When Rigging a Tournament, Let Greediness Blind You

Sushmita Gupta[1], **Sanjukta Roy**[2], **Saket Saurabh**[1,2] and **Meirav Zehavi**[3]
[1] University of Bergen, Bergen, Norway
[2] The Institute of Mathematical Sciences, HBNI, Chennai, India
[3] Ben-Gurion University, Beersheba, Israel
sushmita.gupta@gmail.com, sanjukta@imsc.res.in, saket@imsc.res.in, meiravze@bgu.ac.il

Abstract

A knockout tournament is a standard format of competition, ubiquitous in sports, elections and decision making. Such a competition consists of several rounds. In each round, all players that have not yet been eliminated are paired up into matches. Losers are eliminated, and winners are raised to the next round, until only one winner exists. Given that we can correctly predict the outcome of each potential match (modelled by a tournament D), a seeding of the tournament deterministically determines its winner. Having a favorite player v in mind, the Tournament Fixing Problem (TFP) asks whether there exists a seeding that makes v the winner. Aziz et al. [AAAI'14] showed that TFP is NP-hard. They initiated the study of the parameterized complexity of TFP with respect to the feedback arc set number k of D, and gave an XP-algorithm (which is highly inefficient). Recently, Ramanujan and Szeider [AAAI'17] showed that TFP admits an FPT algorithm, running in time $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$. At the heart of this algorithm is a translation of TFP into an algebraic system of equations, solved in a black box fashion (by an ILP solver). We present a fresh, purely combinatorial greedy solution. We rely on new insights into TFP itself, which also results in the better running time bound of $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$. While our analysis is intricate, the algorithm itself is surprisingly simple.

1 Introduction

A knockout tournament is a standard format of competition, consisting of several rounds. In each round, all players that have not yet been eliminated are paired up into matches. Losers are eliminated, and winners are raised to the next round, until only a single winner exists. For an illustrative example, consider Wimbledon Men's tennis tournament (having 128 players). In fact, knockout tournament is the most widely-used format of competition in sports [Horen and Riezman, 1985; Connolly and Rendleman, 2011; Groh *et al.*, 2012]. Apart from sports and popular culture, knockout tournaments are also prevalent in elimination-based election and decision making schemes. Specifically, they received notable attention from the viewpoints of artificial intelligence [Vu *et al.*, 2009; Williams, 2010; Stanton and Williams, 2011c; 2011b; 2011a; Aziz *et al.*, 2014; Michael *et al.*, 2016; Kim and Williams, 2015; Ramanujan and Szeider, 2017] as well as economics and operation research [Rosen, 1986; Tullock, 1980; Laslier, 1997].

Having a favorite player $v^\star$ in mind, is there a way to conduct the competition so that $v^\star$ wins? To formulate this question formally, note that the execution of a knockout tournament with a set N of N players is governed by a complete (unordered) binary tree T with n leaves and a mapping $\varphi : N \to \text{leaves}(T)$, 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 vertex, mapped to a player who is declared the winner. The question of making $v^\star$ a winner is sensible when we have predictive information about outcomes of matches. Specifically, we assume we have a tournament $D = (V, A)$ (not to be confused with the competition itself that is also termed a tournament), which is a digraph where either $(u, v) \in A$ or $(v, u) \in A$ for all $u, v \in V$. This encodes predictive information as follows : $V = N$, 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 $(u, v) \in A$.

Now, our question is formalized as follows.

TOURNAMENT FIXING PROBLEM (TFP)
Input: A tournament $D = (V, A)$, and a vertex $v^\star \in V$.
Parameter: The feedback arc set number $k^\star$ of D.
Question: Is there a seeding of the $n = |V|$ players such that $v^\star$ wins the resulting knockout tournament?

The problem of rigging a knockout tournament was introduced by Vu et al. [Vu *et al.*, 2009]. This work led to a flurry of research of structural properties of D that guarantee that $v^\star$ wins [Michael *et al.*, 2016; Kim and Williams, 2015; Stanton and Williams, 2011c; 2011b; 2011a; Aziz *et al.*, 2014; Michael *et al.*, 2016; Williams, 2010]. Additional information can be found in surveys such as [Williams, 2016]. The question of whether TFP is NP-hard was posed in several works, including [Vu *et al.*, 2009; Williams, 2010; Russell and Beek, 2011; Stanton and Williams, 2011b; 2011c; 2011a;

Lang *et al.*, 2012]. It was finally resolved in the affirmative by Aziz et al. [Aziz *et al.*, 2014], showing that it can be solved in time $\mathcal{O}(2.83^n)$. Later, Kim and Williams [Kim and Williams, 2015] gave an $\mathcal{O}(2^n)$ time and space algorithm. Even for a small number of players, such a running time is prohibitive. Moreover, approximation of TFP does not make sense.

In light of the discussion above, a most natural framework to study TFP is parameterized complexity. Specifically, Aziz et al. [Aziz *et al.*, 2014] considered the feedback arc set number of D as the parameter $k^\star$. This parameter has a natural interpretation. It is likely that in real world scenarios, there is 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. The number of matches where we guess (before the competition begins) that a player of a certain rank will beat a player of higher rank, which upper bounds $k^\star$, is significantly smaller than n.

In the last decade, parameterized complexity has transformed into a rich and vibrant field, being one of the most intensively studied areas in theoretical computer science (see fpt.wikidot.com/fpt-papers-in-conferences). With some delay, using methods in parameterized complexity to resolve problems in social choice theory is also becoming common. In particular, this line of attack has been proven particularly successful in voting theory (see, e.g., [Bulteau *et al.*, 2015; Bredereck *et al.*, 2016; Dey *et al.*, 2017; Bredereck *et al.*, 2014b; Dey *et al.*, 2016; Chen *et al.*, 2017; Bredereck *et al.*, 2014c; 2014c]); for more information on the current state-of-the-art, we refer to excellent surveys such as [Bredereck *et al.*, 2014a; Betzler *et al.*, 2012; Faliszewski and Niedermeier, 2016; Fischer *et al.*, 2016].

