An Exercise in Tournament Design: When Some Matches Must Be Scheduled

Sushmita Gupta[1], **Ramanujan Sridharan**[2], **Peter Strulo**[2]

[1]The Institute of Mathematical Sciences, HBNI, India
[2]University of Warwick, UK
sushmita.gupta@gmail.com, r.maadapuzhi-sridharan@warwick.ac.uk, peter.strulo@warwick.ac.uk

Abstract

Single-elimination (SE) tournaments are a popular format used in competitive environments and decision making. Algorithms for SE tournament manipulation have been an active topic of research in recent years. In this paper, we initiate the algorithmic study of a novel variant of SE tournament manipulation that aims to model the fact that certain matchups are highly desired in a sporting context, incentivizing an organizer to manipulate the bracket to make such matchups take place. We obtain both hardness and tractability results. We show that while the problem of computing a bracket enforcing a given set of matches in an SE tournament is NP-hard, there are natural restrictions that lead to polynomial-time solvability. In particular, we show polynomial-time solvability if there is a linear ordering on the ability of players with only a constant number of exceptions where a player with lower ability beats a player with higher ability.

Introduction

There is a rich history of work on the algorithmics of designing Single Elimination (SE) or knockout tournaments as they are a format of competition employed in varied scenarios such as sports, elections and different forms of decision making (Tullock 1980; Horen and Riezman 1985; Rosen 1986; Laslier 1997; Connolly and Rendleman 2011). Based on an initial bracket (a permutation of the players, also called a *seeding*), it proceeds in multiple rounds, culminating in a single winner. In each round, all players that have not yet lost a match are paired up to play the next set of matches. Losers exit and winners proceed to the next round, until only one remains, the winner of the tournament. In general, the tournament designer is assumed to be given probabilities p_{ij} expressing the likelihood that player i beats player j. In this paper, we focus on the deterministic model, i.e., when these probabilities are 0 or 1. This model has already been the subject of numerous papers in the last few years. Besides being independently interesting from a structural and algorithmic perspective as shown by (Vassilevska Williams 2010; Aziz et al. 2014; Ramanujan and Szeider 2017; Gupta et al. 2018, 2019; Manurangsi and Suksompong 2023; Zehavi 2023), the deterministic model naturally captures sequential majority elections along binary trees (Lang et al. 2007; Vu, Altman, and Shoham 2009) where each "match" is a comparison of votes of two candidates and the candidate with more votes wins and moves on.

A major question in the study of SE tournament design is the TOURNAMENT FIXING Problem (TF): Can a designer efficiently find a bracket that maximizes the likelihood (or ensures, in the case of the deterministic model) that a player of their choice wins the tournament? However, there are other natural objectives around SE tournament design besides favoring a particular player and that is the focus of this paper. Great rivalries generate great entertainment. Imagine a sports tournament that features marquee matches marked by factors such as historic rivalries, contemporaneous news events, geographic proximity or even personal rivalries between members of the opposing teams. These are some of the most widely known and talked about rivalries in the world of sports that greatly enhanced the notoriety and visibility of the sport and thereby achieved great financial success, publicity and relevancy for all the stake holders, be it the organizers, the sponsors, not to mention the participants. From the competition design perspective, that considerations such as revenue, viewer engagement, and relevance should be at the forefront is straightforward.

Thus, scheduling especially attractive matches is a rational tournament design imperative, motivating our algorithmic study of finding a bracket that aims to ensure that a given set of *demand matches* are played in the SE tournament. One can view our model of scheduling a set of demanded games in this setting to be a special case of revenue maximization with unit revenue given to each demand match and zero to all others. Setting the target revenue to be equal to the number of demand matches implies that achieving the target revenue is the same as scheduling all the demand matches. A more general problem was studied by (Lang et al. 2007), in the context of sequential majority voting along binary trees, except they allow arbitrary costs for the edges and the goal is to achieve minimum possible cost in the final SE tournament. They showed this problem to be NP-hard. We also note that our paper is naturally aligned with investigations into the relationship between round-robin and SE tournaments, e.g., (Stanton and Vassilevska Williams 2011). Specifically, while every demand match obviously occurs in a round-robin tournament, how efficiently could one ensure the same in an SE tournament? This is our focus.

Our contributions. We make advances on two fronts– conceptual and algorithmic. On the conceptual front, we introduce a tournament design objective that is both a special case of revenue maximization and as we demonstrate later, a novel variant of the well-studied Subgraph Isomorphism problem. We call the problem DEMAND TOURNAMENT FIXING (Demand-TF), formally defined as follows.

The input contains (i) a directed graph T (called tournament digraph) whose vertices are the players and for *every* pair of players u and v, there is a directed edge (i.e., arc) from u to v if and only if u beats v (assume no ties) and (ii) a set $\mathcal{S}$ of arcs of T. The vertices of T are denoted by $V(T)$ and the arcs by $A(T)$. The goal is to find a bracket (if one exists) such that in the SE tournament generated by this bracket, the matches corresponding to the arcs in $\mathcal{S}$ (called demand matches) take place.

