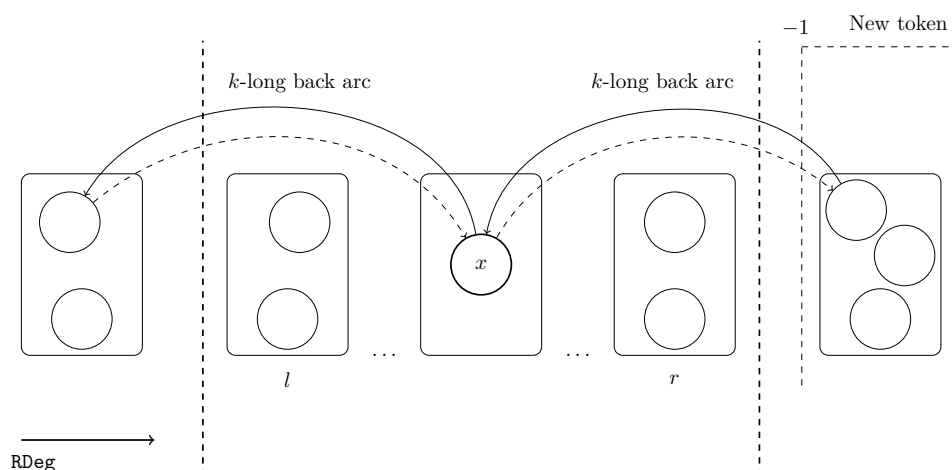
Ideally, to maintain T^{-F} , we would like to partition the vertices into buckets according to indegrees in T^{-F} . Due to the presence of tokens this is not feasible. Instead, we introduce a reduced indegree $\text{RDeg}[v]$ of a vertex v , which relates indegree $d_{T^{-F}}(v)$ of a vertex v in T^{-F} with token positions. To be more precise, let us denote by $\text{CTok}(d)$ the number of tokens at positions smaller or equal than d . The reduced indegree of a vertex v satisfies

► **Invariant 1.** $\text{RDeg}[v] - \text{CTok}(\text{RDeg}[v]) = d_{T^{-F}}(v)$.

This is an invariant of the data structure which guarantees, that the tokens reflect their intended role. Rather than according to indegrees in T^{-F} , the vertices of T^{-F} are then partitioned according to their reduced indegrees, and we refer to this partition as *reduced indegree buckets*. The reduced indegrees and the partition into reduced indegree buckets are maintained by $\mathbb{D}\text{REM}[n, k]$ in order to implicitly maintain tournament T^{-F} . This is illustrated in Figure 3. Similarly as before, also the set of empty reduced indegree buckets is maintained.

Apart from T , F , tokens, and the implicit representation of tournament T^{-F} , the $\mathbb{D}\text{REM}[n, k]$ data structure stores all k -long back arcs in T . For an arc uv we define its length with respect to tournament T as $l_T(uv) = |d_T(u) - d_T(v)|$. An arc is k -long if its length is at least k , otherwise the arc is k -short. In the $\mathbb{D}\text{REM}[n, k]$ data structure, every vertex stores a set $\text{LONG}(k, v)$ of k -long back arcs (with regard to T) adjacent to it.

The vertex removal procedure is illustrated in Figure 4. It is based on a method $\text{NewRd}(u)$, which given any vertex u , recomputes its reduced degree to restore Invariant 1. Given the set of removed vertices F , a list of token positions, and indegrees in T , one can compute $d_{T^{-F}}(u)$ and the value $\text{RDeg}[u]$ satisfying Invariant 1 in $O(|F|) = O(k)$ time. Due to space constraints, the details are given in the proof of Lemma 59 in Appendix C.2 in [28].



■ **Figure 4** Vertex removal/restoration procedure.

In order to remove (or restore) a vertex v , we need to fix the reduced degrees of all affected vertices in order to satisfy Invariant 1. To accomplish that, we first iterate over vertices u having their reduced indegrees inside a small interval $[l, r]$ around the reduced indegree of v , and we fix all these vertices using $\text{NewRd}(u)$. The size of the interval is bounded by $r - l \in O(k)$, so this takes time proportional to k^2 multiplied by maximum size of a reduced indegree bucket. Observe that $\text{RDeg}[v]$, $d_{T-F}(v)$ and $d_T(v)$ differ from each other by at most $O(k)$. Thus, the interval $[l, r]$ can be chosen in a way that all arcs between v and vertices with reduced degrees outside of $[l, r]$ are k -long arcs with regard to T (for an illustration see Figure 4). Thus, for all vertices w such that $\text{RDeg}[w] < l$, whose indegree changes as a result of v 's removal (or restoration), it holds that vw is a k -long back arc with regard to T . By iterating through $\text{LONG}(k, v)$ we can detect such vertices and fix their reduced indegrees. On the other hand, for all vertices w such that $\text{RDeg}[w] > r$, their indegree does decrease by one after v 's removal (or increase after v 's restoration), unless wv is a k -long back arc with regard to T . Thus, placing a token at position $r + 1$ fixes the invariant for all vertices with $\text{RDeg}[w] > r$, with the exception of the endpoints of k -long back arcs whose other endpoint is v . We can detect these vertices by iterating through $\text{LONG}(k, v)$ and fix their reduced indegrees. To sum up, the runtime of removal/restoration of a vertex v depends on $|\text{LONG}(k, v)|$ and on k^2 multiplied by the maximum size of a reduced indegree bucket. In Subsection 4.3 we show how to bound these parameters in terms of $\text{FVST}(T)$ to guarantee efficient running times in the promise model.

