Tournament Fixing Parameterized by Feedback Vertex Set Number Is FPT

Meirav Zehavi*

Department of Computer Science, Ben-Gurion University of the Negev, Beersheba, Israel
meiravze@bgu.ac.il

Abstract

A *knockout (or single-elimination) tournament* is a format of a competition that is very popular in practice (particularly in sports, elections and decision making), and which has been extensively and intensively studied from a theoretical point of view for more than a decade. Particular attention has been devoted to the TOURNAMENT FIXING problem, where, roughly speaking, the objective is to determine whether we can conduct the knockout tournament in a way that makes our favorite player win. Here, part of the input is a tournament graph D that encodes the winner of each possible match. A sequence of papers has studied the parameterized complexity of TOURNAMENT FIXING with respect to the *feedback arc set number* ($\mathbf{fas}$) of D. Given that this parameter yielded tractability, it has been asked explicitly and repeatedly whether TOURNAMENT FIXING is FPT also with respect to the *feedback vertex set number* ($\mathbf{fvs}$) of D. We answer this question positively. In fact, although $\mathbf{fvs}$ can be *arbitrarily smaller* than $\mathbf{fas}$, we attain the same dependency on the parameter in the time complexity. So, additionally, our work subsumes the best known algorithm for TOURNAMENT FIXING with respect to $\mathbf{fas}$.

Introduction

A *knockout (or single-elimination) tournament* is a format of competition that is very popular in practice, particularly in sports and popular culture, elections and decision making. Notably, it is the most widely-used format of competition in sports (Horen and Riezman 1985; Connolly and Rendleman 2011; Groh et al. 2012). Further, it has been extensively and intensively studied from both practical and theoretical point of views for over three decades, particularly from the perspectives of artificial intelligence, computational social choice, economics and operation research. We refer to (Williams 2016; Suksompong 2021; Tullock 1980; Rosen 1986; Laslier 1997) for a few illustrative examples.

Formally, an execution of a knockout tournament with a set N of n players (where n is a power of 2) is conducted as follows. Consider a complete (unordered) binary tree T with n leaves and a mapping $\varphi : N \rightarrow \mathsf{leaves}(T)$, called a *seeding*. In the first round, every two players mapped to leaves with the same parent compete against each other, and the winner is mapped to the common parent. The leaves are then deleted from the tree, and the next round is conducted similarly. The execution stops when the tree contains a single node, mapped to a player who is declared the winner.

Having a favorite player w in mind, a natural question in this context is—does there exist a way to conduct the competition so that w wins? This question is sensible when we have predictive information about outcomes of matches. Specifically, we assume we have a digraph D that is a tournament (not to be confused with the competition itself, which is also termed a tournament). That is, D is a digraph where either $(u, v) \in A(D)$ or $(v, u) \in A(D)$ for all $u, v \in V(D)$. This encodes predictive information as follows: $V(D) = N$, and for every $u, v \in N$, we predict that in a match between u and v, u would beat v if and only if $(u, v) \in A(D)$. Now, the corresponding problem, termed the TOURNAMENT FIXING problem, is formalized as follows. The input consists of a tournament D, and a node $w \in V(D)$. The objective is to decide whether there exists a seeding of the set of n players, $V(D)$, such that w wins the resulting knockout tournament.

Previous Work. The TOURNAMENT FIXING problem was introduced by Vu et al. (Vu, Altman, and Shoham 2009) in 2009. This work has led to a flurry of research of structural properties of D that guarantee that w wins (see, e.g., (Williams 2010; Stanton and Williams 2011b,a; Aziz et al. 2014; Kim and Williams 2015; Kim, Suksompong, and Williams 2016)). Nowadays, there already exist dozens of papers that consider the TOURNAMENT FIXING problem and its different variants. Here, we only briefly survey some of the results most relevant to our work. For more information, we refer to surveys on TOURNAMENT FIXING such as (Williams 2016; Suksompong 2021). Computationally, TOURNAMENT FIXING is NP-hard (Aziz et al. 2014). On the positive side, it was shown to be solvable in time $2^n \cdot n^{\mathcal{O}(1)}$ and polynomial space (Gupta et al. 2018b), improving upon earlier exponential-time algorithms by (Aziz et al. 2014; Kim and Williams 2015). Since n can be quite large, this running time is prohibitive.

The state-of-the-art described above gave rise to the study of the parameterized complexity of the problem. Specifically, Aziz et al. (Aziz et al. 2014) considered the *feedback*

*Supported by the European Research Council (ERC) grant titled PARAPATH.
Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

arc set number of D—that is, the minimum number of arcs to remove (or, equivalently, reverse) in D to turn it into a DAG (directed acyclic graph)—as the parameter k. This parameter has a natural interpretation as follows. In real world scenarios, there often exists an "approximate" ranking of players' strengths (e.g., Wimbledon), so that a player of a certain rank is expected to beat players of a lower rank. In particular, the number of matches where we predict (before the competition begins) that a player of a certain rank will beat a player of a higher rank, which upper bounds k, can be significantly smaller than n. Aziz et al. (Aziz et al. 2014) showed that TOURNAMENT FIXING is XP, which means solvability in time $n^{f(k)}$ for some function f of k. Specifically, they attained a running time of $n^{\mathcal{O}(k)}$. Assuming that k is not fixed (i.e., independent of n), the running time is clearly impractical. The question of whether TOURNAMENT FIXING is *fixed-parameter tractable (FPT)* with respect to k—that is, solvable in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some function f of k—was later resolved by Ramanujan and Szeider (Ramanujan and Szeider 2017). Specifically, based on the use of Integer Linear Programming (ILP), they attained a running time of $k^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$. Afterwards, Gupta et al. (Gupta et al. 2018a) sped-up the aforementioned algorithm based on the use of a greedy procedure instead of ILP, and attained the currently best running time of $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. Additionally, Gupta et al. (Gupta et al. 2019) proved that the problem admits a polynomial kernel with respect to k.