Solving this problem requires one to create a bracket that will ensure that a player u (and its "demand rivals") *all* progress far enough in the tournament so that u is able to play in all the demand matches featuring it. Effectively, we are aiming to create within one single bracket, multiple favorable brackets for each of those players that are somehow highly correlated. Treading this fine line raises fascinating algorithmic challenges as we show in this paper. In fact, for the special case of an acyclic tournament digraph (DAG), i.e., when there is a linear ordering of players according to their strengths where each player beats every player appearing after it, the TF problem is trivial as the strongest player wins every SE tournament. On the other hand, even in this special case, Demand-TF is a challenging problem. However, as we will discuss, our main result implies a polynomial-time algorithm even for this problem on DAGs, as a corollary.

We highlight the relation between our problem and SUBGRAPH ISOMORPHISM (Cygan et al. 2015) problem (SI). TF has a well-established connection with a specific type of spanning tree within the tournament digraph, called a *spanning binomial arborescence* or SBA (Vassilevska Williams 2016). We refer the reader to the section on preliminaries for a formal definition. In fact, there is a solution to the TF instance if and only if the tournament digraph has an SBA rooted at the favorite player. In terms of SI, the tournament digraph is the "host" graph and the "pattern" graph being sought is an SBA rooted at the favorite player. In Demand-TF, the pattern graph is an SBA that contains all the *demand arcs* (those arcs that correspond to demand matches). To the best of our knowledge this is a novel "edge-extension" variant of SI where the goal is to build the pattern graph using a set of given edges as a starting point. The setting of (Manurangsi and Suksompong 2023) can also be interpreted as a constrained version of SI where arcs representing matches between higher ranked players can only occur in parts of the pattern graph that represent later rounds. Their motivation was to prevent the best players from meeting too early.

We next describe our algorithmic contributions (see Table 1). On the one hand, we show that Demand-TF is NP-hard and conditionally rule out any algorithm that runs in time $n^{d^{\mathcal{O}(1)}}$ where d is the number of demand matches. This motivates the search for tractable restrictions of the problem and brings us to the central results of the paper. Here, we make the following contributions:

Algorithms	$n^{O(k)}$-time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$-time if $F \subseteq \mathcal{S}$ $3^n \cdot n^{\mathcal{O}(1)}$-time
Hardness	no $2^{o(n)}$-time algorithm (under ETH) no $n^{d^{\mathcal{O}(1)}}$-time algorithm (if NP $\not\subseteq$ QP)

Table 1: A summary of our results for DEMAND-TF. Here, n is the number of players, k is the size of some minimum feedback arc set F of the input tournament, $\mathcal{S}$ is the set of demand matches and d is the number of demand matches.

Algorithm 1: Demand-TF is P-time solvable when the tournament digraph has a linear ordering on the ability of players with a *constant* number of exceptions where a player with lower ability beats a player with higher ability. In other words, when the *feedback arc set* number of the tournament digraph is constant. This is a natural condition in competitions where there is a clear-cut ranking of the players according to their skills with only a few pairs of players for which the weaker player can beat the stronger player. Motivated by empirical work in (Russell and van Beek 2011), (Aziz et al. 2014) initiated the design of algorithms for TF when the instances have constant feedback arc set number and gave the first P-time algorithm. This restriction was then extensively explored in a series of papers (Ramanujan and Szeider 2017; Gupta et al. 2018, 2019) leading to novel fixed-parameter algorithms. In parameterized complexity parlance, we give an XP algorithm for Demand-TF parameterized by the feedback arc set number k (i.e., running time $n^{\mathcal{O}(k)}$). This brings up the natural question of whether the problem is fixed-parameter tractable (FPT) (i.e., solvable in time $f(k)n^{\mathcal{O}(1)}$ for some function f). Although we do not settle this question in this paper, we identify an additional structural constraint in our next result that leads to an FPT algorithm.

Algorithm 2: If, in the given instance $(T, \mathcal{S})$ of Demand-TF, every upset match is also a demand match, then we get fixed-parameter tractability parameterized by the feedback arc set number. The natural motivation for this scenario is a tournament designer being incentivized (e.g., by betting) to ensure that the upsets take place.

Algorithm 3: We extend our methodology in the preceding algorithms to handle further constraints (in the same running time). In particular, when the designer wants each demand match to take place in a *specific round* of the SE tournament, we can still find such a bracket in time $n^{\mathcal{O}(k)}$. Such constraints allow the designer to ensure that some matches do not occur too early (in the spirit of (Manurangsi and Suksompong 2023)) or too late in the tournament.

Finally, moving to the exact-exponential-time regime we show that assuming the Exponential Time Hypothesis (Impagliazzo and Paturi 2001), one cannot get a subexponential-time algorithm for the problem (i.e., a $2^{o(n)}$ running time where n is the number of players) and complement this

lower bound with an algorithm with running time $2^{O(n)}$ – an asymptotically tight bound.

Organization of the paper. We begin by presenting basic definitions followed by our hardness results. Then, our largest section is dedicated to presenting our main algorithm (**Algorithm 1**). The remainder of the algorithmic contributions (**Algorithm 2** and **Algorithm 3** above) are presented in the following sections. Finally, we conclude with directions for future research.