Aziz et al. [Aziz *et al.*, 2014] showed that TFP is XP, which means solvability in time $n^{f(k^\star)}$ for some function f of $k^\star$. They gave a clever dynamic programming algorithm that runs in time $n^{\mathcal{O}(k^\star)}$. Assuming that $k^\star$ is not fixed (i.e., independent of n), such a running time is again prohibitive. The question of whether TFP is fixed-parameter tractable with respect to $k^\star$, that is, solvable in time $f(k^\star)n^{\mathcal{O}(1)}$ for some function f, was later resolved by Ramanujan and Szeider [Ramanujan and Szeider, 2017]. They showed that TFP is solvable in time $2^{\mathcal{O}(k^{\star 2}\log k^\star)}n^{\mathcal{O}(1)}$. This means that whenever $k^{\star 2}\log k^\star = \mathcal{O}(\log n)$, TFP is solvable in polynomial time.

1.1 Our Contribution

The main component of the algorithm by Ramanujan and Szeider [Ramanujan and Szeider, 2017] is the translation of TFP into an algebraic system of equations, solved in a black box fashion by an ILP solver. In particular, the algorithm (and not just the proof of correctness) by Ramanujan and Szeider [Ramanujan and Szeider, 2017] is thus both complicated and not self-contained. We present a fresh greedy solution whose correctness relies on new insights into TFP itself. The advantages of our algorithm are as follows.

First, our algorithm is *purely combinatorial* (unlike the algebraic algorithm of [Ramanujan and Szeider, 2017]). While the analysis of our algorithm is intricate, the algorithm itself is *simple* and easily implementable. Essentially, after we "guess" a template tree (similarly to [Ramanujan and Szeider, 2017]), our algorithm merely iterates over the paths and subtrees that are not already determined by the guess, and fills them up in a greedy manner. We find it both surprising and pleasing that such a strategy just works. From our proof of correctness, the reasons for the validity of this strategy are revealed, but at first glance, we do not find them apparent.

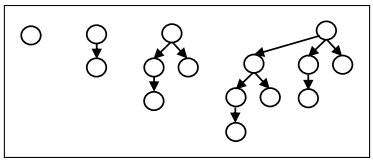

Figure 1: Binomial arborescnences of sizes $2^0, 2^1, 2^2$ and 2^3.

Second, our solution is completely *self-contained*. We do not invoke any black-box ([Ramanujan and Szeider, 2017] invokes a big hammer, i.e., an ILP solver).

Third, our algorithm is substantially faster than that of [Ramanujan and Szeider, 2017]. Specifically, our running time bound is $2^{\mathcal{O}(k^\star \log k^\star)}n^{\mathcal{O}(1)}$ rather than $2^{\mathcal{O}(k^{\star 2}\log k^\star)}n^{\mathcal{O}(1)}$. The difference is clear already for values as small as $k^\star = 8$, where $2^{k^\star \log_2 k^\star} = 2^{24}$ while $2^{k^{\star 2}\log_2 k^\star} = 2^{192}$. Even for $k^\star = 4$, $2^{k^2 \log_2 k} = 2^{32} >> 2^{24}$. Thus, it is very plausible that our improvement constitutes the difference between being practically infeasible and feasible for social choice applications, where n typically ranges between a few tens to a few hundreds, if $k^\star$ is presumed to be, say, 5% of n. In a different view, our result also proves that whenever $k \log k = \mathcal{O}(\log n)$, TFP is solvable in polynomial time.

Proofs omitted entirely are denoted by $\star$.

2 Preliminaries

Let $[n] = \{1, \ldots, n\}$. Given a tournament D, let $V(D)$ and $A(D)$ denote its vertex set and arc set, respectively. For a pair of vertices u, v, by (u, v) we denote an arc from u to v, and say that u is an in-neighbor of v and v is an out-neighbor of u. A uv-path is a path from u to v. Given $X \subseteq V(D)$, by $D[X]$ we denote the subtournament of D induced by X and $N(X)$ is the set of neighbors of the vertices in X outside X. For a rooted tree T and a vertex $v \in V(T)$, by T_v we denote the subtree of T rooted at v. An *arborescence* is a rooted directed tree such that all arcs are directed away from the root.

We now discuss the notion of a binomial arborescence and its relevance to our work (see Fig. 1.).

Definition 1. *An* unlabeled binomial arborescence (**usba**) T *rooted at $v \in V(T)$ is defined recursively as follows:*

- *A single node v is a binomial arborescence rooted at v.*
- *Given two vertex disjoint binomial arborescences of equal size, T_v rooted at v and T_u rooted at u, adding an arc from v to u results in a binomial arborescence T rooted at v.*

Let D be a directed graph. If T is a subgraph of D with $V(T) = V(D)$, then T is a labeled spanning binomial arborescence *(***sba***) of D.*

Given an integer $n = 2^j$, there exists a unique **usba**, denoted by B_n, on n vertices. For our purpose, we *fix a labelling* of the vertices of B_n from $[n]$, that is, a function

$\gamma : V(B_n) \to [n]$. Whenever we refer to an **usba** on n vertices, we always mean B_n, where the vertices are labeled by γ. Our interest in **sba** is due to the following connection between **sba** and finding a seeding of the vertices in D that results in a specific vertex being the winner.

Proposition 1 ([Williams, 2010])**.** *Let D be a tournament with $v^\star \in V(D)$. There is a seeding of the vertices in D such that the resulting knockout tournament is won by $v^\star$ if and only if D has a* **sba** *rooted at $v^\star$.*

Proposition 1 reduces TFP to the problem of finding a **sba** rooted at $v^\star$. Throughout, we take this view of the **sba** problem. In this context, the following lemma will come in handy.