Feedback Vertex Set and Our Contribution. Given the successes above for the feedback arc set number, it has been asked explicitly and repeatedly whether TOURNAMENT FIXING is FPT also with respect to the *feedback vertex set number* of D (Ramanujan and Szeider 2017; Gupta et al. 2018b, 2019; Roy 2020; Suksompong 2021), that is, the minimum number of nodes to remove from D in order to turn it into a DAG. Observe that the feedback vertex set number of D is always bounded from above by twice its feedback arc set number. More importantly, the feedback vertex set number of D can be *arbitrarily smaller* than its feedback arc set number; for example, when the feedback vertex set number is 1, the feedback arc set number can be as large as $n - 1$! Intuitively, the feedback vertex set number is bounded from above by the number of players that are expected to "surprise" us (beating players ranked higher than them), rather than the total number of matches where we except to have "surprises". For example, in tennis or chess tournaments, the outcome of the matches involving most players can be predicted in advance, and only few players—usually young players that have not yet showed their true abilities—"surprise" us. The same is observed in other competitions, particularly (but not only) sports competitions. Then, the feedback vertex set number is small, while the feedback arc set number can be very large. We note that when uncertainties regarding winners of some specific matches arise, one can model it using bidirected arcs (our result still holds in this setting).

Our contribution is the positive answer to the question of whether TOURNAMENT FIXING is FPT with respect to feedback vertex set number. We attain the following runtime:

Theorem 1. *The* TOURNAMENT FIXING *problem is solvable in time* $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ *where* n *and* k *are the number of nodes and the feedback vertex set number of the input tournament* D*, respectively.*

So, additionally, our work subsumes the best known algorithm for TOURNAMENT FIXING with respect to the feedback arc set number as the parameter. Following a certain guessing step that is similar to that done in both (Ramanujan and Szeider 2017; Gupta et al. 2018a), our approach diverges significantly: specifically, we make (quite technically involved) use of dynamic programming. We remark that, by backtracking the computation, our algorithm can also return the solution seeding, if one exists.

Outline. First, we present basic definitions and notations. Then, we present a brief description of the ideas of our proof. Afterwards, we present some of the technical details of the proof. Lastly, we conclude the manuscript with some directions for further research.

Preliminaries

Graph Notation. Given a digraph D, let $V(D)$ and $A(D)$ denote its vertex set and arc set, respectively. Whenever we consider a rooted tree, we treat it as a directed graph where each arc is directed from the parent to the child. Given $U \subseteq V(D)$, let $D[U]$ denote the subgraph of D induced by U, and let $D - U$ denote the subgraph of D induced by $V(D) \setminus U$. Additionally, let $N_D(U)$ denote the set of nodes in $V(D) \setminus U$ that have (ingoing or outgoing) neighbors in U. Given a rooted tree T and a node $u \in V(T)$, let $\mathsf{parent}_T(u)$ denote the parent of u in T (defined to be nil if u is the root of T), let $\mathsf{children}_T(u)$ denote the set of children of u in T, and let T_u denote the subtree of T rooted at u. Given a descendant v of u in T, let $P_T(u, v)$ denote the unique path from u to v in T, and, given also $x \in \{0, 1, \ldots, |A(P_T(u, v))|\}$, let $P_T^x(u, v)$ denote the subpath of $P_T(u, v)$ that consists of the x nodes closest to u excluding u. The *length* of a path is its number of arcs. Two digraphs D and D' are *isomorphic* if there exists a bijective function $f : V(D) \to V(D')$, called an *isomorphism*, such that for every pair of nodes $u, v \in V(D)$, $(u, v) \in A(D)$ if and only if $(f(u), f(v)) \in A(D')$.

Given a tree T and two nodes $u, v \in V(T)$, the *lowest common ancestor* of u and v in T is denoted by $\mathsf{lca}_T(u, v)$.

Definition 1 (LCA-closure)**.** *For a rooted tree* T *and a subset* $X \subseteq V(T)$*, the* least common ancestor-closure (LCA-closure)*,* $\mathsf{LCA}_T(X)$*, is obtained by the following process. Initially, set* $X' = X$*. Then, as long as there exist nodes* $u, v \in X'$ *such that* $\mathsf{lca}_T(u, v) \notin X'$*, add* $\mathsf{lca}_T(u, v)$ *to* X'*. When the process terminates, output* $\mathsf{LCA}_T(X) = X'$*.*

Proposition 1 (Folklore, see, e.g. (Cygan et al. 2015))**.** *For a rooted tree* T *and a subset* $X \subseteq V(T)$*,* (i) $|\mathsf{LCA}_T(X)| \leq 2|X|$*, and* (ii) *for every connected component* C *of* $T - \mathsf{LCA}_T(X)$*,* $|N_T(V(C))| \leq 2$*.*

Lastly, we state the following immediate observation.

Observation 1. *Let* D_1 *be a tournament that is a DAG. Let* D_2 *be a DAG such that* $|V(D_1)| = |V(D_2)|$*. Then,* D_1 *contains a subgraph that is isomorphic to* D_2*.*

Binomial Arborescence. The notion of a binomial arborescence (BA) is defined as follows.

Definition 2 (Binomial Arborescence (BA))**.** *A* binomial arborescence (BA) B *rooted at* v *is defined recursively:*

- *The tree on the single node* v *is a BA rooted at* v*.*
- *Given two node-disjoint BAs of equal size,* B_v *rooted at* v *and* B_u *rooted at* u*, the addition of an arc from* v *to* u *yields a BA rooted at* v*.*

For any n *that is a power of* 2*, the unique (up to isomorphism) BA on* n *nodes is denoted by* B_n*.*

The notion of a BA is of particular importance to our work due to the following definition and proposition that relate it to TOURNAMENT FIXING.

