On Succinct Encodings for the Tournament Fixing Problem

Sushmita Gupta[1] , **Saket Saurabh**[2,3] , **Ramanujan Sridharan**[4] and **Meirav Zehavi**[5]
[1]National Institute of Science Education and Research (NISER), India
[2]The Institute of Mathematical Sciences, HBNI, Chennai, India
[3]University of Bergen, Norway
[4]University of Warwick, UK
[5]Ben Gurion University of the Negev, Israel
sushmitagupta@niser.res.in, r.maadapuzhi-sridharan@warwick.ac.uk, saket@imsc.res.in, zehavimeirav@gmail.com

Abstract

Single-elimination tournaments are a popular format in competitive environments. The Tournament Fixing Problem (TFP), which is the problem of finding a seeding of the players such that a certain player wins the resulting tournament, is known to be NP-hard in general and fixed-parameter tractable when parameterized by the feedback arc set number of the input tournament (an oriented complete graph) of expected wins/loses. However, the existence of polynomial kernelizations (efficient preprocessing) for TFP has remained open. In this paper, we present the first polynomial kernelization for TFP parameterized by the feedback arc set number of the input tournament. We achieve this by providing a polynomial-time routine that computes a SAT encoding where the number of clauses is bounded polynomially in the feedback arc set number.

1 Introduction

Single-elimination (or knockout) tournaments are natural models of competitive environments such as sports competitions, decision making and elections. Consequently, they have attracted a great deal of attention in Artificial Intelligence, especially within social choice theory [Laffond *et al.*, 1993; Fisher and Ryan, 1995; Brandt *et al.*, 2016; Williams, 2016; Brandt *et al.*, 2018].

Formally, the execution of a knockout tournament with a set N of n players is described by a complete (unordered) binary tree T with n leaves and a mapping $\varphi : N \to \mathsf{leaves}(T)$, called a *seeding*. In the first round, pairs of 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 then declared the winner.

More recent years have seen an increased interest in the structure of knockout tournaments which might be amenable to manipulation by an external agent who may, either bribe certain participants to throw games ([Russell and Walsh, 2009; Kim and Vassilevska Williams, 2015; Mattei *et al.*, 2015; Gupta *et al.*, 2018b]) or rearrange the initial seeding (rigging) [Vu *et al.*, 2009; Vassilevska Williams, 2010; Stanton and Vassilevska Williams, 2011; Aziz *et al.*, 2014; Kim and Vassilevska Williams, 2015; Kim *et al.*, 2016] in order to ensure that a certain player wins the tournament. In this paper, we focus on the latter.

The most natural computational problem associated with rigging tournaments is the TOURNAMENT FIXING PROBLEM (TFP) where, we are given a tournament (an oriented complete graph) $D = (N, A)$. For every $u, v \in N$, we assume (or predict) that in a match between u and v, u would beat v if and only if $(u, v) \in A$. For a given player $v^\star \in N$, the objective is to decide whether there is a seeding of the n players such that $v^\star$ is the winner of the resulting knockout tournament, given the match outcomes encoded by D. [Aziz *et al.*, 2014] showed that TFP is NP-hard in general. Moreover, motivated by the fact that TFP is easily solved on *acyclic* tournaments, they initiated the study of the complexity of TFP parameterized by the *feedback arc set number* of the tournament, which is the least number of arcs in the given tournament that need to be flipped to obtain an acyclic (or transitive) tournament.

[Aziz *et al.*, 2014] gave a clever $n^{f(k)}$-time dynamic programming algorithm (for some function f) for this problem, where k is the feedback arc set number of the tournament. Their algorithm clearly runs in polynomial time for every fixed value of k. Subsequently, [Ramanujan and Szeider, 2017] gave a $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$-time algorithm for this problem, which implies polynomial-time solvability as long as $k^2 \log k = \mathcal{O}(\log n)$. This bound was recently improved by [Gupta *et al.*, 2018a], who gave a $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$-time algorithm for this problem.

However, in spite of these recent developments on solving TFP, the feasibility of efficient preprocessing for this problem remained an intriguing open question. The framework of kernelization [Downey and Fellows, 2013; Cygan *et al.*, 2015] offers a mathematically rigorous way of designing, analysing and comparing preprocessing algorithms for numerous problems and has been a resounding success over the last two decades. The notion of efficient preprocessing is captured in this framework by the idea of a *polynomial kernelization*, which is a polynomial-time algorithm that returns an instance

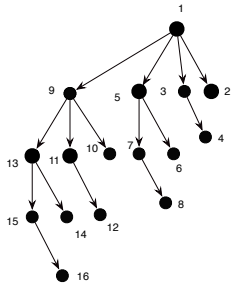

Figure 1: A binomial arborescence resulting from a tournament played by 16 players where the initial seeding pairs Player $2i+1$ against Player $2i+2$ for each $i \in \{0, \ldots, 7\}$.

equivalent to the input such that the size of the output is bounded by a polynomial function of some parameter of the input instance.

Our Contribution. In this paper, we present the first polynomial kernelization for TFP parameterized by the feedback arc set number of the input tournament. At the heart of our result is a complex polynomial-time routine that outputs an equivalent SAT encoding of a given input instance of TFP, such that the number of clauses and variables are bounded polynomially in the feedback arc set number and therefore *independent* of the size of the original tournament. Pipelining our SAT encoding with a chain of known polynomial-time-mapping reductions from SAT to TFP, we obtain our polynomial kernel for TFP.

2 Preliminaries

For a digraph D, a *feedback arc set* is a set of arcs whose reversal in D results in a directed acyclic graph (DAG). An arborescence is a rooted directed tree such that all arcs are directed away from the root.