Preliminaries

Binomial arborescences. 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 digraph T is recursively defined as follows. *(i)* Each $a \in V(T)$ is a binomial arborescence rooted at a. *(ii)* If, for some $i > 0$, H_a and H_b are 2^{i-1}-node binomial arborescences rooted at a and b, respectively, then adding an arc from a to b gives a 2^i-node binomial arborescence (BA) rooted at a. If a binomial arborescence H is such that $V(H) = V(T)$, then H is a *spanning binomial arborescence* (SBA) of T.

The relevance of binomial arborescences comes from the following variant of a result of (Vassilevska Williams 2010).

Proposition 1. *Let T be a tournament digraph and let $\mathcal{S} \subseteq A(T)$. Then, there is a seeding of $V(T)$ such that the resulting SE tournament has every match in $\mathcal{S}$ if and only if T has an* SBA *H such that $A(H) \supseteq \mathcal{S}$.*

We use the terms vertex and player interchangeably. A *rooted forest* is the disjoint union of a set of arborescences. Let H be a rooted forest. For a vertex $u \in V(H)$, denote by $\mathsf{Child}_H(u)$ the set of children of u in H, by $\mathsf{Desc}_H(u)$ the set of descendants of u in H (including u). The *strict* descendants of u comprise descendants of u that are neither u nor children of u. We denote by $\mathsf{Sibl}_H(u)$, the set of siblings of u in H. We define the *height* of v in H, $\mathsf{ht}_H(v) := \log |\mathsf{Desc}_H(v)|$. Note that if H is an SBA of T, then $\forall v \in V(T)$, $\exists i \in [\log n] \cup \{0\}$ such that $|\mathsf{Desc}_H(v)| = 2^i$ and hence $\mathsf{ht}_H(v)$ is an integer. The interpretation in the corresponding SE tournament is that v is the winner of a subtournament played by the players who are descendants of v in H and $\mathsf{ht}_H(v)$ is the number of matches that v wins. If $\mathsf{ht}_H(v) > \mathsf{ht}_H(u)$, then we say that v is *higher* than u and u is *lower* than v. We will also refer to the *height* of a BA meaning the height of its root.

We will use the following characterization of BAs:

Proposition 2. *For $n > 0$, H_n is a* BA *of height n rooted at v_n if and only if $H_n = \bigcup_{i=0}^{n-1} H_i$ where H_i is a* BA *of height i rooted at v_i together with an edge from v_n to each v_i.*

In an instance $(T, \mathcal{S})$ of DEMAND-TF, we call the arcs in $\mathcal{S}$ *demand arcs* or *demand matches* and their endpoints *demand vertices*. For every demand arc (p, q), we say that p is a *demand in-neighbor* or *demand parent* of q and q is a *demand out-neighbor* or *demand child* of p. We will use demand parent and demand child in the context of rooted trees and demand in/out-neighbor otherwise. For a vertex p, the number of its demand in-neighbors is called its *demand in-degree*. The *demand out-degree* is defined symmetrically. $\mathsf{Lose}(\mathcal{S})$ denotes the vertices with a demand in-neighbor, that is, those vertices that lose some demand match.

Hardness Results for DEMAND-TF

We first show that DEMAND-TF is NP-complete and then infer further facts regarding the complexity of this problem using our proof in combination with hardness results on TF proved by (Aziz et al. 2014).

Theorem 1. DEMAND-TF *is NP–complete.*

Proof Sketch. We describe a reduction from an instance (T, v^*) of TOURNAMENT FIXING (TF), which we know is NP-hard (Aziz et al. 2014). Let $n = |V(T)|$. We first construct an n-vertex acyclic tournament, D_1 with source vertex d_1. We call D_1 a "dummy" tournament. Notice that in any SE tournament played by the vertices in D_1 regardless of the seeding, the vertex d_1 will be the winner. Now, define T' to be the tournament obtained by taking the disjoint union of T and D_1 and then doing the following: Add the arc (d_1, v^*), ensuring that v^* loses to d_1. For every other vertex $v \in V(T)$, add the arc (v, d_1), ensuring that d_1 loses to every vertex in T except v^*. Add arcs ensuring that every vertex in T beats every vertex in $V(D_1)$ except for d_1. We initialize $\mathcal{S} := \{(d_1, v^*)\}$, find an arbitrary SBA H_1 on D_1 and for each arc of H_1 incident on d_1, we add it to $\mathcal{S}$. This completes the construction of the DEMAND-TF instance $(T', \mathcal{S})$, which we argue is equivalent to (T, v^*). □

Theorem 2. *From Theorem 1 combined with known hardness of TF (Aziz et al. 2014), we obtain the following.*

1. DEMAND-TF *is NP–complete even if there is a vertex on which every demand arc is incident.*
2. *Unless* NP $\subseteq$ Quasi-*P-time, there is no algorithm for* DEMAND-TF *that runs in time $n^{d^{\mathcal{O}(1)}}$, where d is the number of demand arcs. In particular, this rules out a $2^{d^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$-time fixed-parameter algorithm.*
3. *Assuming the Exponential Time Hypothesis (ETH), there is no $2^{o(n)}$-time algorithm for* DEMAND-TF.