Definition 3 ((D, w)-BA)**.** *Let* (D, w) *be an instance of the* TOURNAMENT FIXING *problem. A* (D, w)-BA *is a subtree* S *of* D *rooted at* w *that is isomorphic to* $B_{|V(D)|}$*; a corresponding isomorphism is denoted by* iso_S*. When* S *is clear from context, we write* $\mathsf{iso} = \mathsf{iso}_S$*.*

Proposition 2 ((Williams 2010))**.** *Let* (D, w) *be an instance of the* TOURNAMENT FIXING *problem. Then,* (D, w) *is a yes-instance if and only if there exists a* (D, w)*-BA.*

We will also make use of the following immediate observation regarding BAs.

Observation 2. *Let* $B = B_n$ *for some* n *that is a power of* 2*. Then, for any node* $u \in V(B)$*:* (i) B_u *is a BA,* (ii) $|\mathsf{children}_B(u)| = \log_2 |V(B_u)|$*, and* (iii) *the length of the path in* B *from the root of* B *to* u *is at most* $\log_2 n$*.*

Lastly, we state the following proposition.

Proposition 3 ((Ramanujan and Szeider 2017))**.** *For every* $k, n \in \mathbb{N}$ *where* $k \leq n$*,* $(\log_2 n)^k \leq (4k \log_2 k)^k + n^2$*.*

The Ideas Behind the Proof

For some minimum-sized feedback vertex set F of D, the first step of our algorithm (described in the Guesses section) is to "guess" some information about the "positions" of the nodes in $F^\star = F \cup \{w\}$—or, more precisely, in $\mathsf{LCA}_S(F^\star)$—in a (D, w)-BA S (if one exists). Here, we aim to "guess" a minimum amount of such information that will enable us to later reconstruct either S itself or a "similar" (D, w)-BA. Concretely, we will iterate over sufficiently many "objects" that can describe this information, with the aim that, if there exists a (D, w)-BA, then at least one of the objects will describe the information corresponding to it.

Afterwards, in the Dynamic Programming section, we focus on one iteration, corresponding to one such object. Here, we only aim to solve the question of the existence of a (D, w)-BA that "complies" with this object, rather than the existence of any (D, w)-BA. To this end, we use dynamic programming where we iterate over the nodes in $V(D) \setminus F^\star$ according to σ, from strong to weak players. At each iteration, we aim to map the current node v from $V(D) \setminus F^\star$, extending partial solutions that already map all nodes in $V(D) \setminus F^\star$ that precede v in σ. When we aim to map v, it is important to distinguish between two cases (and, later, subcases of these cases), depending on whether or not we map v to a neighbor (in B) of a node to whom a node in $F^\star$ is mapped.

When we consider the first case, we require extra care due to the arbitrary adjacency relations (in D) involving nodes in $F^\star$. In one subcase, we map v to a node in B that is both a neighbor and an ancestor (in B) of nodes to whom nodes in $F^\star$ are mapped. To handle this subcase, we suppose to have efficiently guessed information (as part of in the first step) that can, at this step, be used—here, the argument for efficiently is based on the observation that there are only $\mathcal{O}(k)$ such nodes v. In the other subcase, where we map v to a child of a node in B to whom a node in $F^\star$ is mapped, and which has no descendants in B to whom nodes in $F^\star$ are mapped, is, in a sense, "easier", and, indeed, we directly deal with it in the recursive formula. In particular, we know that all nodes yet to be mapped to B are beaten by v (due to our use of σ), and suppose that no node has so far been mapped to a descendant of the node to which we aim to map v.

When we consider the second case, where we map v to a node (in B) that has no neighbor to whom a node in $F^\star$ is mapped, we also directly use the recursive formula. Here, we further distinguish between two subcases, depending on whether or not v is mapped to an ancestor of nodes to whom nodes in $F^\star$ are mapped. In the former subcase, v is necessarily mapped to an internal node, say, x, of a path between two nodes whose information is supposed to be encoded in the current guess (to whom we do not map nodes in $F^\star$, but which have neighbors to whom we map nodes in $F^\star$). We suppose that all nodes above x on this path have already been "filled", and all nodes below it have not yet been "filled", which makes this subcase easy to deal with— the node mapped to the parent of x necessarily beats v, while the node to be mapped to the child of x will necessarily be beaten by v. Now, consider the latter subcase. Here, notice that the node in B to whom v is to be mapped, say, x, has ancestors to whom nodes in $F^\star$ are mapped as well as, possibly, ancestors that are on paths between nodes to whom nodes in $F^\star$ are mapped. The scenario where the closest such ancestor to x belongs to the first type is easy—we suppose to have already mapped a node to the parent of x that beats v, and that we have not yet mapped nodes to descendants of x. The other scenario is more difficult, and to handle it, we make use of relatively technical definitions of so-called *capacity* and *slots*. Some intuition about this is given just after the definition of slots (Definition 15).

Guesses

Let (D, w) be an instance of the TOURNAMENT FIXING problem, and denote $n = |V(D)|$ and $B = B_n$.

We begin by using the following proposition to compute a minimum-sized feedback vertex set F of D in time $3^k \cdot n^{\mathcal{O}(1)}$, where k is the feedback vertex set number of D:

Proposition 4 ((Cygan et al. 2015))**.** *There exists an algorithm that, given a tournament graph* T*, computes a minimum-sized feedback vertex set of* T *in time* $3^t \cdot |V(T)|^{\mathcal{O}(1)}$ *where* t *is the feedback vertex set number of* T*.*

Let $F^\star = F \cup \{w\}$. Since $D - F^\star$ is a directed acyclic tournament, it admits a unique topological order, which we

denote by σ. Thus, for every arc $(u, v) \in A(D)$ with $u, v \notin F^\star$, u precedes v in σ.

Through the rest of this section we follow the first step described in the The Ideas Behind the Proof section, our objective will be to "guess" a minimum amount of information about the "positions" of the nodes in $F^\star$.

First, we would like to "know" the relative ordering of the positions of the nodes in $F^\star$. We encode such information using the definition of a structure, given below. Here, some nodes are identified as the nodes in $F^\star$, while other are "placeholders" to describe the positions of $\mathsf{LCA}_S(F^\star) \setminus F^\star$ (which is unknown, since S is unknown) as well as some of the other nodes adjacent to the nodes in $F^\star$ in S.