Lemma 1 ($\star$)**.** *Let D be a tournament, $X \subseteq V(D)$ such that $D[X]$ is acyclic, and T be arborescence on $|X|$ vertices. Then, $D[X]$ contains (as a subgraph) an arborescence isomorphic to T that can be computed in polynomial time.*

Next, we give some simple properties of **sba**.

Observation 1 ([Ramanujan and Szeider, 2017])**.** *Let $\mathbf{B}$ be an* **sba** *on n vertices. Then, the height of $\mathbf{B}$ (the maximum length of a path from root to leaf) is $\log n$. Moreover, for any vertex v in $\mathbf{B}$, the number of children of v is at most $\log n$, and the size of the subtree of $\mathbf{B}$ rooted at v is one of the following numbers: $2^0, 2^1, \cdots, 2^{\log n}$.*

3 Greedy Algorithm for TFP

As stated earlier, due to Proposition 1, we study the following equivalent question: *Does there exist an* **sba** *rooted at $v^\star$ in D?* We start by computing a minimum sized feedback arc set, F, of D. It is well known that a tournament has a directed cycle if and only if it has a directed triangle. Furthermore, a subset F of $A(D)$ is a minimal *feedback arc set* if and only if the tournament D^{rev} obtained after reversing the arcs in F is acyclic. Using these two facts, one can design a trivial branching algorithm for finding F running in time $3^{k^\star} n^{\mathcal{O}(1)}$. That is, while there exists a triangle in the given tournament, find one and recursively try to find a smaller sized solution by reversing an arc of the triangle. See [Cygan *et al.*, 2015] for further details. In fact one can find a minimum sized feedback arc set, F, of D in time $2^{\mathcal{O}(\sqrt{k^\star})} n^{\mathcal{O}(1)}$ [Feige, 2009; Karpinski and Schudy, 2010]. From now onwards, we assume that we have a feedback arc set F at hand.

Let σ be the *topological ordering* of D after reversing the arcs in the feedback arc set F. Throughout the paper, we always work with the topological ordering σ. If a vertex v precedes u in σ, then we write $v \prec_\sigma u$. Given $V' \subseteq V(D)$, the *highest vertex in V'* is the vertex that precedes all other vertices of V' in the ordering σ. Furthermore, for any arc (u, v), we say that u *beats* v (equivalently, v is *beaten* by u). We now give a few basic definitions central to our algorithm.

Definition 2. *(***LCA closure***) For a rooted tree T and a vertex subset $M \subseteq V(T)$, the least common ancestor-closure (*LCA-closure*) $\mathsf{LCA}(M)$ is obtained by the following process. Initially, set $M' = M$. Then, as long as there are vertices x and y in M' whose least common ancestor w in T is not in M', add w to M'. When the process terminates, output M' as the LCA-closure of M.*

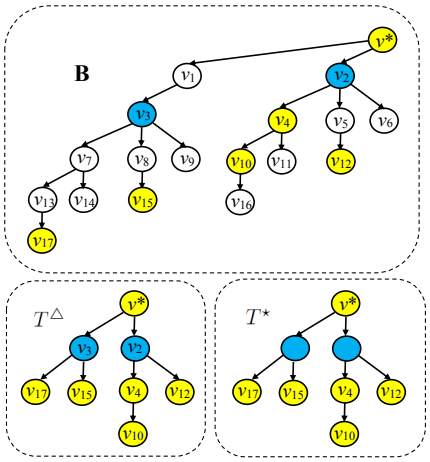

Figure 2: The topology $T^\triangle$ of $\{v^*, v_4, v_{10}, v_{12}, v_{15}, v_{17}\}$ in B (left), and the template $T^\star$ (right). LCA-vertices are colored blue.

The following folklore lemma (see, e.g., [Cygan *et al.*, 2015]) summarizes two basic properties of LCA closures.

Lemma 2. *Let T be a rooted tree, $M \subseteq V(T)$ and $M' = \mathsf{LCA}(M)$. Then $|M'| \leq 2|M|$ and for every connected component C of $T \setminus M'$, $|N(C)| \leq 2$.*

Our algorithm will "guess" some information about a solution (if one exists), and then greedily complete the parts unknown. To discuss these guesses (Phase 1), we require the following definition (see Fig. 2).

Definition 3. *(***Topology** *and* **Template***) Let $\mathbf{B}$ be an* **sba** *rooted at $v^\star$ and let $X \subseteq V(\mathbf{B})$. We construct a tree $T^\triangle$ from $\mathbf{B}$ as follows. First delete all vertices that do not lie on any $v^\star u$-path in $\mathbf{B}$ for any $u \in \mathsf{LCA}(X)$. Now, as long as there is a vertex $u \in V(\mathbf{B}) \setminus \mathsf{LCA}(X)$ with exactly one in-neighbor u^- and exactly one out-neighbor u^+, delete the vertex u and add the arc (u^-, u^+). Then $T^\triangle$ is a tree obtained at the end of this process, it is called the* topology *of X in $\mathbf{B}$. The* template tree *of X in $\mathbf{B}$ is a tree obtained from $T^\triangle$ by deleting the labels of all vertices in $\mathsf{LCA}(X) \setminus X$. A tree $T^\star$ is a* template *of X if it is a template of X in at least one* **sba** $\mathbf{B}$ *rooted at $v^\star$.*

Before going further, we recall some notation.

Let F denote the feedback arc set and V_F denote the set of vertices that are adjacent to arcs in F. We denote $k = |V_F|$. Observe that $k \leq 2k^\star$. Furthermore, σ denote the *topological ordering* of D after reversing the arcs in F.

For the phases where we complete missing information, the following definition will be crucial.