Proof. The first statement follows from the construction in Theorem 1. Moreover, notice that in the same construction, the number of demand arcs is $\log n + 1$. Hence, an algorithm for DEMAND-TF with running time $n^{d^{\mathcal{O}(1)}}$ would imply a quasi-polynomial-time algorithm for TF, which is NP-complete. Finally, the proof of NP-hardness of TF given in (Aziz et al. 2014) reduces an instance of 3-SAT-2L (3-SAT where every literal appears at most twice) with n variables to an instance of TF with $\mathcal{O}(n)$ vertices. Since our reduction from TF only doubles the number of vertices, a $2^{o(n)}$-time algorithm for DEMAND-TF would imply the same running time for 3-SAT-2L, which violates ETH. □

Recall that the naive algorithm for DEMAND-TF has running time $2^{\mathcal{O}(n \log n)}$ as a result of brute-forcing over all possible brackets. We next improve this to a single-exponential running time with a dynamic programming

algorithm, *asymptotically* matching the $2^{o(n)}$-time lower bound from Theorem 2.

Theorem 3. DEMAND-TF *can be solved in time* $3^n n^{\mathcal{O}(1)}$.

DEMAND-TF on Graphs of Bounded Feedback Arc Set Number

We now turn our attention to tournament digraphs with a constant-size (denoted by k) feedback arc set. In this section we will assume that an instance of Demand-TF is a triple $(T, \mathcal{S}, F)$, where F is a minimum feedback arc set of T. The assumption that F is given, is without loss of generality since a minimum feedback arc set of size at most k can be computed in time $3^k n^{\mathcal{O}(1)}$-time (Cygan et al. 2015). We refer to the endpoints of F as *feedback vertices*. Additionally we will assume that $\sigma = v_1, v_2, \ldots, v_n$ is a linear ordering of the vertices of T such that the arcs (v_i, v_j) with $i > j$ are precisely the "upset" matches, i.e., the arcs of F. We say that v_i is *stronger* than v_j for all $j > i$. We say that a subgraph of the tournament digraph is *valid* if every demand arc between two vertices in the subgraph is also present in the subgraph. Thus, Proposition 1 implies that a solution to DEMAND-TF corresponds to a valid SBA of the given tournament digraph.

Note that any player can lose at most one match in an SE tournament. Hence, if $(T, \mathcal{S}, F)$ is an instance of DEMAND-TF in which some vertex has demand in-degree greater than 1, then it is a no-instance. So, we may assume without loss of generality that in any non-trivial instance of DEMAND-TF, every vertex has at most one demand in-neighbor. In the rest of this section we will also assume that all but at most one of the feedback vertices has a demand in-neighbour: this is true if, in the final SBA, *every* feedback vertex except the root has a demand parent. We will ensure this in our algorithm by guessing the parent of every feedback vertex (i.e., the player that beats it) and adding the resulting arc to the set of demand arcs (the overhead is at most n^{2k}).

We also guess the heights of each feedback vertex (overhead $(\log n)^{2k}$). Using this as a starting point, we obtain an estimate for the heights of every vertex via the following definition, where the reader may think of the function g as our guesses for the heights of the feedback vertices.

Definition 2 (Function ht^* and compactness property)**.** Fix a function $g : V(F) \to [\log n]$. For each $v \in V(T)$ let $\mathsf{ht}^*_g(v) = g(v)$ if $v \in V(F)$. Otherwise let $\mathsf{ht}^*_g(v)$ be the minimum non-negative integer satisfying:

1. For each demand arc (v, w), $\mathsf{ht}^*_g(v) > \mathsf{ht}^*_g(w)$.
2. For each u, w such that $(u, v), (u, w) \in \mathcal{S}$, $\mathsf{ht}^*_g(v) \neq \mathsf{ht}^*_g(w)$ if either (i) w is weaker than v or (ii) $w \in V(F)$.

We say that a binomial arborsescence H is *compact with respect to* g if $\mathsf{ht}_H(v) = \mathsf{ht}^*_g(v)$, for every $v \in \mathsf{Lose}(\mathcal{S}) \cap V(H)$.
Additionally, we say that a BA H is *weakly compact* if it is compact with respect to the function ht_H restricted to $V(F)$.

Intuitively, $\mathsf{ht}^*_g(v)$ can be described as follows. Fix a hypothetical solution, that is a valid SBA H, and suppose that in H, we know the heights of each demand child and each weaker demand sibling of v. Moreover, suppose that we know the heights of the vertices in $V(F)$, which is expressed by the function g. Based on this information, since H contains every demand arc, one can narrow down the set of all possible heights that v can have in H, e.g., by using the fact that v is higher than every child, v cannot have the same height as a sibling, and so on. The value of $\mathsf{ht}^*_g(v)$ is the smallest candidate value of the height of v we are left with. In other words, $\mathsf{ht}^*_g(v)$ gives a lower bound on the height of v in *any* solution. This is formally stated below.

Observation 1. *If a valid* BA H *is compact with respect to* g*, then, for all* $v \in V(H)$, $\mathsf{ht}_H(v) \geq \mathsf{ht}^*_g(v)$.

Note that for any vertex v, $\mathsf{ht}^*_g(v)$ is completely determined by only $\mathsf{ht}^*_g(w)$ where w is a child of v or a sibling that is weaker or a feedback vertex. If w is a feedback vertex then $\mathsf{ht}^*_g(w)$ is determined by g, otherwise in both of the other cases, w is weaker than v. So ht^*_g can be easily calculated in polynomial time by simply applying the definition to vertices in strength order beginning with the weakest. From now on we will assume that we know ht^*_g.