4.2 Detecting Triangles

Next, using the $\text{DREM}[n, k]$ data structure, we need to implement triangle detection in T^{-F} . To achieve that, we follow ideas from Section 2. Recall, that in order to find a triangle in a tournament T , we needed a method to find v_{\min} - a vertex of minimum indegree in $T^{-\text{pref}}$ and we needed a method for listing in-neighbours. We showed that this can be done in time proportional to $|\text{Empty}|$ and $|\text{Back}|$. In fact, we did not need to iterate through all back arcs, but just the ones incident to a vertex whose in-neighbours we seek. We now want to run these procedures on T^{-F} instead of T . We can access T^{-F} via the reduced indegree buckets stored by $\text{DREM}[n, k]$ data structure. In Appendix C.4 in [28] we carefully show, that the reduced degree buckets are functionally very close to the indegree buckets of T^{-F} , and we

can essentially use them instead to find v_{\min} and list in-neighbours with regard to T^{-F} . Still, we need reasonable bounds in terms of $\text{FVST}(T)$ on $|\text{Empty}^{-F}|$ and $|\text{Back}_v^{-F}|$, which are the number of empty indegree buckets in T^{-F} and the number of back arcs incident to a vertex v in T^{-F} . We show how to bound these in the next subsections.

4.3 Bounds

We first observe that the maximum size of an indegree bucket in a tournament T satisfies

► **Fact 11** (Lemma 55 in Appendix C.4 in [28]). $\max_{\text{Db}}(T) \leq 2\text{FVST}(T) + 1$.

To see why this holds, consider an indegree bucket $\text{Db}_T[d]$ of maximum size. Let $S = \text{Db}_T[d] \setminus F_V$, where F_V is the feedback vertex set of minimum size. After removal of F_V , each vertex in S lands in a separate indegree bucket. This implies that one of the vertices in S decreases its indegree by at least $|S| - 1$ after removing F_V . Each vertex removal decreases indegree of any other vertex by at most one. Thus, $|S| - 1 \leq |F_V|$ and $|\text{Db}_T[d]| \leq |S| + |F_V| \leq 2|F_V| + 1$.

Let now $\max_{\text{RDb}}(T, F)$ denote the maximum size of a reduced indegree bucket. We can relate $\max_{\text{RDb}}(T, F)$ to $\max_{\text{Db}}(T)$ as follows.

► **Fact 12** (Observation 61 in Appendix C.4 in [28]). $\max_{\text{RDb}}(T, F) \leq \max_{\text{Db}}(T)(|F| + 1)$

The reason why Fact 12 holds is as follows. As Figure 3 suggests, any reduced degree bucket is entirely contained in some indegree bucket with regard to T^{-F} . The vertices of one indegree bucket in T^{-F} are contained in at most $|F| + 1$ indegree buckets in T , as any vertex can decrease its indegree by at most $|F|$ after removing F .

Let $\text{LONG}(k, X)$ for $X \subseteq V(T)$ denote a set of k -long back arcs (with regard to T) between the vertices in X . Slightly more elaborate arguments of similar nature as above allow us to bound $|\text{Empty}^{-F}|$ as follows.

► **Fact 13** (Lemma 67 and Corollary 68 in Appendix C.4 in [28]).

$$|\text{Empty}^{-F}| \leq \text{FVST}(T^{-F}) \cdot (2(k + |F|) + |\text{LONG}(k, V(T^{-F}))|) + 2\text{FVST}(T^{-F}) + 5) + 4|F|$$

Given the above bound, our goal is now to bound $|\text{LONG}(k, V(T^{-F}))|$. As Section 4.1 and Section 4.2 suggest, we also need to bound $|\text{Back}_v^{-F}|$ and $|\text{LONG}(k, v)|$ for all $v \in V(T^{-F})$.