Definition 4. *(***Fas-vertex, Fas-parent** *and* **Fas-child***) Let $T^\star$ be a template of $V_F \cup \{v^\star\}$. Vertices in $V_F \cup \{v^\star\}$ are called* fas-vertex*. Let $v \in V(D)$. The* fas-parent *of v is the last fas-vertex w not equal to v on the $v^\star v$-path in $T^\star$. An* fas-child *of v is an fas-vertex w, that is the first fas-vertex not equal to v on the vw path in $T^\star$. The set of fas-children of v in $T^\star$ is denoted by $\mathsf{faschild}(v)$.*

As a first step, our algorithm would need to enumerate all templates of $V_F \cup \{v^\star\}$. Due to Lemma 2, a template of $V_F \cup \{v^\star\}$ in any **sba** is a rooted tree T on at most $2k+1$ vertices with some of its vertices labeled with the vertices in $V_F \cup \{v^\star\}$ and the others unlabeled, beings *placeholder for LCA vertices*. We can therefore enumerate a superset of templates by enumerating every rooted tree on at most $2k+1$ vertices with a surjective mapping of *some* of its vertices to vertices in $V_F \cup \{v^\star\}$ such that the root is mapped to $v^\star$, and all the leaves are mapped as well (to vertices in $V_F \cup \{v^\star\}$):

Observation 2. *A superset of the set of templates of $V_F \cup \{v^\star\}$ of size $k^{\mathcal{O}(k)}$ can be enumerated in time $k^{\mathcal{O}(k)}$.*

Intuition. Our algorithm proceeds in four steps. First, we guess a partial structure of the **sba**. Second, we verify that the guesses made in the previous step are "realizable". These two steps are straightforward, and are also present in a similar form in [Ramanujan and Szeider, 2017]. Then, we turn to the heart of our algorithm. Here, we design a "greedy" procedure to complete the partial **sba** in two steps. The first fills up paths corresponding to edges of the template at hand. Roughly speaking, here we always select a yet "unhandled" fas-vertex v that is highest in $V(D)$ with respect to σ. Then, we greedily fill the path between v and its fas-parent using the highest "available" vertices that are "between" v and its fas-parent (i.e., beat v and beaten by the parent). This is a slight simplification since when we look at the path between v and its fas-parent, a part of it (closer to the parent) is already filled. However, by considering lca vertices this technicality is easily handled. The second phase fills-up the subtrees hanging from the paths we have just filled. Roughly speaking, while we still have an unfilled subtree, we pick one that hangs from a vertex v that is highest. We greedily fill the subtree with "available" vertices that are the highest ones beaten by v. If the greedy procedure ever gets stuck, we move to the next partial **sba**. While the procedure itself is very simple, correctness is surprising, and indeed the proof is quite intricate.

3.1 Phase I: Guessing

We first intuitively explain what we intend to achieve by guessing. Suppose $I = (D, v^\star, k)$ is a YES-instance of TFP. Then, there exists an **sba** $\mathbf{B}$ rooted at $v^\star$. We first guess the template $T^\star$ of $V_F \cup \{v^\star\}$ in $\mathbf{B}$. Given $T^\star$, for each arc (u, v), we guess a length, denoted by $\mathsf{len}(u, v)$, that is *the number of internal vertices on the unique uv-path in* $\mathbf{B}$. Then, for each vertex $v \in V_F$, we guess the size of the subtree rooted at v in $\mathbf{B}$, denoted by $\mathsf{siz}(v)$. Due to Observation 1, $\mathsf{len}(u, v) \in \{0, \cdots, \log n\}$ and $\mathsf{siz}(v) \in \{2^0, 2^1, \cdots, 2^{\log n}\}$. Formally, in this phase we guess the following.

1. Using Proposition 2, we iterate over all templates of $V_F \cup \{v^\star\}$. Let $T^\star$ denote one such template.
2. For each arc (u, v) in $A(T^\star)$, guess an integer in $\{0, \cdots, \log n\}$. Formally, we enumerate over all possible "length functions" $\mathsf{len} : A(T^\star) \to \{0, \cdots, \log n\}$.
3. Next, we guess the "size function". That is, we enumerate all possible functions $\mathsf{siz} : V_F \to \{2^0, \cdots, 2^{\log n}\}$.

Definition 5. *An* **sba** $\mathbf{B}$ *complies with* $(T^\star, \mathsf{len}, \mathsf{siz})$ *if* (i) *the template of $V_F \cup \{v^\star\}$ in $\mathbf{B}$ is $T^\star$,* (ii) *for each arc $(u, v) \in A(T^\triangle)$ where $T^\triangle$ is the topology of $V_F \cup \{v^\star\}$ in $\mathbf{B}$,the number of internal vertices on the unique uv-path in $\mathbf{B}$ is $\mathsf{len}(u, v)$, and* (iii) *for every $v \in V_F \cup \{v^\star\}$, the size of the subtree rooted at v in $\mathbf{B}$ is $\mathsf{siz}(v)$.*

As we do exhaustive guessing, the following is immediate.

Lemma 3. *If $(D, v^\star, k)$ is a* YES*-instance of* TFP*, there exists an* **sba** $\mathbf{B}$ *that complies with some tuple $(T^\star, \mathsf{len}, \mathsf{siz})$ we enumerate.*

3.2 Phase II: Verification of Guesses

In this section, we give an algorithm to check whether a guessed tuple $(T^\star, \mathsf{len}, \mathsf{siz})$ is "realizable". We first define the verification operations we perform on a tuple $(T^\star, \mathsf{len}, \mathsf{siz})$.