The central insight behind our algorithm that leads to the notion of compactness is that in yes-instances, there is always a solution where the height of *every vertex* that loses a demand match is precisely this smallest candidate value (this is formalized in Lemma 2). This motivated our definition of weak compactness in Definition 2. Given this fact, our algorithmic strategy is to "pack" the rest of the vertices into the solution SBA using an intricate subroutine that is guided by this insight. Roughly speaking, our algorithm will use a greedy approach to complete the packing, where at any step, a set of partially constructed subgraphs are available and the goal is to make a new partial solution that contains the latest vertex that is processed. However, the challenge our approach has to face is that in the intermediate steps of our algorithm we would be dealing with partially constructed subgraphs (i.e., forests) where each component is not necessarily an SBA, yet we cannot simply break them apart since they encode important height information that we wish to enforce in the complete solution. Thus, the step-by-step challenge, solved by subroutine PACK which we describe later, is to carefully "glue" some of these structures together to form a supergraph in each step such that in the final step we have a BA. The trickiest aspect is to do this in such a way that if at any point we cannot find appropriate pieces to glue together, we are able to correctly reject.

Guaranteed compactness. We next formalize our central insight and prove that every yes-instance has a valid *weakly compact* SBA. Towards this, we argue that we can modify any valid SBA to achieve this property using the following "exchange" lemma.

Lemma 1. *Suppose* H *is a valid* SBA*,* $(u, v) \in \mathcal{S}$*,* $w \in \mathsf{Sibl}_H(v)$ *such that* $v, w \notin V(F)$*,* w *is lower than* v *and let* B *be the set of children of* v *that are at least as high as* w*. Moreover, suppose that* v *has no demand out-neighbors in* B *and either (i)* w *is stronger than* v *or (ii)* $(u, w) \notin \mathcal{S}$*. Then,* H *can be transformed to a valid* SBA*,* H' *where the following hold:*

1. $\mathsf{ht}_{H'}(v) = \mathsf{ht}_H(w)$. *That is, after the transformation,* v *now has the "old" height of* w.
2. *For every* $x \notin B \cup \{v, w\}$, $\mathsf{ht}_H(x) = \mathsf{ht}_{H'}(x)$. *That is, except for a few vertices adjacent to* v *in* H *the heights of all other vertices remain the same after the transformation.*

Lemma 2. *If there exists a valid* SBA H*, then there exists a valid, weakly compact* SBA.

Proof Sketch. In any non-weakly-compact SBA there is a weakest vertex that contradicts the definition of weak compactness. We choose a valid SBA that has the strongest such "certificate of non-compactness" and aim to find another SBA with a stronger one which would give the required contradiction. By properties of BAs this certificate vertex has a sibling that is the correct height so we apply Lemma 1 to swap these heights. This gives us the required SBA. □

Before moving to the description of our packing subroutine, we need to define a natural relaxation of BA to account for feedback vertices.

Partial Binomial Arborescences. In our greedy packing strategy, we will process vertices from weakest to strongest, with the underlying assumption that when we arrive at a vertex v, all descendants of v are weaker and have their respective sub-arborescences built in an earlier step. When T is acyclic this works fine, however this does not go smoothly when we have cycles since a descendant of the current vertex, v, may actually be stronger than v. Hence it is not yet processed by our algorithm, and consequently its sub-arborescence is not yet built. This leads to the scenario that the BA in the solution that is rooted at v cannot be fully built either. Notwithstanding this difficulty, we note that the partial structures our algorithm deals with have enough BA-like properties that one direction of Proposition 2 still holds. This leads us to the following definition, which relaxes the conditions of a BA when feedback vertices are encountered.

Definition 3 (Feedback descendants and partial binomial arborescence)**.** Given a rooted forest Q and a vertex $v \in V(Q)$, define the *feedback descendants of* v *in* Q,

$$\mathsf{FDesc}_Q(v) := \bigcup_{f \in \mathsf{Desc}_Q(v) \cap V(F), f \neq v} \mathsf{Desc}_Q(f) \setminus \{f\}$$

Given a BA H of height i, on a tournament digraph T with feedback arc set F, we call a subtree H' of H a *partial binomial arborescence (*PBA*) of height* i if $V(H) \setminus V(H') \subseteq \mathsf{FDesc}_H(\mathsf{root}(H))$.

In Definition 3, $\mathsf{FDesc}_Q(v)$ are strict descendants of a feedback vertex that is itself a strict descendant of v. Equivalently these are vertices x where the path from v to x contains a feedback vertex that is not v or x, that is, x is "past" a feedback vertex. A PBA is a BA that is missing some feedback descendants of its root. Figure 1 contains an example of feedback descendants.

Observation 2. *For* $n > 0$*, if* $H_n = \bigcup_{i=0}^{n-1} H_i$ *where* H_i *is a* PBA *of height* i *rooted at* v_i *together with an edge from* v_n *to each* v_i *then* H_n *is a* PBA *of height* n *rooted at* v_n.