Definition 1 ([Vassilevska Williams, 2010])**.** *The set of binomial arborescences over a tournament D is recursively defined as follows.* (i) *Each $a \in V(D)$ is a binomial arborescence rooted at a.* (ii) *If, for some $i > 0$, T_a and T_b are 2^{i-1}-node binomial arborescences rooted at a and b, respectively, then the tree T resulting from adding an arc from a to b is the 2^i-node binomial arborescence (b.a) rooted at a. If a binomial arborescence T is such that $V(T) = V(D)$, then T is a* spanning binomial arborescence *(s.b.a.) of D.*

We will often refer to a binomial arborescence simply as a b.a and spanning binomial arborescence as an s.b.a.

Proposition 1 ([Vassilevska Williams, 2010])**.** *Let D be a tournament and let $v^\star \in V(D)$. Then, there is a seeding of $V(D)$ such that the resulting knockout tournament is won by $v^\star$ if and only if D has an* s.b.a *rooted at $v^\star$.*

Note that if T is an s.b.a of D, then $\forall v \in V(T)$, $\exists i \in [\log n] \cup \{0\}$ such that v has 2^i descendants in T and v is the winner of a subtournament played by the descendants of v in T. We use the terms vertex and player interchangeably.

Definition 2. *Let D be a tournament with a special player $v^\star$. Let $F \subseteq A(D)$ be a feedback arc set of D, and let $D^\dagger$ the DAG obtained by reversing the arcs of F in D. Let $\pi : V(D) \to [n]$ be the linear ordering of the players in decreasing order of strength: a player u appears before player v if and only if $(u, v) \in A(D^\dagger)$. Then, we say that π* witnesses *F. We call the vertices in $\{v^\star\} \cup V(F)$* affected *vertices and denote this set by $\mathbf{A}_F$.*

We say that a permutation γ of $X \subseteq V(D)$ respects the permutation π witnessing F if the vertices of X appear in the same order (relative to each other) in γ as they do in π.

We now consider the 'type' function that partitions the vertices in $V(D) \setminus \mathbf{A}_F$ into at most $|\mathbf{A}_F| + 1$ partitions. We first define the set $\mathsf{Types} = [|\mathbf{A}_F| + 1] \cup \{\mathbf{A}_F\}$.

Definition 3. *Let D be a tournament with a special player $v^\star$. Let $F \subseteq A(D)$ be a feedback arc set of D, let π be the ordering witnessing F, and let γ be the ordering of $\mathbf{A}_F$ that respects π. Define the* type function *$\tau^\pi_\gamma : V(D) \to \mathsf{Types}$ as follows. For any $v \in \mathbf{A}_F$, $\tau^\pi_\gamma(v) = v$. For any $v \in V(D) \setminus \mathbf{A}_F$, $\tau^\pi_\gamma(v) = i$ where i is the smallest index in $[|\mathbf{A}_F|]$ such that $\pi(v) < \pi(\gamma(i))$. If no such index exists, that is, $\pi(v) > \pi(\gamma(|\mathbf{A}_F|))$, then $\tau^\pi_\gamma(v) = |\mathbf{A}_F|+1$. For any $i \in \mathsf{Types}$, denote the set of vertices in the pre-image of i under τ^π_γ by $\mathcal{P}_i(\tau^\pi_\gamma)$.*

Observe that vertices of the same type have the same behaviour with respect to every vertex *of a different type* in the graph. Henceforth, we assume without loss of generality that instances of TFP are of the form $(D, k, v^\star, F, \pi)$, where F is a given feedback arc set of D of size $\mathcal{O}(k)$ and π is the ordering witnessing F.

In a parameterized problem, inputs are tuples (I, k) where $I \in \Sigma^*$ is the *problem instance* and $k \in \mathbb{N}$ is called the *parameter*. Let $\Pi \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem and g be a computable function. We say that Π *admits a kernel of size g* if there exists an algorithm referred to as a *kernelization* (or a *kernel*) that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs in time polynomial in $|x| + k$, a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that (a) $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$, and (b) $\max\{|x'|, k'\} \leq g(k)$. If $g(k) = k^{\mathcal{O}(1)}$, then we say that Π *admits a polynomial kernel*.

A *polynomial compression* of a parameterized language $Q \subseteq \Sigma^* \times \mathbb{N}$ into an unparameterized language $R \subseteq \Sigma^*$ is an algorithm that takes as input an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, works in time polynomial in $|x| + k$, and returns a string $y \in \Sigma^*$ such that: (a) $|y| \leq p(k)$ for some polynomial $p(\cdot)$, and (b) $y \in R$ if and only if $(x, k) \in Q$.

3 Polynomial Compressions

Due to the $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ algorithm of Gupta *et al.* [2018a], we may assume henceforth that $k \log k \geq \log n$. Indeed, if this were not the case, then their algorithm would already be a polynomial-time algorithm for TFP.

Let $\widehat{D}_F$ denote the subtournament induced on $\mathbf{A}_F$. Moreover, for each $i \in \mathsf{Types}$, we denote by $\mu(i)$ the number of vertices of type i. Given $D, k, v^\star, F, \pi$, we denote by $\omega(D, k, v^\star, F, \pi)$ the combined bit-string encoding of $\widehat{D}_F, k, v^\star, \gamma$ (the relative ordering of $\mathbf{A}_F$ in π), and $\mu(i)$ for every $i \in \mathsf{Types}$. Since $\widehat{D}_F$ has $\mathcal{O}(k)$ vertices and the bit-size of $\mu(i)$ is bounded by $\log n \leq k \log k$ for every $i \in \mathsf{Types}$,

it follows that $\omega(D, F, \pi) = \mathcal{O}(k^2 \log k)$. Moreover, observe that given $D, k, v^\star, F, \pi$ and F, the string $\omega(D, k, v^\star, F, \pi)$ can be computed in polynomial time.