Definition 6 (Realizability). *A tuple $(T^\star, \mathsf{len}, \mathsf{siz})$ is* realizable *if there is a* **usba** B *on n vertices with an injective function $\phi : V(T^\star) \to [n]$ satisfying the properties below.*[1]

1. *$v^\star$ is the root of B, that is, ϕ maps $v^\star$ to the root of B.*
2. *$T^\star$ is the template of $V_F \cup \{v^\star\}$ in B.*
3. *For each $(u, v) \in A(T^\star)$, $\mathsf{len}(uv)$ is equal to the number of internal vertices on the uv-path in B.*
4. *For each $v \in V_F \cup \{v^\star\}$, $\mathsf{siz}(v)$ is equal to the size of the subtree rooted at v in B.*

Lemma 4 ($\star$). *There is a polynomial-time algorithm that checks whether a given tuple $(T^\star, \mathsf{len}, \mathsf{siz})$ is realizable by at least one pair (B, ϕ), and if the answer is positive, outputs such a pair.*

3.3 Phase III: Greedy Choice for Path Resolution

Given a tuple $(T^\star, \mathsf{len}, \mathsf{siz})$, we use a greedy strategy to construct an **sba** $\mathbf{B}$ such that $(T^\star, \mathsf{len}, \mathsf{siz})$ is realizable by $\mathbf{B}$ (if one exists). As a preprocessing step, for every arc $(u, v) \in A(T^\star)$ with $u, v \in V_F \cup \{v^\star\}$ and $\mathsf{len}(u, v) = 0$, we check that indeed $(u, v) \in A(D)$. If this is not the case, then we conclude that there is no **sba** that complies with $(T^\star, \mathsf{len}, \mathsf{siz})$.

Now, recall that our greedy algorithm proceeds in two steps. First, we substitute each arc (v, u) in $T^\star$ by a path, which brings us closer to uncovering some **sba** $\mathbf{B}$. In this section, we only consider the procedure to fill-up these paths. Specifically, given the template $T^\star$, we will construct a tree T that is a template of a superset of $V_F \cup \{v^\star\}$. In particular, it will be a template of $V(T)$ itself.

The tree we construct in the process is referred to as T. Initially, $T = T^\star$. For an arc $(u, v) \in A(T^\star)$, we call (u, v) *resolved* if it has been replaced by a path in T that has $\mathsf{len}(uv)$ internal vertices. An fas-vertex v is called *resolved* if either it is the root ($v^\star$) or all the arcs on the path from its fas-parent, w, to v are resolved. We call a vertex $v \in V(D) \setminus V_F$ *available* if $v \in V \setminus V(T)$. That is, as vertices are added to T to fill the paths corresponding to arcs in $V(T^\star)$, they become unavailable. A vertex is the *highest available* vertex in $X \subseteq V(D)$ if it precedes all other available vertices in X according to σ.

With this notation, we repeat procedure PathGreedy until all arcs are resolved. Note that in one step (that is, one execution of the procedure), several arcs of $A(T^\star)$ can be resolved.

[1] Here, we abuse notation: if a vertex v in $T^\star$ is mapped (by ϕ) to some vertex $i \in [n]$ in B, we refer to i as v.

PathGreedy:

1. Let v be the highest unresolved fas-vertex in $T^\star$ and w be its fas-parent.
2. Consider the wv-path (P_{wv}) in T (not in $T^\star$). Note that in each step we always resolve all the arcs of a path simultaneously. So the set of unresolved arcs on the path P_{wv} form a subpath of P_{wv} that ends at v. Let x be the first vertex on the path P_{wv} such that the no arc on the unique xv-path (P_{xv}) in T is resolved.
3. Let H be the set of available vertices between x and v in the ordering σ. If $|H| \geq \eta = \sum_{(a,b)\in A(P_{xv})}(\mathsf{len}(ab)+1)-1$, then let H' denote the subset of H consisting of highest η vertices in it. Then, we replace the path P_{xv} by the path P'_{xv}, which consists of vertices in H': in the path P'_{xv}, the vertices in H' are ordered from highest to lowest rank.
4. If $|H| < \eta$, return that there is no **sba** that complies with $(T^\star, \mathsf{len}, \mathsf{siz})$.

Let T^i denote the tree T that we had at hand immediately after we resolved the i^{th} vertex. The root is automatically resolved, and hence $T^1 = T^\star$. Note that $T^{k+1} = T$. We say that PathGreedy *fails in step* i, if it concludes in i^{th} step (step in which it resolves the i^{th} vertex) that no **sba** complies with $(T^\star, \mathsf{len}, \mathsf{siz})$, and else we say that it *succeeds* in i^{th} step. We say that PathGreedy *fails* if it concludes that no **sba** complies with $(T^\star, \mathsf{len}, \mathsf{siz})$ in some step, and else we say that it *succeeds*. To argue about the correctness of our greedy choices, we need to extend Definition 5.

Definition 7. *An* **sba** $\mathbf{B}$ *complies with* T^i *if it complies with* $(T^\star, \mathsf{len}, \mathsf{siz})$ *and the template of* $V(T^i) \cap V(D)$ *in* $\mathbf{B}$ *is* T^i.

The main components of proof are Lemmata 5 and 6.

Lemma 5 ($\star$)**.** *Suppose that* PathGreedy *succeeds up to step* i *for some* $i \in [k]$. *Let* $\mathbf{B}$ *be an* **sba** *that complies with* T^i. *Then,* PathGreedy *succeeds at step* $i+1$.

For the sake of clarity, before we present Lemma 6, we define a procedure used only for analysis. Suppose that PathGreedy succeeds up to step $i+1$. Here, suppose we have some **sba** $\mathbf{B}$ that complies with T^i for some $i \in [k]$. As PathGreedy succeeds at step $i+1$, T^{i+1} is well defined. Let v, x, H', P_{xv} and P'_{xv} be as defined in step $i+1$ of PathGreedy. Now, supposing that $\mathbf{B}$ does not comply with T^{i+1}, there exists a vertex p on the unique xv-path in $\mathbf{B}$ that does not belong to H', and a vertex $q \in H'$ that does not belong to the unique xv-path in $\mathbf{B}$. Since the length of these paths are same we have that the only way the last assertion can be violated if both these paths use the same set of vertices but in different order. However, this is not possible since $D[H']$ is a transitive tournament and thus there is a unique directed path spanning all the vertices in H'. Having these (hypothetical) elements at hand, we consider procedure PathAnalysis.