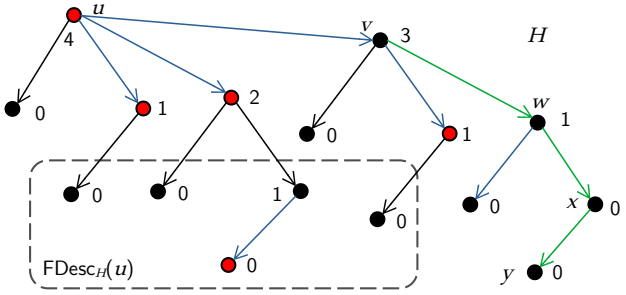

Figure 1: A BA H and the feedback descendants of its root. Feedback vertices are in red and demand arcs are in blue. ht^*_g (where g is ht_H restricted to $V(F)$) is noted next to each vertex. $\mathsf{ht}^*_g(v) = 3$ due to its demand siblings. The green arcs are those that would be added during the inner loop of the algorithm when $v_{n-i} = v$ and $j = 2$. The call to PACK would use $P = \{w, x, y\}$, add the arcs (w, x) and (x, y) and output w. Finally Step 11 would add the arc (v, w).

Guessed size. Definition 3 defines the height of a PBA implicitly. Our algorithm constructs an SBA by gluing PBAs of specific heights together so it needs their heights, or equivalently an appropriate notion of their size, to see if there are any good candidate PBAs to glue together. The following definition allows us to calculate the size of a PBA.

Definition 4. Given a rooted forest Q, and $g : V(F) \to [\log(n)]$ define the *guessed size of* v,

$$\mathsf{sz}_{Q,g}(v) = \begin{cases} 2^{g(v)} & \text{if } v \in V(F) \\ 1 + \sum_{w \in \mathsf{Child}_Q(v)} \mathsf{sz}_{Q,g}(w) & \text{otherwise} \end{cases}$$

We will drop the reference to g when it is clear from the context.

Note that if H is an SBA and g is ht_H restricted to $V(F)$, then $\mathsf{sz}_{H,g}(v) = |\mathsf{Desc}_H(v)| = 2^{\mathsf{ht}_H(v)}$ for all v. Furthermore if $H' \subset H$ is a PBA of height i rooted at v then $\mathsf{sz}_{H',g}(v) = 2^i$. Effectively, deleting feedback descendants of v does not change the value of $\mathsf{sz}(v)$. Note that the converse is not true so we will need to prove that a given subgraph is a PBA and sz will then check its height.

Since Q is a rooted forest, each vertex is the child of at most one vertex. So applying the definition recursively will only require calculating $\mathsf{sz}_Q(w)$ once for each vertex $w \in V(Q)$. Therefore $\mathsf{sz}_Q(v)$ can be calculated in polynomial time for any Q and v directly from the definition.

The subroutine PACK. The following lemma describes a crucial subroutine for us. Informally, the subroutine creates a PBA of height j by adding arcs between the provided vertices in one of its inputs (P) and then outputs the root of this PBA. Crucially, it is a very local algorithm: it only affects vertices from P. This allows us to repeatedly call it without undoing work it has already done in a previous call. In order to describe this property we will need the following definition: suppose Q and Q' are both rooted forests, then define $\mathsf{Desc}^Q_{Q'}(v)$ as the set of vertices that are descendants

of v in Q' but that have no parent in Q. Note that all of these vertices except v must have a parent in Q'.

Lemma 3 (Packing lemma)**.** *There is a polynomial-time algorithm* PACK *that:*

- *Takes as input a tuple (Q, P, j) such that:*
  1. *$Q \subset T$ is a valid rooted forest.*
  2. *For every $w \in P$:*
  (a) $Q[\mathsf{Desc}_Q(w)]$ is a PBA *of height at most j.*
  (b) w has no parent in Q.
  3. *$\sum_{w \in P} \mathsf{sz}_Q(w) \geq 2^j$*
- *Outputs a tuple (Q', v) such that:*
  1. *$Q' \subset T$ is a valid rooted forest and Q' is a supergraph of Q.*
  2. *$Q'[\mathsf{Desc}_{Q'}(v)]$ is a* PBA *of height j.*
  3. *v has no parent in Q'.*
  4. *If $Q'[\mathsf{Desc}_{Q'}(x)] \neq Q[\mathsf{Desc}_Q(x)]$ then $x \in \mathsf{Desc}^Q_{Q'}(v)$. Additionally $\mathsf{Desc}^Q_{Q'}(v) \subseteq P$. That is, all vertices affected by* PACK *become descendants of v in Q' and were from P.*
  5. *For all $s \in P \setminus \mathsf{Desc}^Q_{Q'}(v)$ and $t \in \mathsf{Desc}^Q_{Q'}(v)$ we have $\mathsf{sz}_Q(s) \leq \mathsf{sz}_Q(t)$. That is, the algorithm uses the vertices with largest height.*

Proof Sketch. Initially let $Q_0 = Q$ and $P_0 = P$. For each $i \geq 0$ either:

- There exists $v \in P_i$ with $\mathsf{sz}_{Q_i}(v) = 2^j$. Then return (Q_i, v).
- Or there is no such v. Then let x, y be two vertices of largest sz with x the stronger. More precisely, choose $x, y \in P_i$ such that $\mathsf{sz}_{Q_i}(x) = \mathsf{sz}_{Q_i}(y)$ and for all other $a, b \in P_i$ satisfying $\mathsf{sz}_{Q_i}(a) = \mathsf{sz}_{Q_i}(b)$ we have $\mathsf{sz}_{Q_i}(a) \leq \mathsf{sz}_{Q_i}(x)$. Let $Q_{i+1} = Q_i \cup \{(x, y)\}$ and $P_{i+1} = P_i \setminus \{y\}$ then repeat for the next i. □

We will also need the following corollary of Lemma 3.