Definition 4 (Structure)**.** *A* structure *is a tree T rooted at w such that $V(T) \cap V(D) = F^\star$, and both of the following conditions hold:*

1. *For every $(u, v) \in A(T)$ such that $\{u, v\} \subseteq F^\star$: $(u, v) \in A(D)$.*
2. *For every $u \in V(T) \setminus V(D)$, at least one of the following conditions holds:*
   (a) $\mathsf{children}_T(u) \cap F \neq \emptyset$.
   (b) $\mathsf{parent}_T(u) \in F^\star$ *and* $\mathsf{children}_T(u) \neq \emptyset$.
   (c) $|\mathsf{children}_T(u)| \geq 2$.

The way in which a structure encodes the aforementioned relative ordering will become clear soon (from Definitions 7 and 8 ahead). Here, we note that the sensibility of Condition 1 is due to the fact that we will ensure that any node adjacent to a node $u \in F^\star$ in T will be mapped to a node adjacent to u in the (D, w)-BA. The other condition is meant to ensure that we do not encode information for "too many" placeholders (required for the efficiency of Lemma 1): we have placeholders for parents of nodes in $F^\star$, their children that have descendants in $F^\star$, and nodes corresponding to the LCA-closure; so, in total, we have $\mathcal{O}(k)$ placeholders.

Next, we enrich the notion of a structure so that it will also encode the following information: for each of its arcs (u, v), we would like to know the length of the path from the node to which u is mapped and the node to which v is mapped in our hypothetical (D, w)-BA, and for each of its nodes u, we would like to know the size of the subtree of the node to which u is mapped in that hypothetical (D, w)-BA. We require knowledge of these measure to know how many nodes to "fill" in these paths and subtrees.

Definition 5 (Length Function)**.** *Given a structure T, a function* $\mathsf{len} : A(T) \to \{1, 2, \ldots, \log_2 n\}$ *is a* length function *if for every arc $(u, v) \in A(T)$ such that $\{u, v\} \cap F^\star \neq \emptyset$,* $\mathsf{len}(u, v) = 0$.

In particular, the condition above is meant to ensure that we have placeholders for all the neighbors (in B) of the nodes in $F^\star$.

Definition 6 (Size Function)**.** *Given a structure T, a function* $\mathsf{siz} : V(T) \to \{2^0, 2^1, \ldots, 2^{\log_2 n}\}$ *is a* size function.

We refer to a triple $(T, \mathsf{len}, \mathsf{siz})$ that consists of a structure, a length function and a size function, respectively, as an *annotated structure*. Next, we define the notion of the realizability of an annotated structure. Intuitively, here we map the nodes of the structure to a BA B in a manner that "complies" with the information encoded by the annotated structure.

Definition 7 (Realizability)**.** *Let $(T, \mathsf{len}, \mathsf{siz})$ be an annotated structure. An injective function* $\mathsf{rea} : V(T) \to V(B)$ *is a* realizability witness *for $(T, \mathsf{len}, \mathsf{siz})$ if all of the following conditions hold:*

1. *For any two nodes $u, v \in V(T)$, $\mathsf{rea}(\mathsf{lca}_T(u, v)) = \mathsf{lca}_B(\mathsf{rea}(u), \mathsf{rea}(v))$. In particular, $\mathsf{LCA}_B(\{\mathsf{rea}(u) : u \in V(T)\}) = \{\mathsf{rea}(u) : u \in V(T)\}$.*
2. *For any arc $e = (u, v) \in A(T)$, the length of the path from $\mathsf{rea}(u)$ to $\mathsf{rea}(v)$ in B is $\mathsf{len}(e)$.*
3. *For any node $u \in V(T)$, the size of the subtree of $\mathsf{rea}(u)$ in B is $\mathsf{siz}(u)$.*

If $(T, \mathsf{len}, \mathsf{siz})$ admits a realizability witness, then it is said to be realizable.

The first condition ensures that we respect ancestory relations as encoded by T, while the other two conditions ensure that we respect the length and size functions.

Finally, we are ready to define the notion of "compliance" of an annotated structure and a (D, w)-BA. Intuitively, if an annotated structure complies with some (D, w)-BA, then it correctly encodes the information about that (D, w)-BA.

Definition 8 (Compliance)**.** *An annotated structure $(T, \mathsf{len}, \mathsf{siz})$* complies *with a (D, w)-BA S if $(T, \mathsf{len}, \mathsf{siz})$ has a realizability witness* $\mathsf{rea} : V(T) \to V(B)$ *such that: for every $u \in F^\star$,* $\mathsf{iso}(u) = \mathsf{rea}(u)$.

The following lemma is very similar to Lemma 4 in (Gupta et al. 2018a) (yet our notion of realizability is slightly different) as well as to lemmas given in (Ramanujan and Szeider 2017; Gupta et al. 2019). Accordingly, the proof follows very similar lines, and therefore we omit it. Still, we briefly note that, for this proof, simply generate all trees on $\mathcal{O}(k)$ nodes and all length and size functions; using Proposition 3, the number of possibilities can be bounded by $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. Then, for each possibility, check realizability using a straightforward dynamic programming computation.

Lemma 1. *There exists a $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$-time algorithm that outputs a collection $\mathcal{C}$ of $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ realizable annotated structures such that, if there exists a (D, w)-BA, then $\mathcal{C}$ contains an annotated structure that complies with it.*

Dynamic Programming

The Definition of the Table M

Fix a realizable annotated structure $(T, \mathsf{len}, \mathsf{siz})$. Each of our table entries will correspond to a single *index*, defined below.