Lemma 6 ($\star$)**.** *Assume* PathGreedy *succeeds up to step* $i+1$, *and let* $\mathbf{B}, v, x, H', P_{xv}, P'_{xv}, p, q$ *be as above. If* $\mathbf{B}$ *does not comply with* T^{i+1}, *then if these elements are the input to* PathAnalysis, *it returns* $\mathbf{B}^\star$ *that is an* **sba** *with the following properties:* (i) *it complies with* T^i; (ii) *the number of vertices on the unique* xv*-path in* $\mathbf{B}^\star$ *from* H' *is larger by* 1 *than the number of vertices on the unique* xv*-path in* $\mathbf{B}$ *from* H'.

PathAnalysis:

1. Let $Q = x \to u_1 \to \cdots \to u_\eta \to v$ be the unique xv-path in $\mathbf{B}$. (This is well defined as $\mathbf{B}$ complies with T^i.)
2. For all $i \in [\eta]$, let u'_i be the i^{th} highest vertex in $(\{u_1, \ldots, u_\eta\} \setminus \{p\}) \cup \{q\}$ according to σ.
3. For all $i \in [\eta]$, put u'_i in place of u_i in $\mathbf{B}$, and put p in (the original) place of q in $\mathbf{B}$. Denote the result by $\mathbf{B}'$.
4. As long as p has a child r' in $\mathbf{B}'$ such that $(r', p) \in A(D)$, let r be the highest such child according to σ, and swap p and r. Denote the result by $\mathbf{B}^\star$.
5. Return $\mathbf{B}^\star$.

From Lemma 6, we derive the following consequence.

Corollary 1 ($\star$)**.** *Suppose that* PathGreedy *succeeds up to step* $i+1$. *If there is an* **sba** $\mathbf{B}$ *that complies with* T^i, *then there is an* **sba** $B^\star$ *that complies with* T^{i+1}.

Having the lemma above at hand, our proof is done by induction. For the sake of clarity, let us extract the claim.

Lemma 7 ($\star$)**.** *For all* $i \in [k+1]$, *if there is an* **sba** $\mathbf{B}$ *complying with* $(T^\star, \mathsf{len}, \mathsf{siz})$, *then* (i) PathGreedy *succeeds up until step* i, *and* (ii) *there is an* **sba** *that complies with* T^i.

As a corollary to Lemma 7, we obtain the following.

Corollary 2. *If there is an* **sba** $\mathbf{B}$ *complying with* $(T^\star, \mathsf{len}, \mathsf{siz})$, *then* (i) PathGreedy *succeeds, and* (ii) *there exists an* **sba** *that complies with* T.

Let us also explicitly state an observation that directly follows from our construction. (Here, correctness also relies on our preprocessing step.)

Observation 3. *If* PathGreedy *succeeds, then* T *is a subtree of* D, *and in particular* $A(T) \subseteq A(D)$.

3.4 Phase IV: Greedy Choices for Subtree Resolution

Because we managed to reach this phase, we have at hand *(i)* a pair (B, ϕ) that realizes $(T^\star, \mathsf{len}, \mathsf{siz})$ (from Phase II), and *(ii)* the tree T outputted by GreedyPaths (from Phase III). Before we proceed to our greedy choices, let us first update (B, ϕ) to $(B^\star, \phi^\star)$ as follows. First, initialize $B^\star = B$. Second, define $\phi^\star : V(T) \to [n]$ as the injective function such that for all $v \in V(T^\star)$, we have $\phi^\star(v) = \phi(v)$, and for all $v \in V(T) \setminus V(T^\star)$, we have $\phi^\star(v) = i$ where i is the j^{th} vertex on the unique $\phi(v^\star)\phi(w)$-path in $B^\star$ for w being the fas-child of v in $T^\star$ of highest rank and j being the position of v on the unique $v^\star w$-path of T. We remark that this notation is well defined because (B, ϕ) realizes $(T^\star, \mathsf{len}, \mathsf{siz})$ and by our construction of T. The choice of w as the fas-child of highest rank does not matter in the sense that any choice of an fas-child w of v would have resulted in the same $\phi^\star(v)$.

A central notion in our greedy choices is of *private vertices*, defined as follows.

Definition 8. *Let* $\widehat{T}$ *be a rooted tree where* $V_F \cup \{v^\star\} \subseteq V(\widehat{T})$. *The set of private* vertices *of a vertex* $v \in V_F \cup \{v^\star\}$

SubtreeGreedy:

1. Let v be the highest unresolved fas-vertex.
2. Let H be the set of the β_v highest available vertices beaten by v. If not enough such vertices are found, return that no **sba** complies with $(T^\star, \mathsf{len}, \mathsf{siz})$.
3. Update W_v to be equal to H.

in $\widehat{T}$, denoted by $\mathsf{priv}_{\widehat{T}}(v)$, is the set of vertices that belong to $\widehat{T}_v$, but not to $\widehat{T}_u$ rooted at a descendant $u \in V_F \setminus \{v\}$ of v.In case $\widehat{T} = T$, denote $\mathsf{priv}(v) = \mathsf{priv}_T(v)$.

We are now ready to describe our second phase of greedy choices. In this phase, each fas-vertex v has a "bucket", which is a set W_v that is initially empty. Moreover, for each fas-vertex v, denote $\alpha_v = \mathsf{siz}(v) - \sum_{u \in \mathrm{faschild}_{T^\star}(v) \cap V_F} \mathsf{siz}(u)$. In addition, denote $\beta_v = \alpha_v - |\mathsf{priv}(v)|$. Here, we say that an fas-vertex v is *unresolved* if its set W_v has not yet been updated. Moreover, we say that a vertex $u \in V(D)$ is *available* if $u \notin V(T) \cup (\bigcup_{v \in V_F \cup \{v^\star\}} W_v)$. As long as there is an unresolved vertex, we apply the steps of procedure SubtreeGreedy.

We say that SubtreeGreedy *fails* if it concludes that no **sba** complies with $(T^\star, \mathsf{len}, \mathsf{siz})$, and else we say that it *succeeds*. For $i \in [k+1]$, let v^i be the fas-vertex considered at step i. To argue about the correctness of our greedy choices, we need to further extend Definition 7.