Corollary 1. *Lemma 3 holds if every occurrence of* PBA *is replaced by* BA*.*

Informally this says that the same subroutine can be used to pack BAs: by replacing PBAs with BAs in the input we get a BA in the output.

We can now present our main result.

Theorem 4. *An instance of* DEMAND-TF*, $(T, \mathcal{S})$, can be solved in time $n^{\mathcal{O}(k)}$ where k is the feedback arc set number of T.*

We first describe the algorithm claimed in the above statement, following which we sketch the proof of correctness.

In our algorithm, we first guess an injective $p : V(F) \to V(T) \cup \{\perp\}$ representing the guessed parent of each feedback vertex. Then, for each $v \in V(F)$, we add the arc $(p(v), v)$ to $\mathcal{S}$ (unless $v \in \mathsf{Lose}(\mathcal{S})$ or $p(v) = \perp$) to ensure this guess is honored by the algorithm and justify our assumption that every feedback vertex has a parent. We then guess $g : V(F) \to [\log(n)]$ representing the guessed heights of the feedback vertices and calculate ht^*_g. Now we can sanity check g: if $(u, v) \in \mathcal{S}$ we check that $\mathsf{ht}^*_g(u) > \mathsf{ht}^*_g(v)$ and if $(u, v), (u, w) \in \mathcal{S}$ we check that $\mathsf{ht}^*_g(v) \neq \mathsf{ht}^*_g(w)$. Finally, if $p(v) = \perp$, we check that $g(v) = \log(n)$ since this is guessing that v has no parent, i.e., it is the root. Following this, we call Algorithm 1 once with each possible combination of the guesses of g and value of $\mathcal{S}$ given by our guess of p. We reject the input if every invocation of Algorithm 1 fails to output a solution.

Algorithm 1:

```
1  Q_{0,0} ← (V(T), S);
2  for 0 ≤ i < n do
3      for 0 ≤ j < ht*_g(v_{n−i}) do
4          if there exists y ∈ Child_{Q_{i,0}}(v_{n−i}) with
             sz_{Q_{i,0}}(y) = 2^j then
5              Q_{i,j+1} ← Q_{i,j};
6          else
7              Let P_{i,j} be the set of vertices, w, that are
                 weaker than v_{n−i}, have no parent in
                 Q_{i,j}, and have sz_{Q_{i,j}}(w) ≤ 2^j;
8              if Σ_{w∈P_{i,j}} sz_{Q_{i,j}}(w) < 2^j then
9                  Reject;
10             (Q̂_{i,j+1}, w_{i,j}) ← PACK(Q_{i,j}, P_{i,j}, j);
11             Q_{i,j+1} ← Q̂_{i,j+1} ∪ {(v_{n−i}, w_{i,j})};
12     Q_{i+1,0} ← Q_{i,ht*_g(v_{n−i})};
13 Let P* be the set of vertices without parents in Q_{n,0};
14 if Σ_{z∈P*} sz_{Q_{n,0}}(z) < n then
15     Reject;
16 (Q*, v*) ← PACK(Q_{n,0}, P*, log(n));
17 return Q*;
```

The outer loop iterates over each vertex from weakest to strongest. By Proposition 2, v_{n-i} needs to have a child of height j for each j that the second loop considers. Step 4 checks if such a child already exists (this happens when it is a demand child). Otherwise we check if there are enough vertices that could become descendants of v_{n-i} and then call PACK to create such a child (see Figure 1). After applying this process to every vertex we will have a number of BAs that we can then pack into an SBA at Step 16.

Recall that ht^*_g can be calculated in polynomial time. Also $\mathsf{ht}^*_g \leq \log(n)$ so both loops run at most n times. Recall that each value of sz can be calculated in polynomial time. There are at most n children of any vertex so the existence of y can be checked in polynomial time. Each $P_{i,j}$ can be calculated in polynomial time by checking whether each vertex satisfies the conditions on vertices of $P_{i,j}$. Finally PACK is a polynomial-time subroutine. So since there are $\log(n)^{\mathcal{O}(k)}$ possible values for g and $n^{\mathcal{O}(k)}$ possible values for p (and the resulting value of $\mathcal{S}$), the overall running time is $n^{\mathcal{O}(k)}$.

Sketch of correctness. We first consider the case where the algorithm does not reject. We prove by induction that, after

each iteration, the descendants of the vertices we will try to pack at the next iteration form PBAs. This allows us to apply PACK. We then extend this statement to BAs for all vertices without parents in $Q_{n,0}$. We can then use Corollary 1 to show that the final output of the algorithm is an SBA.