Definition 9 (Index)**.** *An* index *is a quadruple $I = (i, f, g, h)$ where $i \in \{0, 1, \ldots, n - |F^\star|\}$, $f : V(T) \setminus F^\star \to \{\mathsf{false}, \mathsf{true}\}$, $g : A(T) \to \{0, 1, \ldots, \log_2 n - 1\}$ and $h : F^\star \to \{0, 1, \ldots, \log_2 n\}$, such that all of the following conditions hold:*

1. *For every $u \in V(T) \setminus F^\star$ such that $f(u) = \mathsf{true}$, either $\mathsf{parent}_T(u) \in F^\star$ or both $f(\mathsf{parent}_T(u)) = \mathsf{true}$ and $g((\mathsf{parent}_T(u), u)) = \mathsf{len}(e) - 1$.*
2. *For every $e \in A(T)$, $g(e) \leq \mathsf{len}(e) - 1$.*

3. *For every* $e = (u, v) \in A(T)$ *such that* $g(e) > 0$, *(*$u \notin F^\star$ *and)*[1] $f(u) = \mathsf{true}$.
4. *For every* $u \in F^\star$, $h(u) \leq \log_2 \mathsf{siz}(u) - |\mathsf{children}_T(u)|$.

Intuitively, an entry $M[(i, f, g, h)]$ will store a Boolean indicating the existence of a "partial solution"—being part of the sought isomorphism corresponding to a (D, w)-BA—that, in particular, satisfies the following: f encodes, for $u \in V(T) \setminus F^\star$, whether the partial solution maps a node from D to the node corresponding to u in B (according to some realizability witness); g encodes, for $(u, v) \in A(T)$, to how many internal nodes on the path from the node corresponding to u to the node corresponding to v the partial solution maps nodes from D; and h encodes, for $u \in F^\star$, the number of children of the node corresponding to u in B—excluding those corresponding to children of u in T—to whom the partial solution maps nodes from D. So, keeping Observation 2 in mind, Conditions 2 and 4 simply state that we respect the length and size functions.

The correctness of our algorithm will critically rely on (technically involved) constraints to ensure that some specific nodes in B are "filled before" some other specific nodes in B: Roughly speaking, suppose that we have an ordering on the nodes in $V(D) \setminus F^\star$ (specifically, we will be using the ordering σ defined in the Guesses section) so that earlier nodes in this order should be mapped "first" to the nodes in $V(B)$; then, we want to ensure that if, currently, we are going to map a node from $V(D) \setminus F^\star$ to some specific node in B, then we have already mapped nodes from $V(D) \setminus F^\star$ to some other specific nodes in B. In Definition 9, the constraints that aim to enforce such precedences are Conditions 1 and 3. Specifically, Condition 1 is meant to ensure that if we have mapped a node from D to the node corresponding to some node $u \in V(T) \setminus F^\star$, then we have already mapped nodes from D to the node corresponding to its parent in T as well as to all the nodes on the path between them. Condition 4 is meant to ensure that if we have mapped nodes from D to nodes on the path between nodes corresponding to two adjacent nodes in T, then we have already mapped a node to the parent among them.

To formulate additional precedence constraints, we require Definition 10 and, in particular, Definition 11 that relies on it. Intuitively, the capacity of a node u is the number of descendants of the node corresponding to u in B *(i)* to whom the partial solution may map nodes in D, and *(ii)* that belong to some "particular set" (defined in the The Meaning of the Table M section). More intuition about the necessity of the definition of open and closed nodes in this content appears later, after Definition 15.

Definition 10 (Open Node)**.** *Let* $I = (i, f, g, h)$ *be an index. A node* $u \in V(T) \setminus F^\star$ *is* I-open *if* $f(u) = \mathsf{true}$ *and for every* $e = (u, v) \in A(T)$, $g(e) = \mathsf{len}(e)$*; otherwise,* u *is* I-closed.

Definition 11 (Capacity of a Node)**.** *Let* $I = (i, f, g, h)$ *be an index. The* I-capacity *of a node* $u \in V(T)$*, denoted by* $\mathsf{cap}_I(u)$*, is defined as follows:*

1. *If* $u \in F^\star$*:* $\mathsf{cap}_I(u) = \sum_{s \in S_{T,u,h}} s$*, where* $S_{T,u,h}$ *denotes the set of the* $h(u)$ *largest numbers in* $\{2^0, 2^1, \ldots, 2^{\mathsf{siz}(u)}\} \setminus \{2^{\mathsf{siz}(v)} : v \in \mathsf{children}_T(u)\}$.
2. *If* $u \notin F^\star$ *and it is* I*-open:* $\mathsf{cap}_I(u) = \mathsf{siz}(u) - 1 - \sum_{e=(u,v) \in A(T)} g(e) - \sum_{c \in \mathsf{children}_T(u)} \mathsf{siz}(c)$.
3. *If* $u \notin F^\star$ *and it is* I*-closed, then* $\mathsf{cap}_I(u) = 0$.

Having defined the capacity of each individual node, we define the capacity of the entire index:

Definition 12 (Capacity and Validity of an Index)**.** *Let* $I = (i, f, g, h)$ *be an index. The* capacity *of* I*, denoted by* $\mathsf{cap}(I)$*, is the sum of the following numbers:*

1. $|F^\star| + |\{u \in V(T) \setminus F^\star : f(u) = \mathsf{true}\}| + \sum_{e=(u,v) \in A(T)} g(e)$.
2. *The sum of the* I*-capacities of all the nodes in* T.

Moreover, I *is* valid *if its capacity is at least* $i + |F^\star|$.

Finally, we are ready to define the table M.

Definition 13 (Table M)**.** *The table* M *has an Boolean entry* $M[I]$ *for every valid index* I*. Access to* M *using an argument that is not a valid index is supposed to return* false.

The Meaning of the Table M

Recall (from the The Definition of the Table M section) that the order in which we map nodes to B corresponds to σ. Concretely, when we consider an index (i, f, g, h), the set of nodes that we map is the set of the first i nodes in σ, denoted as follows.

Definition 14 (v_i and $V_{\leq i}$)**.** *For any* $i \in \{0, 1, \ldots, n - |F^\star|\}$*,* v_i *denotes the* i*-th node (from* $V(D) \setminus F^\star$*) in* σ*; additionally,* $V_{\leq i} = \{v_1, v_2, \ldots, v_i\}$.