Define the language $Q \subseteq \{0,1\}^*$ to be the set $\{\omega(D, k, v^\star, F, \pi) \mid (D, k, v^\star, F)$ is a yes-instance of TFP$\}$. Since given $\omega(D, k, v^\star, F, \pi)$, the tournament D is uniquely defined, we have obtained a polynomial compression of TFP into the language Q. However, note that the most natural certificate of membership in Q is a seeding of the players resulting in a knockout tournament won by $v^\star$ (equivalently, an s.b.a contained in D and rooted at $v^\star$) and has size ($|V(D)|$) that, in general, could be superpolynomial in the length of the input ($\mathcal{O}(k^2 \log k)$). Hence, it is not clear that we have a polynomial test for membership of Q. We now proceed to build on techniques from [Ramanujan and Szeider, 2017] and [Gupta *et al.*, 2018a] to obtain a different compression for this problem, which forms the crux of our SAT encoding.

3.1 Packing-Based Variant

In this subsection, we interpret the work of Gupta *et al.* [2018a] in terms of a packing variant.

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

Observation 1. *Let T be an arborescence and $M \subseteq V(T)$. Then, $|\mathsf{LCA}(M)| \leq 2|M|$.*

We define a refinement of the notion of a *template tree* of [Ramanujan and Szeider, 2017] and [Gupta *et al.*, 2018a] as follows. Let T be a b.a rooted at $v^\star$ and let $X \subseteq V(T)$ such that $v^\star \in X$. Let $Z = \mathsf{LCA}(X)$. Let T' be a tree constructed as follows. We begin with T and delete all nodes which do not lie on a $v^\star$-u path in T for any $u \in Z$. For every $x, y \in Z$ such that x is an ancestor of y in T' and there is no other vertex of Z which is an internal node of the x-y path in T', we add to Z the unique child of x which lies on this x-y path in T'. (In [Ramanujan and Szeider, 2017], an *arbitrary* internal node (if one exists) was added to Z, and in [Gupta *et al.*, 2018a], no internal node was added.) We then repeatedly short-circuit each node of degree 2 in T' which is not in Z. That is, we repeat the following step as long as possible: pick a vertex $u \in V(T) \setminus Z$ with exactly one in-neighbor u^- and one out-neighbor u^+; delete u and add the arc (u^-, u^+). Let T'' be the tree that remains when the above step can no longer be performed. We call T'' the *template tree* of X in T.

Observe that the vertex set of the template tree T'' of X in the above definition is the set Z, whose size at most $2|\mathsf{LCA}(X)| \leq 4|X|$ (by Observation 1), and it contains the set X. Moreover, the leaves of T'' are contained in X.

Definition 4. *A tuple (L, ψ, χ, ℓ) is* valid *for an instance $(D, k, v^\star, F)$ if the following conditions hold.*

1. *L is an arborescence on at most $4(2k+1)$ nodes rooted at $v^\star$ such that $V(L) \supseteq \mathbf{A}_F$ and $\mathsf{leaves}(L) \subseteq \mathbf{A}_F$,*
2. *$\psi : V(L) \to$ Types such that $\psi(u) = u$ for every $u \in \mathbf{A}_F$, and $\psi(u) \in$ Types $\setminus \mathbf{A}_F$ for every $u \in V(L) \setminus \mathbf{A}_F$,*
3. *$\chi : V(L) \to [\log n] \cup \{0\}$ where $\chi(v^\star) = \log n$,*
4. *$\ell : A(L) \to [\log n] \cup \{0\}$, and*
5. *for each $(u, v) \in A(L)$ with $\ell(u, v) = 0$, either $\psi(u) = \psi(v)$ or vertices of type $\psi(u)$ beat vertices of type $\psi(v)$.*

We relate the definition above with binomial arborescence via the notion of a template tree as follows.

Definition 5. *A tuple (L, ψ, χ, ℓ) is* realizable *for an instance $(D, k, v^\star, F)$ if it is valid and there is an n-node b.a T rooted at $v^\star$ with $V(L) \subseteq V(T)$ such that the following hold: (i) L is the template tree of $\mathbf{A}_F$ in T, (ii) for each $u \in V(L)$, the number of nodes in the subarborescence of T rooted at u is $2^{\chi(u)}$, and (iii) for each $(u, v) \in A(L)$, the number of internal nodes on the path from u to v in T is $\ell(u, v)$.*

In the above definition, T is said to *witness* the realizability of this valid tuple. With each valid tuple (L, ψ, χ, ℓ), we associate a "packing condition". Specifically, we say that (L, ψ, χ, ℓ) is *packable* if there exists a partition $\mathcal{P}$ of $V(D)$ and a mapping $f : V(L) \cup A(L) \to \mathcal{P}$ that satisfies the following constraints.

1. For each $u \in V(L)$, we have *(i)* $|f(u)| = 2^{\chi(u)} - \sum_{v \in \mathsf{Child}_L(u)} (2^{\chi(v)} + \ell(u, v))$, *(ii)* for each $v \in f(u)$, v is beaten by some vertex of type $\psi(u)$, and *(iii)* there exists a vertex of type $\psi(u)$ in $f(u)$.
2. For each $(u, v) \in A(L)$, we have *(i)* $|f(u, v)| = \ell(u, v)$, and *(ii)* for each $w \in f(u, v)$, w is beaten by some vertex of type $\psi(u)$ but beats some vertex of type $\psi(v)$.

The following proposition follows (though not stated explicitly) from the work of Gupta *et al.* [2018a],

Proposition 2. *A given instance of* TFP *is a yes-instance if and only if some realizable tuple (L, ψ, χ, ℓ) is packable.*

We define the language $W \subseteq \{0,1\}^*$ to be the set $\{\omega(D, k, v^\star, F, \pi) \mid (D, k, v^\star, F, \pi)$ has a realizable tuple (L, ψ, χ, ℓ) that is packable$\}$. Proposition 2 then implies a polynomial compression of TFP into the language W. However, W is also not easily seen to be in NP as the natural certificate consists of a tuple (L, ψ, χ, ℓ) and a partition $\mathcal{P}$; while it is not difficult to substitute every set in $\mathcal{P}$ with a function that only states how many vertices of each type belong to that set and use that as a witness (we will do this later), it is not clear how to verify that (L, ψ, χ, ℓ) is realizable—in particular, the dynamic programming algorithm of Gupta *et al.* [2018a] for this task takes time polynomial in $|V(D)|$ that, in general, could be superpolynomial in the input length ($\mathcal{O}(k^2 \log k)$).