Due to Fact 12 and a close relation between reduced degree buckets and indegree buckets of T^{-F} , there is a bounded number of $(k + |F|)$ -short arcs in Back_v^{-F} . Since removing a vertex can decrease the indegree of any other vertex by at most one, the arcs that are $(k + |F|)$ -long in T^{-F} are k -long in T . Thus, bounding $|\text{Back}_v^{-F}|$ for $v \in T^{-F}$ boils down to bounding $|\text{LONG}(k, v)|$ for $v \in V(T^{-F})$.

To sum up, it suffices that we bound $|\text{LONG}(k, V(T^{-F}))|$ and $|\text{LONG}(k, v)|$ for all $v \in V(T^{-F})$. We deal with this in the subsequent section.

4.4 Kernelization Technique

In order to bound $|\text{LONG}(k, V(T^{-F}))|$ and $|\text{LONG}(k, v)|$ for $v \in V(T^{-F})$, we actually reduce the number of k -long back arcs by removing some vertices from T . To achieve this, we define a k -long graph G_{LONG} of the tournament T , which is an undirected graph, where vertices are connected via an edge in G_{LONG} if they are connected via a k -long back arc in the tournament T . We also define the k -heavy set $\text{Heavy}_k(T)$ of the tournament T as the set of vertices of degree higher than k in G_{LONG} . If $F_V \subseteq V(T)$ is a feedback vertex set in T of size at

most k , then F_V is necessarily a vertex cover in G_{LONG} : there is no other way to get rid of a k -long back arc from T than to remove its endpoint (see Lemma 64 in [28] for a more formal argument). Due to standard vertex cover kernelization arguments, $\text{Heavy}_k(T) \subseteq F_V$ for any feedback vertex set F_V of T with $|F_V| \leq k$. Moreover, when we remove $\text{Heavy}_k(T)$ from G_{LONG} , at most $k|F_V|$ edges remain in G_{LONG} . This simple but crucial observations are covered by Lemma 64 in [28]. So the idea is to keep the set $\text{Heavy}_k(T)$ removed from the tournament, in order to keep the number of k -long back arcs connecting the remaining vertices small. Observe also, that if $\text{Heavy}_k(T) \subseteq F$, where F is the set of removed vertices, any vertex that is left in T^{-F} has at most k adjacent k -long back arcs, what gives the desired bound on $|\text{LONG}(k, v)|$ for $v \in V(T^{-F})$. We want to emphasize, that the property of an arc being k -long is considered here with regard to tournament T and it does not depend on F , so we can maintain G_{LONG} in the form of adjacency lists efficiently.

To implement the above idea, in Appendix C.3 we introduce a wrapper data structure around the $\text{DREM}[n, k]$ called $\text{DREMP}[n, k]$. It allows only one kind of updates, arc reversals, and keeps the invariant that the k -heavy set of the maintained tournament is removed. This not only allows us to efficiently implement the methods for finding triangles, but also ensures fast running times of $\text{DREM}[n, k]$ operations in the promise model. The wrapper is defined in Lemma 66 in [28].

4.5 The Final Data Structure

In order to implement Algorithm 2, we use the $\text{DREMP}[n, k]$ data structure for $k = g(K)$, where K is the problem parameter. The $\text{DREMP}[n, k]$ data structure keeps $\text{Heavy}_k(T)$ removed from the tournament, i.e., $\text{Heavy}_k(T) \subseteq F$ at all times, where F is the set of currently removed vertices. This provides us with the bounds on $|\text{LONG}(k, v)|$ for $v \in T^{-F}$ and $|\text{LONG}(k, V(T^{-F}))|$, which we need to efficiently find triangles. When we branch on the vertices of the found triangle, we use the methods of the $\text{DREM}[n, k]$ data structure (internally maintained by $\text{DREMP}[n, k]$), in order to temporarily remove these vertices from the tournament. This approach leads to the following theorem.

► **Theorem 14** (Theorem 51 in Appendix C in [28]). *The dynamic K -FVST problem admits a data structure with initialization time $\mathcal{O}(n^2)$, worst-case update time $\mathcal{O}(g(K)^5)$ and worst-case query time $\mathcal{O}(3^K K^3 g(K))$ under the promise that there is a computable function g , such that tournament T always has a feedback vertex set of size at most $g(K)$.*

References

- 1 Josh Alman, Matthias Mních, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. *ACM Trans. Algorithms*, 16(4), July 2020. doi:10.1145/3395037.
- 2 Noga Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics*, 20(1):137–142, 2006. doi:10.1137/050623905.
- 3 Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikolettseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2009. doi:10.1007/978-3-642-02927-1_6.
- 4 Jeffrey Banks. Sophisticated voting outcomes and agenda control. Working Papers 524, California Institute of Technology, Division of the Humanities and Social Sciences, 1984. URL: <https://EconPapers.repec.org/RePEc:clt:sswopa:524>.