Additionally, recall that, before Definition 11, we stated that the capacity of a node u concerns the descendants of the node corresponding to u in B that belong to some "particular set". Now, we present the precise definition of this set. Importantly, notice that this definition is dependent on the choice of a realizability witness, while the definition of a capacity (and, hence, of the table M) is not.

Definition 15 (Node Slots)**.** *Let* $I = (i, f, g, h)$ *be an index. Let* rea *be a realizability witness for* $(T, \mathsf{len}, \mathsf{siz})$*. The* set of (I, rea)-slots of a node $u \in V(T)$*, denoted by* $\mathsf{slots}_{I,\mathsf{rea}}(u)$*, is defined as follows:*

1. *If* $u \in F^\star$*:* $\mathsf{slots}_{I,\mathsf{rea}}(u) = \bigcup_{c \in C_{u,h,\mathsf{rea}}} V(B_c)$ *where* $C_{u,h,\mathsf{rea}}$ *is the set of the* $h(u)$ *children of* $\mathsf{rea}(u)$ *in* B*—excluding those in the image of* rea*—whose subtrees (in* B*) are largest.*
2. *If* $u \notin F^\star$ *and it is* I*-open:* $\mathsf{slots}_{I,\mathsf{rea}}(u) = V(B_u) \setminus \Big(\bigcup_{(u,v) \in A(T)} V(P_B(\mathsf{rea}(u), \mathsf{rea}(v))) \cup V(B_{\mathsf{rea}(v)})\Big)$.
3. *If* $u \notin F^\star$ *and it is* I*-closed:* $\mathsf{slots}_{I,\mathsf{rea}}(u) = \emptyset$.

At this point, one may wonder why we define the set of slots of an I-closed node u to be empty, or, more generally, why we even need the definition of being open or closed. In particular, given that we have already filled some of the nodes on paths of the form $P_B(\mathsf{rea}(u), \mathsf{rea}(v))$ for $(u, v) \in A(T)$, why should we not allow to fill the descendant of these nodes that are not descendants of yet unfilled

[1] Follows from Definition 5 and Condition 2 in Definition 9.

nodes? Unfortunately, doing this violates correctness; in particular, the number of nodes of the aforementioned type depends on the choice of rea, while there can be $n^{\Omega(k)}$ choices for rea! Specifically, for the sake of efficiency, it is critical that the definition of our table (and its computation in the next subsection) will be independent of rea. Generally, let us remark that each condition in each of our definitions is critical for efficiency or correctness (mostly, for correctness).

Next, having defined the slots of each individual node, we define the slots of the entire index:

Definition 16 (Index Slots)**.** *Let $I = (i, f, g, h)$ be an index. Let* rea *be a realizability witness for $(T, \mathsf{len}, \mathsf{siz})$. The* set of rea-slots of I*, denoted by $\mathsf{slots}_{\mathsf{rea}}(I)$, is defined as follows:*

1. $\{\mathsf{rea}(u) : u \in F^\star\} \cup \{\mathsf{rea}(u) : u \in V(T) \setminus F^\star, f(u) = \mathsf{true}\} \cup \bigcup_{e=(u,v)\in A(T)} V(P_B^{g(e)}(\mathsf{rea}(u), \mathsf{rea}(v)))$.
2. *The union of the sets of (I, rea)-slots of the nodes in T.*

Notice that, due to Condition 1 in Definition 7 and Observation 1, we have that $V(P_B^{g(e)}(\mathsf{rea}(u), \mathsf{rea}(v))) \cap V(P_B^{g(e')}(\mathsf{rea}(u'), \mathsf{rea}(v'))) = \emptyset$ for any $e = (u, v), e' = (u', v') \in A(T)$ such that $e \neq e'$ (even if $\{u, v\} \cap \{u', v'\} \neq \emptyset$). Thus, the following observation is immediate.

Observation 3. *Let* rea *be a realizability witness for $(T, \mathsf{len}, \mathsf{siz})$. For any node $u \in V(T)$, $|\mathsf{slots}_{I,\mathsf{rea}}(u)| = \mathsf{cap}_I(u)$. Moreover, $|\mathsf{slots}_{\mathsf{rea}}(I)| = \mathsf{cap}(I)$.*

Our main (and most technical) definition is that of an I-witness, given below. Intuitively, an I-witness is the precise definition of the form of the partial solutions whose existence is to be verified when computing the table entry $M[I]$. In Conditions 1, 2a and 2b, the partial solution is enforced to correspond to T, f, g and h as discussed in the paragraph before Definition 4 and the paragraph below Definition 9; this is done, implicitly, through the use of a realizability witness for $(T, \mathsf{len}, \mathsf{siz})$. Here, Condition 2b is stronger than the condition discussed in paragraph below Definition 9—not only do we enforce $h(u)$ children to be in the image, but we also identify which children they should be. This is necessary in order to be consistent with our definitions of capacity and slots. Condition 2c enforces an additional precedence constraint on the order in which the nodes of B are filled, which will be critical for the correctness of our algorithm—intuitively, we must enforce that if we have mapped a node in D to some node in B, then we have already mapped a node in D also to the parent of that node in B, since, otherwise, we will no be able to map a node to that parent later on. Lastly, Condition 3 is the one that enforces that our partial solution is indeed part of a solution, that is, an isomorphism.