Definition 9. *An* **sba** $\mathbf{B}$ *complies with v^i if it complies with T and $\mathsf{priv}_{\mathbf{B}}(v) = W_{v^i} \cup \mathsf{priv}(v^i)$. Moreover, $\mathbf{B}$ complies with $v^{\leq i}$ if it complies with all the vertices $v^1, \ldots, v^i$.*

The main argument in our proof are summarized in Lemmata 8, 10 and 11.

Lemma 8 ($\star$)**.** *Suppose that* SubtreeGreedy *succeeds up to step i. Let $\mathbf{B}$ be an* **sba** *that complies with $v^{\leq i}$ for some $i \in [k+1]$. Then,* SubtreeGreedy *succeeds at step $i+1$.*

Towards Lemma 10, we need yet another lemma.

Lemma 9 ($\star$)**.** *For all $i \in [k+1]$ and $u \in \mathsf{priv}(v^i)$, the vertex u beats all the vertices in W_{v^i}.*

We are now ready to state Lemma 10.

Lemma 10 ($\star$)**.** *Suppose that* PathGreedy *succeeds up to step $i+1$ for some $i \in [k+1]$. Let $\mathbf{B}$ be an* **sba** *that complies with $v^{\leq i}$. Then, there is an* **sba** *that complies with $v^{\leq i+1}$.*

Towards the presentation of Lemma 11, we need the SubtreeCompletion procedure. Lemmata 8 and 10 are not sufficient to complete the proof, since they are based on the supposition that there exists an **sba** that complies with $(T^\star, \mathsf{len}, \mathsf{siz})$, which is precisely the information we need to determine. From these lemmata alone, we would only be able to conclude that if there is an **sba** $\mathbf{B}$ complying with $(T^\star, \mathsf{len}, \mathsf{siz})$, then SubtreeGreedy succeeds, but *not* that if SubtreeGreedy succeeds, then there is an **sba** $\mathbf{B}$ complying with $(T^\star, \mathsf{len}, \mathsf{siz})$. The reverse direction will be ensured using Lemma 11. In case we do not only want to determine whether there exists an **sba** that makes $v^\star$ the winner, but also compute one such **sba**, the procedure below should be performed (that is, in that case this procedure is not given only for the sake of analysis). Here, it will also become apparent why we had to have at hand a pair (B, ϕ) that realizes our initial guess. A surprising implication of the proof below is that, although there can be many pairs that realize our initial guess, for the sake of exhibiting an **sba** that complies with our initial guess, taking any one of them would work. Initialize $(B^0, \phi^0) = (B^\star, \phi^\star)$. For $i = 1, 2, \ldots, k+1$, $B^i = B^{i-1}$, and the procedure executes SubtreeCompletion.

SubtreeCompletion:

1. Initialize ϕ^i to be ϕ^{i-1}, and U to be W_{v^i}.
2. For all $u \in \mathsf{priv}(v^i)$, execute the following operations:
   (a) Let $\widehat{B}_u$ be the subtree of B^i_u from which we remove the subtree of t for all $t \in [n]$ for which there is a vertex $w \in V(T) \setminus \{u\}$ such that $\phi^i(w) = t$.
   (b) Let U_u be some subset of $|V(\widehat{B}_u)| - 1$ vertices from U, and update U to be $U \setminus U_u$.
   (c) Note that $D[\{u\} \cup U_u]$ is acyclic. By Lemma 1, $D[\{u\} \cup U_u]$ contains (as a subgraph) an $\widehat{A}_u$ arborescence isomorphic to $\widehat{B}_u$ that is computable in polynomial time. Observe that u is the root of $\widehat{A}_u$, else u were been beaten by at least one vertex in U_u, contradicting Lemma 9.
   (d) We update ϕ^i to label the vertices of $\widehat{B}_u$ according to their "copies" in $\widehat{A}_u$. (More precisely, let f be an isomorphism from $\widehat{B}_u$ to $\widehat{A}_u$. Then, for all $t \in V(\widehat{B}_u)$, define $\phi^i(t) = f(t)$.)

The arguments given in the presentation of the procedure above directly imply the correctness of the following lemma.

Lemma 11 ($\star$)**.** *Suppose that* SubtreeGreedy *succeeds up to step i. Then,* (1) *for all $(u, v) \in A(B^{i+1})$ such that both u, v are in the image of ϕ^{i+1} and are private vertices of v^i in B^i, we have $(\phi^i(x), \phi^i(y)) \in A(D)$, and* (2) *$\phi^i$ extends ϕ^{i-1}, and it injectively maps all the vertices in W_{v^i}.*

From Lemmata 8, 10 and 11, we have the following lemma.

Lemma 12 ($\star$)**.** *For all $i \in [k+1] \cup \{0\}$, if there is an* **sba** $\mathbf{B}$ *complying with $(T^\star, \mathsf{len}, \mathsf{siz})$, then* (i) SubtreeGreedy *succeeds up until step i and there exists an* **sba** *complying with $v^{\leq i}$,* (ii) *for all $(u, v) \in A(B^i)$ such that both u, v are in the image of ϕ^i, we have $(x, y) \in A(D)$ where $\phi^i(x) = u$ and $\phi^i(y) = v$, and* (iii) *ϕ^i extends ϕ^j for all $j < i$, and it injectively maps all the vertices in $V(T) \cup W_{v^1} \cup \ldots W_{v^i}$.*

From Lemma 12, we derive the following corollary.

Corollary 3 ($\star$)**.** *If there is an* **sba** $\mathbf{B}$ *complying with $(T^\star, \mathsf{len}, \mathsf{siz})$, then* SubtreeGreedy *succeeds and $\mathbf{B}^{k+1}$ (where we abuse notation such that each vertex i is labelled by $\phi^{k+1^{-1}}(i)$) is an* **sba** *rooted at $v^\star$.*

Finally, the correctness of our main result, summarized in the theorem below, directly follows from Lemma 3 and Corollary 3. We remark that here we rely on the straightforward observation that given an **sba** where $v^\star$ is the root, a seeding to make $v^\star$ win can be computed in polynomial time. The running time of the algorithm is dominated by the guesses ($k^{\mathcal{O}(k)}$) in the first phase, as all other phases run in polynomial time.