We next give an explicit reduction from Q to CNF-SAT. (In turn, this reduction implies that Q and W are in NP.)

4 The CNF-SAT Encoding

Due to lack of space, we describe each subformula that will be part of our final output CNF-SAT formula only in words. However, we present our descriptions in such a way that it is straightforward to write it as an AND of clauses. Moreover, we use intuitive terminology for the subformulas as often as possible in order to facilitate readability.

The main idea in our reduction is to simulate the "guess-and-verify" operation for instances of W as a test for the satisfiability of an appropriate CNF formula. Towards that, let

$\mathsf{Dummy} = \{d_1, \ldots, d_\ell\}$ be a set of $\ell = 4(2k+1) - |\mathbf{A}_F|$ "dummy" nodes. Intuitively, the dummies are placeholders for the (unknown) vertices outside $\mathbf{A}_F$ that either *(i)* belong to in $\mathsf{LCA}(\mathbf{A}_F)$ in the (unknown) template tree L, or *(ii)* are the children of these vertices. We distinguish between the two types of dummies via the following lemma, which is a direct consequence of the definition of template trees.

Lemma 1. *Let (L, ψ, χ, ℓ) be a realizable tuple for an instance $(D, k, v^\star, F)$. For each $u \in V(L) \setminus \mathbf{A}_F$, we have $u \in \mathsf{LCA}(\mathbf{A}_F)$ in any* s.b.a *that witnesses the realizability of this tuple if and only if u has at least two children in L.*

We require the following set of variables.

- $\forall$ two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$ where $v \neq v^\star$, we have a variable $x^L_{u \to v}$ to indicate that u is the parent of v in the (unknown) template tree L. (Only dummies assigned a parent will be considered as part of the tree.) We exclude $v^\star$ to assert that it is the root of L.
- $\forall$ dummy $u \in \mathsf{Dummy}$ and type $i \in \mathsf{Types} \setminus \mathbf{A}_F$, we have a variable $x^\psi_{u \Leftarrow i}$ to indicate that ψ assigns the type i to u.
- $\forall u \in \mathbf{A}_F \cup \mathsf{Dummy}$ and "bit-index" $b \in [\log n] \cup \{0\}$, we have a variable $x^{\chi,b}_u$ to indicate whether the b^{th} bit (where the least significant one is indexed by 0) of $2^{\chi(u)}$ is 1.
- $\forall$ distinct $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$ where $v \neq v^\star$ and "bit-index" $b \in [\log\log n] \cup \{0\}$, we have a variable $x^{\ell,b}_{u \to v}$ to indicate whether the b^{th} bit of $\ell(u, v)$ is 1.
- $\forall$ element $u \in (\mathbf{A}_F \cup \mathsf{Dummy}) \setminus \{v^\star\}$ and "distance" $d \in [\log n]$, we have a variable $x^{\mathsf{dist},d}_u$ to indicate that there is a path of length d from $v^\star$ to u in L. (These variables are necessary for a compact encoding of a formula that will be given to ensure the validity of (L, ψ, χ, ℓ).)
- $\forall e \in \mathbf{A}_F \cup \mathsf{Dummy} \cup ((\mathbf{A}_F \cup \mathsf{Dummy}) \times (\mathbf{A}_F \cup \mathsf{Dummy}))$ (i.e., e is a single element or a pair of elements from $\mathbf{A}_F \cup \mathsf{Dummy}$), type $i \in \mathsf{Types} \setminus \mathbf{A}_F$ and "bit-index" $b \in [\log n] \cup \{0\}$, we have a variable $x^{g,b}_{e,i}$ to indicate whether the b^{th} bit of $g(e, t)$ is 1. Here, g is the witness of the satisfaction of the implicit packing condition which will be defined in Section 4.2.
- Additional "local variables" (where a local variable is one used in a single specific formula for implementation purposes) will be presented when required.

4.1 Constraints for the Realizability of (L, ψ, χ, ℓ)

We first present constraints that assert that (L, ψ, χ, ℓ) is valid. To assert that L is an arborescence, the following lemma will come in handy.

Lemma 2. *Let L be a digraph where $v^\star \in V(L)$ has in-degree 0. Then, L is an arborescence rooted at $v^\star$ of height at most $\log n$ if and only if the following conditions hold.*

- *Each vertex $u \in V(L) \setminus \{v^\star\}$ has in-degree at most 1.*
- *For each $u \in V(L) \setminus \{v^\star\}$, there is $i \in [\log n]$ such that L has a path of length i from $v^\star$ to u.*

Lemma 2 is based on the fact that L is an arborescence rooted at $v^\star$ if and only if every vertex other than $v^\star$ has in-degree exactly 1 and is reachable from $v^\star$. The same fact helps us identify those dummies which participate in L and to ensure that $V(L) \supseteq \mathbf{A}_F$. Towards this, the following definition and lemma will be useful.

Definition 6. *Let L' be a digraph where $v^\star \in V(L)$ has in-degree 0. Then, the* reachability digraph *of $v^\star$ in L' is the subgraph of L' induced by the set of $v^\star$ and the other vertices $u \in V(L')$ such that L' has a path of length i from $v^\star$ to u for some $i \in [\log n]$.*

Lemma 3. *Let L be an arborescence rooted at $v^\star$ of height at most $\log n$. Then, there exists a digraph L' where every vertex $u \in V(L') \setminus \{v^\star\}$ has in-degree at most 1, the reachability digraph of $v^\star$ in L' is precisely L, and every leaf in L has out-degree 0 in L'.*

We now present the constraints.