Definition 17 (I-Witness)**.** *Let $I = (i, f, g, h)$ be an index. For a realizability witness* rea *for $(T, \mathsf{len}, \mathsf{siz})$, an injective function $\mathsf{wit} : F^\star \cup V_{\leq i} \to \mathsf{slots}_{\mathsf{rea}}(I)$ is an (I, rea)-*witness *if all of the following conditions hold:*

1. *For any node $u \in F^\star$, $\mathsf{wit}(u) = \mathsf{rea}(u)$.*
2. *The image of* wit *contains the following sets:*
   (a) $\{\mathsf{rea}(u) : u \in V(T) \setminus F^\star : f(u) = \mathsf{true}\} \cup \bigcup_{e=(u,v)\in A(T)} V(P_B^{g(e)}(\mathsf{rea}(u), \mathsf{rea}(v)))$.
   (b) $\bigcup_{u\in F^\star} C_{u,h,\mathsf{rea}}$.
   (c) $\{\mathsf{parent}_B(\mathsf{wit}(u)) : u \in V_{\leq i}\}$.
3. *For any pair of nodes $u, v \in F^\star \cup V_{\leq i}$ such that $(\mathsf{wit}(u), \mathsf{wit}(v)) \in A(B)$, $(u, v) \in A(D)$.*

*A function is an I-*witness *if there exists a realizability witness* rea *for $(T, \mathsf{len}, \mathsf{siz})$ such that it is an (I, rea)-witness.*

Finally, we state the "meaning" of the table M: For any valid index I, we would like $M[I]$ to store true if and only if there exists an I-witness.

Basis and Recursive Formula

The order of computation of the entries of M is increasing with respect to i (the first argument of an index), where the order between entries having the same i is arbitrary. In what follows, we consider some valid index $I = (i, f, g, h)$.

Basis. The basis is when $i = 0$. We have two cases:

1. If f is the function that assigns only false, g is the function that assigns only 0, and h is the function that assigns only 0, then $M[I]$ stores true.
2. Otherwise, $M[I]$ stores false.

Recursive Formula. Now, suppose that $i \geq 1$. Then, $M[I]$ stores true if and only if at least one of the following entries stores true. (Here, we remind that access to M with an argument that is not a valid index, which might be performed in some of the cases below, returns false.)

1. $M[(i-1, f, g, h)]$.
2. $M[(i-1, f', g, h)]$ for any f' such that:
   (a) f' is identical to f except for exactly one node $u^\star \in V(T) \setminus F^\star$ to which f assigns true but f' assigns false.
   (b) For every outgoing-arc $(u^\star, v)$ of $u^\star$ in $A(T)$ such that $v \in F^\star$: $(v_i, v) \in A(D)$.
   (c) For the ingoing-arc $(v, u^\star)$ of $u^\star$ in $A(T)$: if $v \in F^\star$, then $(v, v_i) \in A(D)$.
3. $M[(i-1, f, g', h)]$ for any g' that is identical to g except for exactly one arc $e^\star$ to which g assigns some number $\ell \geq 1$ but g' assigns $\ell - 1$.
4. $M[(i-1, f, g, h')]$ for any h' such that:
   (a) h' is identical to h except for exactly one node $u^\star \in F^\star$ to which h assigns some number $\ell \geq 1$ but h' assigns $\ell - 1$.
   (b) $(u^\star, v_i) \in A(D)$.

Intuitively, the various cases correspond to the "locations" in B to which v_i is mapped. In the first case, it is mapped to a node that has no neighbours nor descendants corresponding to nodes in $F^\star$. In the second case, it is mapped to a node corresponding to a node in T. In the third case, it is mapped to an internal node on the path between two nodes corresponding to nodes in T. Lastly, in the fourth case, it is mapped to the child corresponding to a node in $F^\star$.

Time Complexity and Correctness

Due to lack of space, we only state the time complexity (in Lemma 2) and the correctness (in Lemma 3) of the dynamic programming algorithm. While the proof of Lemma

2 is simple, the proof of Lemma 3 requires non-trivial technical work. Here, correctness means that the table M indeed stores what it is meant to (as we describe in the The Meaning of the Table M section).

Lemma 2. *Table M can be computed in time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.*

Lemma 3. *For any valid index I, $M[I] = \mathsf{true}$ if and only if there exists an I-witness.*

Putting It All Together

For the description of the algorithm, we define the index that will correspond to the output of the algorithm (specifically, to the table entry that interests us) as follows.

Definition 18. *Let (D, w) be an instance of* Tournament Fixing. *Let $(T, \mathsf{len}, \mathsf{siz})$ be a realizable annotated structure. Let $I = (i, f, g, h)$ such that: $i = |V(D) \setminus F^\star|$; f is the function that, for each node $u \in V(T) \setminus F^\star$, assigns true; g is the function that, for each arc $e \in A(T)$, assigns $\mathsf{len}(e)$; h is the function that, for each node $u \in F^\star$, assigns $\log_2 \mathsf{siz}(u)$. Then, I is termed the* index of $(T, \mathsf{len}, \mathsf{siz})$.

For Definition 18, the observation below is immediate.

Observation 4. *Let (D, w) be an instance of* Tournament Fixing. *Let $(T, \mathsf{len}, \mathsf{siz})$ be a realizable annotated structure. Then, the index of $(T, \mathsf{len}, \mathsf{siz})$ is a valid index.*

The Algorithm: Given an instance (D, w) of Tournament Fixing, the algorithm works as follows:

1. Call the algorithm in Lemma 1 to compute a collection $\mathcal{C}$ of realizable annotated structures.
2. For every $(T, \mathsf{len}, \mathsf{siz}) \in \mathcal{C}$: *(A)* Compute the table M corresponding to $(T, \mathsf{len}, \mathsf{siz})$ (see the Dynamic Programming section). *(B)* If $M[I] = \mathsf{true}$ where I is the index of $(T, \mathsf{len}, \mathsf{siz})$, then output Yes.
3. Output No.

Time Complexity: According to Lemma 1, the runtime of its algorithm and $|\mathcal{C}|$ are bounded from above by $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. Then, Lemma 2 yields that, for every $(T, \mathsf{len}, \mathsf{siz}) \in \mathcal{C}$, the algorithm spends $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. So, in total, the time complexity is bounded from above by $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

Correctness: We require the following lemma, whose proof is relegated to the Appendix.