Theorem 1. TFP *is solvable in time* $2^{\mathcal{O}(k^\star \log k^\star)} n^{\mathcal{O}(1)}$. *Moreover, for a* YES*-instance* $(D, v^\star, k^\star)$*, a seeding to make* $v^\star$ *win can be constructed within this time bound.*

References

[Aziz *et al.*, 2014] Aziz, Gaspers, Mackenzie, Mattei, Stursberg, and Walsh. Fixing a balanced knockout tournament. In *28th AAAI*, pages 552–558, 2014.

[Betzler *et al.*, 2012] Betzler, Bredereck, Chen, and Niedermeier. Studies in computational aspects of voting - A parameterized complexity perspective. In *The Multivariate Algorithmic Revolution*, pages 318–363, 2012.

[Bredereck *et al.*, 2014a] Bredereck, Chen, Faliszewski, Guo, Niedermeier, and Woeginger. Parameterized algorithmics for computational social choice. *Tsinghua Science and Technology*, 19(4):358, 2014.

[Bredereck *et al.*, 2014b] Bredereck, Chen, Faliszewski, Nichterlein, and Niedermeier. Prices matter for the parameterized complexity of shift bribery. In *28th AAAI*, pages 1398–1404, 2014.

[Bredereck *et al.*, 2014c] Bredereck, Chen, Hartung, Kratsch, Niedermeier, Suchý, and Woeginger. A multivariate complexity analysis of lobbying in multiple referenda. *J. Artif. Intell. Res.*, 50:409–446, 2014.

[Bredereck *et al.*, 2016] Bredereck, Faliszewski, Niedermeier, and Talmon. Complexity of shift bribery in committee elections. In *30th AAAI*, pages 2452–2458, 2016.

[Bulteau *et al.*, 2015] Bulteau, Chen, Faliszewski, Niedermeier, and Talmon. Combinatorial voter control in elections. *Theor. Comput. Sci.*, 589:99–120, 2015.

[Chen *et al.*, 2017] Chen, Faliszewski, Niedermeier, and Talmon. Elections with few voters: Candidate control can be easy. *J. Artif. Intell. Res.*, 60:937–1002, 2017.

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

[Cygan *et al.*, 2015] Cygan, Fomin, Kowalik, Lokshtanov, Marx, Pilipczuk, Pilipczuk, and Saurabh. *Parameterized Algorithms*. Springer, 2015.

[Dey *et al.*, 2016] Dey, Misra, and Narahari. Kernelization complexity of possible winner and coalitional manipulation problems in voting. *Theor. Comput. Sci.*, 616:111–125, 2016.

[Dey *et al.*, 2017] Dey, Misra, and Narahari. Parameterized dichotomy of choosing committees based on approval votes in the presence of outliers. In *16th AAMAS*, pages 42–50, 2017.

[Faliszewski and Niedermeier, 2016] Faliszewski and Niedermeier. Parameterization in computational social choice. In *Encyclopedia of Algorithms*, pages 1516–1520. 2016.

[Feige, 2009] Feige. Faster FAST. *abs/0911.5094*, 2009.

[Fischer *et al.*, 2016] Fischer, Hudry, and Niedermeier. Weighted tournament solutions. In *Handbook of Computational Social Choice*, pages 85–102. 2016.

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

[Horen and Riezman, 1985] Horen and Riezman. Comparing draws for single elimination tournaments. *Operations Research*, 33(2):249–262, 1985.

[Karpinski and Schudy, 2010] Karpinski and Schudy. Faster algorithms for feedback arc set tournament, kemeny rank aggregation and betweenness tournament. In *21st ISAAC*, pages 3–14, 2010.

[Kim and Williams, 2015] Kim and Williams. Fixing tournaments for kings, chokers, and more. In *24th IJCAI*, pages 561–567, 2015.

[Lang *et al.*, 2012] Lang, Pini, Rossi, Salvagnin, Venable, and Walsh. Winner determination in voting trees with incomplete preferences and weighted votes. *Autonomous Agents and Multi-Agent Systems*, 25(1):130–157, 2012.

[Laslier, 1997] Laslier. Tournament solutions and majority voting. *Springer-Verlag*, 1997.

[Michael *et al.*, 2016] Michael, Suksompong, and Williams. Who can win a single-elimination tournament? In *30th AAAI*, pages 516–522, 2016.

[Ramanujan and Szeider, 2017] Ramanujan and Szeider. Rigging nearly acyclic tournaments is fixed-parameter tractable. In *31st AAAI*, pages 3929–3935, 2017.

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

[Russell and Beek, 2011] Russell and Beek. An empirical study of seeding manipulations and their prevention. In *22nd IJCAI*, pages 350–356, 2011.

[Stanton and Williams, 2011a] Stanton and Williams. Manipulating single-elimination tournaments in the braverman-mossel model. In *IJCAI Workshop on Social Choice and Artificial Intelligence*, 2011.

[Stanton and Williams, 2011b] Stanton and Williams. Manipulating stochastically generated single-elimination tournaments for nearly all players. In *7th WINE*, pages 326–337, 2011.

[Stanton and Williams, 2011c] Stanton and Williams. Rigging tournament brackets for weaker players. In *22nd IJCAI*, pages 357–364, 2011.

[Tullock, 1980] Tullock. Toward a theory of the rent-seeking society. *Texas A&M University Press*, 1980.

[Vu *et al.*, 2009] Vu, Altman, and Shoham. On the complexity of schedule control problems for knockout tournaments. In *8th AAMAS*, pages 225–232, 2009.

[Williams, 2010] Williams. Fixing a tournament. In *24th AAAI*, 2010.

[Williams, 2016] Williams. Knockout tournaments. In *Handbook of Computational Social Choice*, pages 453–474. 2016.