1. For every $u \in (\mathbf{A}_F \cup \mathsf{Dummy}) \setminus \{v^\star\}$, the formula $\mathsf{AtMostOneParent}(u)$ is true if and only if at most one variable in $\{x^L_{v \to u} : v \in (\mathbf{A}_F \cup \mathsf{Dummy}) \setminus \{u\}\}$ is true.
2. For every $u \in \mathsf{Dummy}$, the formula $\mathsf{ExactlyOneType}(u)$ is true if and only if exactly one variable in $\{x^\psi_{u \Leftarrow i} : i \in \mathsf{Types} \setminus \mathbf{A}_F\}$ is true.
3. For every $u \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formula $\mathsf{ExactlyOneSizeBit}(u)$ is true if and only if exactly one variable in $\{x^{\chi,b}_u : b \in [\log n] \cup \{0\}\}$ is true. Moreover, $x^{\chi,\log n}_{v^\star}$ is true.
4. For every $u \in (\mathbf{A}_F \cup \mathsf{Dummy}) \setminus \{v^\star\}$, the formula $\mathsf{AtMostOneDist}(u)$ is true if and only if at most one variable in $\{x^{\mathsf{dist},d}_u : d \in [\log n]\}$ is true.
5. For every $u \in \mathbf{A}_F \setminus \{v^\star\}$, the formula $\mathsf{AtLeastOneDist}(u)$ is true if and only if there exists $d \in [\log n]$ such that $x^{\mathsf{dist},d}_u$ is true. We note that this set of constraints will ensure that $V(L) \supseteq \mathbf{A}_F$.
6. For every $u \in \mathbf{A}_F \setminus \{v^\star\}$ and $d \in [\log n]$, the formula $\mathsf{VerifyDist}_d(u)$ is true if and only if $x^{\mathsf{dist},d}_u$ is true and either *(i)* $d = 1$ and $x^L_{v^\star \to u}$ is true, or *(ii)* $d \geq 2$ and there is a $w \in (\mathbf{A}_F \cup \mathsf{Dummy}) \setminus \{v^\star, u\}$ such that both $x^L_{w \to u}$ and $x^{\mathsf{dist},d-1}_w$ are true.
7. For each $u \in \mathsf{Dummy}$, the formula $\mathsf{Reachable}(u)$ is true if and only if there is a $d \in [\log n]$ such that $x^{\mathsf{dist},d}_u$ is true.
8. For every $u \in \mathsf{Dummy}$, the formula $\mathsf{NotLeaf}(u)$ is true if and only if $\mathsf{Reachable}(u)$ is false or there exists $w \in (\mathbf{A}_F \cup \mathsf{Dummy}) \setminus \{v^\star, u\}$ such that $x^L_{u \to w}$ is true. This set of constraints will be used to ensure that $\mathsf{leaves}(L) \subseteq \mathbf{A}_F$.

We now introduce a few formulas that will be used several times. First, we identify the nodes and arcs of L as follows. For every $u \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formula $\mathsf{IsNode}(u)$ is true if and only if either $u \in \mathbf{A}_F$ or both $u \in \mathsf{Dummy}$ and $\mathsf{Reachable}(u)$ is true. Moreover, for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formula $\mathsf{IsArc}(u, v)$ is true if and only if $v \neq v^\star$, $\mathsf{IsNode}(u)$ is true, $\mathsf{IsNode}(v)$ is true, and $x^L_{u \to v}$ is true. Additionally, for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formula $\mathsf{Beats}(u, v)$ is true if and only if there exist $i, j \in \mathsf{Types}$ such that the following conditions hold: *(i)* $i = j$ or vertices of type i beat vertices of type j; *(ii)* either $u = i \in \mathbf{A}_F$ or both $u \notin \mathbf{A}_F$ and $x^\psi_{u \Leftarrow i}$ is true; *(iii)* either $v = j \in \mathbf{A}_F$ or both $v \notin \mathbf{A}_F$ and $x^\psi_{v \Leftarrow j}$ is true.

To validate Condition 5 in Definition 4, we ensure that for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formula $\mathsf{IfArcThenValid}(u, v)$ is true if and only if, if $\mathsf{IsArc}(u, v)$ is true, then $\mathsf{Beats}(u, v)$ is true.

LCA-Closure. Based on Lemma 1, we determine whether a dummy belongs to the LCA-closure as follows. For every $u \in \mathsf{Dummy}$, we have a formula $\mathsf{InClosure}(u)$ that is true if and only if $\mathsf{IsNode}(u)$ is true and there exist two distinct $q, p \in (\mathbf{A}_F \cup \mathsf{Dummy}) \setminus \{v^\star, u\}$ such that both $x^L_{u\to p}$ and $x^L_{u\to q}$ are true. Moreover, for every $u \in \mathsf{Dummy}$, we have a formula $\mathsf{ChildOfClosure}(u)$ that is true if and only if $\mathsf{IsNode}(u)$ is true and $\mathsf{InClosure}(u)$ is false. To verify that every degree-2 dummy is indeed a child of a node in the LCA-closure, for every two distinct $u, v \in \mathsf{Dummy}$, we have a formula $\mathsf{NoDeg2ConsecDum}(u, v)$ that is true if and only if, if both $\mathsf{IsArc}(u, v)$ is true and $\mathsf{ChildOfClosure}(u)$ is true, then $\mathsf{ChildOfClosure}(v)$ is false.