Lemma 4. *Let (D, w) be an instance of* Tournament Fixing. *Then:*

1. *Let S be a (D, w)-BA S with a corresponding isomorphism iso. Let $(T, \mathsf{len}, \mathsf{siz})$ be a realizable annotated structure that complies with it. Then, iso is an I-witness where I is the index of $(T, \mathsf{len}, \mathsf{siz})$.*
2. *Let $(T, \mathsf{len}, \mathsf{siz})$ be a realizable annotated structure. Let wit be an I-witness where I is the index of $(T, \mathsf{len}, \mathsf{siz})$. Define S as follows: $V(S) = V(D)$; $A(S) = \{(u, v) : (\mathsf{wit}(u), \mathsf{wit}(v)) \in A(B)\}$. Then, S is a (D, w)-BA.*

In one direction, suppose that (D, w) is a yes-instance of Tournament Fixing. By Proposition 2, there exists a (D, w)-BA S. Let iso be the corresponding isomorphism. By Lemma 1, $\mathcal{C}$ contains a realizable annotated structure $(T, \mathsf{len}, \mathsf{siz})$ that complies with it. We consider the iteration of the algorithm corresponding to $(T, \mathsf{len}, \mathsf{siz})$. By Lemma 4, iso is an I-witness where I is the index of $(T, \mathsf{len}, \mathsf{siz})$. So, by Observation 4 and Lemma 3, $M[I] = \mathsf{true}$ (in this iteration), and the algorithm returns Yes.

In the other direction, suppose that the algorithm returns Yes. So, $\mathcal{C}$ contains a realizable annotated structure $(T, \mathsf{len}, \mathsf{siz})$ such that $M[I] = \mathsf{true}$ where I is the index of $(T, \mathsf{len}, \mathsf{siz})$. By Observation 4 and Lemma 3, there exists an I-witness. Hence, by Lemma 4, there exists a (D, w)-BA. In turn, by Proposition 2, this implies that (D, w) is a yes-instance of Tournament Fixing. Thus, we conclude the correctness of Theorem 1.

Conclusion

Our work resolved the question of whether Tournament Fixing is FPT parameterized by the feedback vertex set number in the affirmative. We believe that our algorithm can be extended so as to deal with bribery similarly to (Gupta et al. 2018b), and, in case w cannot be made the winner, ensure that w will win as many matches as possible.

Having the answer above at hand, several follow-up questions arise:

- First, it would be interesting to chart the limits of tractability of Tournament Fixing with respect to various structural parameters—in particular, as now we know that it is FPT with respect to the feedback vertex set number, is it also FPT with respect to the directed treewidth (which is a smaller parameter)?
- Also, it would be interesting to investigate more general settings of tournament digraphs, such as those of probabilistic tournaments and incomplete tournaments, as well as other variants of tournament competitions, such as double-elimination tournaments.
- Additionally, one might consider the kernelization complexity of Tournament Fixing parameterized by the feedback vertex set number, and the parameterized analysis of the counting version.
- Furthermore, one might consider to implement our algorithm (which should be simple) and test it on real data.
- Lastly, we would like to ask whether the time complexity of our algorithm is optimal. In particular, does Tournament Fixing admit an algorithm with running time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ where k is the feedback vertex set number? While we find it unlikely to attain a running time of $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ when k is the feedback vertex set number, this might not be the case when k is the feedback *arc* set number—so, proving (or refuting) the existence of such a sub-exponential-time parametrized algorithm might be challenging.

References

Aziz, H.; Gaspers, S.; Mackenzie, S.; Mattei, N.; Stursberg, P.; and Walsh, T. 2014. Fixing a Balanced Knockout Tournament. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, 552–558.

Connolly; and Rendleman. 2011. Tournament qualification, seeding and selection efficiency. *Technical Report 2011-96, Tuck School of Business*.

Cygan, M.; Fomin, F. V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer.

Groh; Moldovanu; Sela; and Sunde. 2012. Optimal seedings in elimination tournaments. *Economic Theory*, 49(1): 59–80.

Gupta, S.; Roy, S.; Saurabh, S.; and Zehavi, M. 2018a. When Rigging a Tournament, Let Greediness Blind You. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 275–281.

Gupta, S.; Roy, S.; Saurabh, S.; and Zehavi, M. 2018b. Winning a Tournament by Any Means Necessary. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 282–288.

Gupta, S.; Saurabh, S.; Sridharan, R.; and Zehavi, M. 2019. On Succinct Encodings for the Tournament Fixing Problem. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 322–328.

Horen; and Riezman. 1985. Comparing Draws for Single Elimination Tournaments. *Operations Research*, 33(2): 249–262.

Kim, M. P.; Suksompong, W.; and Williams, V. V. 2016. Who Can Win a Single-Elimination Tournament? In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, 516–522.

Kim, M. P.; and Williams, V. V. 2015. Fixing Tournaments for Kings, Chokers, and More. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 561–567.

Laslier. 1997. Tournament Solutions and Majority Voting. *Springer-Verlag*.

Ramanujan, M. S.; and Szeider, S. 2017. Rigging Nearly Acyclic Tournaments Is Fixed-Parameter Tractable. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, 3929–3935.

Rosen. 1986. Prizes and incentives in elimination tournaments. *The American Economic Review*, 76(4): 701–715.

Roy, S. 2020. Select, Allocate, and Manipulate via Multivariate Analysis, PhD Thesis [HBNI Th184].

Stanton, I.; and Williams, V. V. 2011a. Manipulating Stochastically Generated Single-Elimination Tournaments for Nearly All Players. In *Internet and Network Economics - 7th International Workshop, WINE 2011, Singapore, December 11-14, 2011. Proceedings*, 326–337.

Stanton, I.; and Williams, V. V. 2011b. Rigging Tournament Brackets for Weaker Players. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 357–364.

Suksompong, W. 2021. Tournaments in Computational Social Choice: Recent Developments. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, 4611–4618.

Tullock. 1980. Toward a Theory of the Rent-seeking Society. *Texas A&M University Press*.

Vu, T.; Altman, A.; and Shoham, Y. 2009. On the complexity of schedule control problems for knockout tournaments. In *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15, 2009, Volume 1*, 225–232.

Williams, V. V. 2010. Fixing a Tournament. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

Williams, V. V. 2016. Knockout Tournaments. In *Handbook of Computational Social Choice*, 453–474.