- 5 Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski, and Anna Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '21*, pages 796–809, USA, 2021. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611976465.50.
- 6 Don Coppersmith, Lisa K. Fleischer, and Atri Rurda. Ordering by weighted number of wins gives a good ranking for weighted tournaments. *ACM Trans. Algorithms*, 6(3), July 2010. doi:10.1145/1798596.1798608.
- 7 Zdenek Dvořák, Martin Kupec, and Vojtech Tůma. A dynamic data structure for MSO properties in graphs with bounded tree-depth. In *22th Annual European Symposium on Algorithms, ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2014. doi:10.1007/978-3-662-44777-2_28.
- 8 Zdenek Dvořák and Vojtech Tůma. A dynamic data structure for counting subgraphs in sparse graphs. In *13th International Symposium on Algorithms and Data Structures, WADS 2013*, volume 8037 of *Lecture Notes in Computer Science*, pages 304–315. Springer, 2013. doi:10.1007/978-3-642-40104-6_27.
- 9 Pál Erdős and John W. Moon. On sets of consistent arcs in a tournament. *Canadian Mathematical Bulletin*, 8:269–271, 1965. URL: <https://api.semanticscholar.org/CorpusID:19010097>.
- 10 Uriel Feige. Faster fast(feedback arc set in tournaments), 2009. arXiv:0911.5094.
- 11 W Fernandez de la Vega. On the maximum cardinality of a consistent set of arcs in a random tournament. *Journal of Combinatorial Theory, Series B*, 35(3):328–332, 1983. doi:10.1016/0095-8956(83)90060-6.
- 12 Fedor V. Fomin and Michał Pilipczuk. On width measures and topological problems on semi-complete digraphs. *Journal of Combinatorial Theory, Series B*, 138:78–165, 2019. doi:10.1016/j.jctb.2019.01.006.
- 13 Serge Gaspers and Matthias Mnich. Feedback vertex sets in tournaments. In Mark de Berg and Ulrich Meyer, editors, *Algorithms – ESA 2010*, pages 267–277, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-15775-2_23.
- 14 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, July 2001. doi:10.1145/502090.502095.
- 15 Yoichi Iwata and Keigo Oka. Fast dynamic graph algorithms for parameterized problems. In *14th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2014*, volume 8503 of *Lecture Notes in Computer Science*, pages 241–252. Springer, 2014. doi:10.1007/978-3-319-08404-6_21.
- 16 Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing, STOC '07*, pages 95–103, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1250790.1250806.
- 17 Tuukka Korhonen, Konrad Majewski, Wojciech Nadara, Michał Pilipczuk, and Marek Sokołowski. Dynamic treewidth, 2023. arXiv:2304.01744.
- 18 R. Krithika, Abhishek Sahu, Saket Saurabh, and Meirav Zehavi. The parameterized complexity of packing arc-disjoint cycles in tournaments, 2018. arXiv:1802.07090.
- 19 Mithilesh Kumar and Daniel Lokshantov. Faster Exact and Parameterized Algorithm for Feedback Vertex Set in Tournaments. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2016.49.
- 20 Daniel Lokshantov, Pranabendu Misra, Joydeep Mukherjee, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. 2-approximating feedback vertex set in tournaments. *ACM Trans. Algorithms*, 17(2), April 2021. doi:10.1145/3446969.

- 21 Konrad Majewski, Michał Pilipczuk, and Marek Sokołowski. Maintaining CMSO2 Properties on Dynamic Structures with Bounded Feedback Vertex Number. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:13, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2023.46.
- 22 Matthias Mnich, Virginia Vassilevska Williams, and László A. Végh. A $7/3$ -Approximation for Feedback Vertex Sets in Tournaments. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 67:1–67:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2016.67.
- 23 Jędrzej Olkowski, Michał Pilipczuk, Mateusz Rychlicki, Karol Węgrzycki, and Anna Zych-Pawlewicz. Dynamic Data Structures for Parameterized String Problems. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:22, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2023.50.
- 24 Venkatesh Raman and Saket Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006. Parameterized and Exact Computation. doi:10.1016/j.tcs.2005.10.010.
- 25 K.B. Reid and E.T. Parker. Disproof of a conjecture of Erdős and Moser on tournaments. *Journal of Combinatorial Theory*, 9(3):225–238, 1970. doi:10.1016/S0021-9800(70)80061-8.
- 26 E. Speckenmeyer. On feedback problems in digraphs. In *Proceedings of the Fifteenth International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '89, pages 218–231, Berlin, Heidelberg, 1990. Springer-Verlag.
- 27 Anke van Zuylen, Rajneesh Hegde, Kamal Jain, and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 405–414, USA, 2007. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283426>.
- 28 Anna Zych-Pawlewicz and Marek Żochowski. Dynamic parameterized feedback problems in tournaments, 2024. doi:10.48550/arXiv.2404.12907.