A crucial ingredient of our encoding is a "local check" to verify whether the size logarithms assigned by χ enable the existence of a b.a that gives rise to L as per Definition 5. (Not all sizes enable this—for example, if $n = 16$, L is a star with 6 nodes, and each leaf is assigned size 2, the tuple is not realizable; indeed, $v^\star$ has only 4 incomparable children whose subtrees have size 2, see Fig. 1). This ingredient is encapsulated in Lemma 5. (The first item has already been validated.) Towards its presentation and proof, the following lemma will be useful.

Lemma 4. *1. Let T be a b.a, and let $u \in V(T)$. Then, the subarborescence rooted at u is a b.a.*
2. For any set of nodes U, $r \in U$ and $\chi : U \to [\log n] \cup \{0\}$, there exists a b.a rooted at r where each node in $U \setminus \{r\}$ is a child of r and for each $u \in U$ the size of the subarborescence rooted at u is $2^{\chi(u)}$ if and only if χ is injective and $\chi(r) > \chi(u)$ for each $u \in U$.
3. For any two nodes r and u, $\chi : \{r, u\} \to [\log n] \cup \{0\}$ and $\ell \in [\log n] \cup \{0\}$, there exists a b.a rooted at r with a path of ℓ internal nodes from r to u and where for each $v \in \{r, u\}$ the size of the subarborescence rooted at v is $2^{\chi(v)}$ if and only if $2^{\chi(r)} \geq 2^{\chi(u)+\ell+1}$.

Proof. (1) and (2) follow directly from Definition 1, while (3) can be proved by fixing r, u and $\chi : \{r, u\} \to [\log n] \cup \{0\}$ and performing a straightforward induction on ℓ. □

Based on Lemmas 1 and 4, we have the following useful characterization of realizable tuples.

Lemma 5. *Let (L, ψ, χ, ℓ) be a valid tuple for an instance $(D, k, v^\star, F)$. Then, (L, ψ, χ, ℓ) is realizable if and only if the following conditions are satisfied.*

1. There does not exist an arc $(u, v) \in A(L)$ such that $u, v \notin \mathbf{A}_F$ and both u and v have degree 2 in L.
2. For every arc $(u, v) \in A(L)$ such that $u \in \mathbf{A}_F$ or u has degree at least 3 in L, we have $\ell(u, v) = 0$.
3. For every arc $(u, v) \in A(L)$, $\chi(u) > \chi(v)$.
4. For every $u \in V(L)$ of degree at least 3 and any two distinct children v, w of u in L, we have $\chi(v) \neq \chi(w)$.
5. For every arc $(u, v) \in A(L)$ such that $u \notin \mathbf{A}_F$ and v is the only child of u in L, we have $2^{\chi(u)} \geq 2^{\chi(v)+\ell(u,v)+1}$.

Proof. ($\implies$) (1) follows from Lemma 1, (2) follows from the definition of template trees and realizability, (3) follows from Lemma 4 (2), (4) follows from the second statement of this lemma and Definition 1, and (5) follows from by Lemma 4 (3).

($\impliedby$) For every leaf u of L, we root a $2^{\chi(u)}$-node b.a at u. For every $(u, v) \in A(L)$, we subdivide it exactly $\ell(u, v)$ times to obtain an arborescence L' of height $\log n$ rooted at $v^\star$ which, (using the 5 properties in the premise) can be seen to be a subgraph of an s.b.a. Therefore, we use $\chi(u)$ for every $u \in V(L)$ to root (or reconstruct) appropriately sized b.a's at every node of $V(L)$ and for each $(u, v) \in A(L)$, use $\ell(u, v)$ and the height of v in L' to reconstruct the b.a's rooted at the newly created subdivision vertices. It is easy to prove that assuming the given 5 properties, this is always possible and moreover, the resulting s.b.a T rooted at $v^\star$ indeed witnesses the realizability of the given tuple. □

For Condition 2 in Lemma 5, for each distinct $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formula $\mathsf{LocCheckLenSensible}(u, v)$ is true if and only if, if both *(i)* $\mathsf{IsArc}(u, v)$ is true, and *(ii)* either $u \in \mathbf{A}_F$ or $u \in \mathsf{Dummy}$ and $\mathsf{InClosure}(u)$ is true, then for every $b \in [\log \log n] \cup \{0\}$, $x^{\ell,b}_{u\to v}$ is false.

To validate Condition 3 in Lemma 5, for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formula $\mathsf{LocCheckSizeDec}(u, v)$ is true if and only if, if $\mathsf{IsArc}(u, v)$ is true, then the largest $b \in [\log n] \cup \{0\}$ such that $x^{\chi,b}_u$ and $x^{\chi,b}_v$ are either not both true or not both false exists and is such that $x^{\chi,b}_u$ is true and $x^{\chi,b}_v$ is false.

To validate Condition 4 in Lemma 5, for every three distinct elements $u, v, w \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formula $\mathsf{LocCheckSizeDiff}(u, v, w)$ if true if and only if, if both $\mathsf{IsArc}(u, v)$ and $\mathsf{IsArc}(u, w)$ are true, then there exists $b \in [\log n] \cup \{0\}$ such that either both $x^{\chi,b}_v$ is true and $x^{\chi,b}_w$ is false or both $x^{\chi,b}_v$ is false and $x^{\chi,b}_w$ is true.

To validate Condition 5 in Lemma 5, we need to implement standard binary addition. For this, we introduce local variables. Specifically, for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$ and $b \in [\log n] \cup \{0\}$, we have a new variable $y^{\mathsf{LocCheckPath}}_{(u,v),b}$ to indicate whether the b^{th} bit in the binary encoding of the (nonnegative) integer

$$\sum_{i=0}^{\log n} [x^{\chi,i}_v] \cdot 2^i + 2^{\sum_{i=0}^{\log \log n} [x^{\ell,i}_{u\to v}] \cdot 2^i},$$

where we use $[z]$ (for any variable z) to denote 1 if z is true and 0 otherwise. Towards this, for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$ and $b \in [\log] \cup \{0\}$, we have a new variable $z^{\mathsf{LocCheckPath}}_{(u,v),b}$ to indicate whether the b^{th} bit in the binary encoding of the (nonnegative) integer $2^{\sum_{i=0}^{\log \log n} [x^{\ell,i}_{u\to v}] \cdot 2^i}$. Then, for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$, for every function $f : \{[\log \log n] \cup \{0\}\} \to \{0, 1\}$ (because there are only $\mathcal{O}(\log n)$ such functions, the number of formulas to consider is $k^{\mathcal{O}(1)}$), we have a formula that is satisfied if and only if, if for every $i \in [\log \log n] \cup \{0\}$ we have that $f(i) = 1$ if and only if $x^{\ell,i}_{u\to v}$ is true, then for every $b \in [\log n] \cup \{0\}$ we have that $z^{\mathsf{LocCheckPath}}_{(u,v),b}$ is true if and only if $b = \sum_{i=0}^{\log \log n} f(i) \cdot 2^i - 1$.

To validate the "y-variables", for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$ and $b \in [\log n - 1] \cup \{0\}$, we also have a new variable $c^{\mathsf{LocCheckPath}}_{(u,v),b}$ and a formula that is satisfied if and only either *(i)* $b = 0$ and both $x^{\chi,0}_v$

and $z^{\mathsf{LocCheckPath}}_{(u,v),0}$ are true, or *(ii)* $b \geq 1$ and at least two among $x^{\chi,b}_v$, $z^{\mathsf{LocCheckPath}}_{(u,v),b}$ and $c^{\mathsf{LocCheckPath}}_{(u,v),b-1}$ are true. Note that the "c-variables" are meant to encode carry-overs in addition computations. Now, for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$ and $b \in [\log n] \cup \{0\}$, we have a formula that is satisfied if and only if either *(i)* $b = 0$ and $y^{\mathsf{LocCheckPath}}_{(u,v),0}$ is true if and only if exactly one variable among $x^{\chi,0}_v$ and $z^{\mathsf{LocCheckPath}}_{(u,v),0}$ is true, or *(ii)* $b \geq 1$ and $y^{\mathsf{LocCheckPath}}_{(u,v),b}$ is true if and only if exactly one variable or exactly three variables among the variables $x^{\chi,b}_v$, $z^{\mathsf{LocCheckPath}}_{(u,v),b}$ and $c^{\mathsf{LocCheckPath}}_{(u,v),b-1}$ are true.

Lastly, for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formula $\mathsf{LocCheckPath}(u, v)$ is true if and only if, if both $\mathsf{IsArc}(u, v)$ and $\mathsf{ChildOfClosure}(v)$ are true, then either *(i)* for every $b \in [\log n] \cup \{0\}$, the two variables $x^{\chi,b}_u$ and $y^{\mathsf{LocCheckPath}}_{(u,v),b}$ are both true or both false, or *(ii)* the largest $b \in [\log n] \cup \{0\}$ such that $x^{\chi,b}_u$ and $y^{\mathsf{LocCheckPath}}_{(u,v),b}$ are either not both true or not both false is such that $x^{\chi,b}_u$ is true and $y^{\mathsf{LocCheckPath}}_{(u,v),b}$ is false.

The REALIZABLE formula. We define the formula REALIZABLE as the AND of the following formulas: *(i)* every formula in items 1–8 in "Validity"; *(ii)* for every two distinct elements $u, v \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formulas $\mathsf{IfArcThenValid}(u, v)$, $\mathsf{LocCheckLenSensible}(u, v)$, $\mathsf{LocCheckSizeDec}(u, v)$ and $\mathsf{LocCheckPath}(u, v)$; *(iii)* for every three distinct elements $u, v, w \in \mathbf{A}_F \cup \mathsf{Dummy}$, the formula $\mathsf{LocCheckSizeDiff}(u, v, w)$.

4.2 Constraints for the Packing-Based Condition

We rephrase the packing condition as follows. We say that (L, ψ, χ, ℓ) is *implicitly packable* if there exists a function $g : (V(L) \cup A(L)) \times \mathsf{Types} \to [n] \cup \{0\}$ that satisfies:

1. For each $u \in V(L)$, we have *(i)* $|\sum_{t \in \mathsf{Types}} g(u, t)| = 2^{\chi(u)} - \sum_{v \in \mathsf{Child}_L(u)} (2^{\chi(v)} + \ell(u, v))$, *(ii)* for each $t \in \mathsf{Types}$ such that $g(u, t) \geq 1$, either $\psi(u) = t$ or vertices of type t are beaten by vertices of type $\psi(u)$, and *(iii)* $g(u, \psi(u)) \geq 1$.
2. For each $(u, v) \in A(L)$, we have *(i)* $|\sum_{t \in \mathsf{Types}} g((u, v), t)| = \ell(u, v)$, and *(ii)* for each $t \in \mathsf{Types}$ such that $g(u, t) \geq 1$, there exists a vertex of type t that is beaten by some vertex of type $\psi(u)$ but beats some vertex of type $\psi(v)$.
3. For each $t \in \mathsf{Types}$, $|\sum_{e \in V(L) \cup A(L)} g(e, t)| = \mu(t)$.

It can be verified that the following statement holds true. (For lack of space, we omit a proof.)

Lemma 6. *Let (L, ψ, χ, ℓ) be a realizable tuple for an instance $(D, k, v^\star, F)$. Then, (L, ψ, χ, ℓ) is packable if and only if it is implicitly packable.*

Thus, Proposition 2 implies the following lemma.

Lemma 7. *A given instance of* TFP *is a yes-instance if and only if some realizable tuple (L,ψ,χ,ℓ) is implicitly packable.*

By arguments similar to those in the previous subsection, it is not difficult to validate (in CNF-SAT) the satisfaction of each of the three constraints in the definition of the implicit packing condition. For lack of space, we omit these details. The formula to be generated in this subsection is denoted by PACKABLE; we state its properties in the next subsection.

The output of the reduction is $\varphi = \varphi(D, k, v^\star, F) = \mathsf{REALIZABLE} \wedge \mathsf{PACKABLE}$. Then, the following holds.

Observation 2. *The size of the formula φ is polynomial in k, and it is computable in polynomial time.*

We have defined φ in such a way that a satisfying assignment for φ corresponds precisely to the tuple (L, ψ, χ, ℓ, g) where (L, ψ, χ, ℓ) is a realizable tuple and is implicitly packable (using the function g). Conversely, given a realizable tuple (L, ψ, χ, ℓ) and the function g certifying that this tuple is implicitly packable, one can construct a satisfying assignment for φ. This proves our main theorem.

Theorem 1. TFP *admits a polynomial compression into* CNF-SAT.

5 Outline of the Kernelization Algorithm

[Aziz *et al.*, 2014] give a polynomial-time reduction to TFP from the variant of 3-SAT where every literal appears at most twice. Call this reduction $\mathcal{D}$. This variant of 3-SAT can be easily seen to be NP-hard by a simple reduction from 3-SAT where one creates distinct copies of each variable for each of its literals and then finally adds a 2-CNF formula encoding the equality among these copies. Call this reduction from 3-SAT, $\mathcal{E}$. Finally, let $\mathcal{F}$ denote the standard polynomial-time reduction from CNF-SAT to 3-SAT.

Let $\mathcal{Q}$ denote the composed algorithm $\mathcal{D} \circ \mathcal{E} \circ \mathcal{F}$ which takes as input a CNF-SAT instance, runs Algorithm $\mathcal{F}$ on it, then Algorithm $\mathcal{E}$ on the output of $\mathcal{F}$ and finally, Algorithm $\mathcal{D}$ on the output of $\mathcal{E}$. Notice that $\mathcal{Q}$ is a polynomial-time reduction from CNF-SAT to TFP. Executing $\mathcal{Q}$ on our CNF-SAT encoding results in the polynomial kernelization for TFP parameterized by the feedback arc set number.

6 Conclusions and Future Research

Our CNF-SAT encoding for the Tournament Fixing Problem with output size bounded polynomially in the feedback arc set number of the input is pleasantly surprising because TFP is a graph layout problem and designing even fixed-parameter tractable algorithms for such problems is quite challenging, let alone kernelization algorithms. Moreover, combining our SAT encoding with state-of-the-art SAT solvers could potentially lead to new and faster algorithms for the TFP problem. Finally, resolving the parameterized complexity of TFP with respect to stronger parameters such as *feedback vertex set* remains an interesting open problem for future research.

Acknowledgements

The authors wish to thank the anonymous reviewers for their helpful comments and acknowledge support by the European Research Council (ERC) via grant LOPPRE, reference no. 819416, Israel Science Foundation (ISF) grant no. 1176/18 and the Centre for Discrete Mathematics and its Applications, University of Warwick.

References

[Aziz *et al.*, 2014] Haris Aziz, Serge Gaspers, Simon Mackenzie, Nicholas Mattei, Paul Stursberg, and Toby Walsh. Fixing a balanced knockout tournament. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 552–558, 2014.

[Brandt *et al.*, 2016] Felix Brandt, Markus Brill, and Bernhard Harrenstein. Tournament Solutions. In F Brandt, V Conitzer, U Endriss, J Lang, and A. D. Procaccia, editors, *Handbook of Computational Social Choice*, pages 453–474. Cambridge University Press, 2016.

[Brandt *et al.*, 2018] Felix Brandt, Markus Brill, and Paul Harrenstein. Extending tournament solutions. *Social Choice and Welfare*, 51(2):193–222, 2018.

[Cygan *et al.*, 2015] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[Downey and Fellows, 2013] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[Fisher and Ryan, 1995] David Fisher and Jennifer Ryan. Tournament games and positive tournaments. *Journal of Graph Theory*, 19(2):217–236, 1995.

[Gupta *et al.*, 2018a] Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. 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.*, pages 275–281, 2018.

[Gupta *et al.*, 2018b] Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. 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.*, pages 282–288, 2018.

[Kim and Vassilevska Williams, 2015] Michael P. Kim and Virginia Vassilevska Williams. 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*, pages 561–567, 2015.

[Kim *et al.*, 2016] Michael P. Kim, Warut Suksompong, and Virginia Vassilevska Williams. Who can win a single-elimination tournament? In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 516–522, 2016.

[Laffond *et al.*, 1993] G. Laffond, J. F. Laslier, and M. Le Breton. The bipartisan set of a tournament game. *Games and Economic Behavior*, 5:182–201, 1993.

[Mattei *et al.*, 2015] Nicholas Mattei, Judy Goldsmith, Andrew Klapper, and Martin Mundhenk. On the complexity of bribery and manipulation in tournaments with uncertain information. *Journal of Applied Logic*, 13(4, Part 2):557 – 581, 2015. Special JAL Issue dedicated to Uncertain Reasoning at FLAIRS.

[Ramanujan and Szeider, 2017] M. S. Ramanujan and Stefan Szeider. 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.*, pages 3929–3935, 2017.

[Russell and Walsh, 2009] Tyrel Russell and Toby Walsh. Manipulating tournaments in cup and round robin competitions. In Francesca Rossi and Alexis Tsoukias, editors, *Algorithmic Decision Theory*, pages 26–37, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Stanton and Vassilevska Williams, 2011] Isabelle Stanton and Virginia Vassilevska Williams. 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*, pages 357–364, 2011.

[Vassilevska Williams, 2010] Virginia Vassilevska Williams. Fixing a tournament. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, 2010.

[Vu *et al.*, 2009] Thuc Vu, Alon Altman, and Yoav Shoham. 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*, pages 225–232, 2009.

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