We then prove the converse: that is, if the algorithm rejects then we do indeed have a negative instance. Suppose for a contradiction that the algorithm rejects but there exists a valid SBA. By Lemma 2 there exists a valid, weakly compact SBA. We use the following lemma:

Lemma 4. *Given a valid, weakly-compact* SBA H*, for all* i, j *there exists a valid* SBA $H_{i,j}$ *that is compact with respect to* ht_H *restricted to* $V(F)$*, such that* $Q_{i,j} \subset H_{i,j}$*.*

Now we have an SBA that agrees with the algorithm up to the point it rejects. Since this is a compact SBA it has the same (or greater) heights as the algorithm was trying to pack to so we can look at which vertices are descendants in this SBA. The algorithm could have chosen these vertices which contradicts the rejection of the algorithm and completes the proof of correctness. □

In the following sections, we describe how our algorithm can be extended or modified to obtain the further algorithmic results outlined in the Introduction.

FPT Algorithm for DEMAND-TF When Upsets Are Demanded

Suppose that $F \subseteq \mathcal{S}$. A closer inspection of our algorithm shows that where we assumed that every feedback vertex except the root has a demand parent, a weaker assumption is sufficient, specifically, the following property.

Property 1. *For all* $v \notin \mathsf{Lose}(\mathcal{S})$ *there is no* $(u, v) \in F$*.*

That is, every vertex without a demand parent has no in-feedback arc. Equivalently $\mathsf{Lose}(F) \subseteq \mathsf{Lose}(\mathcal{S})$. Clearly this is implied by $F \subseteq \mathcal{S}$. This means that any vertex that is stronger than v, and only these vertices, can be used as its parent. Furthermore, even if v is a feedback vertex the descendants of v are weaker than its ancestors, at least until further feedback vertices. Effectively v does not act as a feedback vertex (if there is a feedback edge (v, w) we will handle this when talking about w) so we replace $V(F)$ with $\mathsf{Lose}(F)$ throughout the algorithm: these are the remaining feedback vertices that still behave as such. Whenever the analysis refers to a vertex not being a feedback vertex because it has no demand parent we now simply use Property 1: although it may be a feedback vertex, it behaves as a non-feedback vertex since it has no incoming feedback edges.

The only changes required to our algorithm are therefore the removal of the guess of the parents of feedback vertices (eliminating the overhead of $n^{\mathcal{O}(k)}$) and a slightly smaller set of feedback vertices used in the domain of g and the definitions of ht^* and sz. Since the guess of the parents of the feedback vertices has been removed the runtime is dominated by the guess of g: there are $(\log n)^{\mathcal{O}(k)} = (k \log k)^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ possibilities for g so the overall run time is FPT in k.

XP Algorithm for DEMAND-TF With Specified Rounds for Demands

The first key observation is that the round of a match is exactly the height of that the losing player of the match takes in the solution (numbering rounds from zero). That is, if a demand match (u, v) occurs at round i in the tournament represented by an SBA H then $\mathsf{ht}_H(v) = i$. So if every demand match has a specified round this is equivalent to specifying the heights of every vertex in $\mathsf{Lose}(\mathcal{S})$. Theorem 4 guesses the heights of vertices in $V(F)$ and calculates the heights of all other vertices in $\mathsf{Lose}(\mathcal{S})$. So by extending the domain of g to $\mathsf{Lose}(\mathcal{S}) \cup V(F)$ and ensuring it agrees with the required heights the algorithm will ensure that the SBA it outputs will satisfy our additional constraints. Note that when we extend the domain of g in this way, the guess is still made only for the vertices in $V(F)$ as the value of g for the vertices in $\mathsf{Lose}(\mathcal{S})$ is part of the input. Thus, the running time of the new algorithm remains the same as that of Theorem 4.

Future Work

1. Theorem 4 shows the viability of a systematic study of the parameterized complexity of Demand-TF with respect to other parameters studied for TF, such as feedback *vertex* set.
2. We also propose a relaxation of our model where, if, there is no seeding that enables every demand match to take place, then the goal is to compute a seeding that makes the *maximum* number of demand matches take place. How efficiently could one do this? Our work implies a $2^d n^{\mathcal{O}(k)}$-time algorithm for this problem, where d is the total number of demands and k is the feedback arc set number – simply guess the set of satisfied demands and invoke Theorem 4. A natural follow-up question is whether one can remove the exponential dependence on d and have an XP algorithm parameterized by k alone? Resolution of this question would require additional ideas to this paper. For instance, we can no longer assume that every vertex has at most one demand in-neighbor. Efficient approximation algorithms for this problem are also an interesting research direction. In terms of exact-exponential-time algorithms, it is easy to see that the algorithm of Theorem 3 already extends naturally to this variant.
3. Could one extend our results for Demand-TF to the probabilistic model? Here, the tournament designer has two natural objectives – maximize the probability that all demand matches are played or maximize the expected number of satisfied demands.
4. The edge-constrained version of SUBGRAPH ISOMORPHISM we define in this paper is a problem of independent interest. Techniques such as color coding (Alon, Yuster, and Zwick 1995) could give FPT algorithms for this problem parameterized by the size of some simple pattern graphs. However, the behavior of this problem with respect to various structural parameterizations of graphs is less clear and we leave this as a research direction of broad interest to the algorithms community.

Acknowledgements

Sushmita Gupta acknowledges support from SERB's MATRICS Grant (MTR/2021/000869) and SUPRA Grant (SPR/2021/000860). Ramanujan Sridharan acknowledges support by the Engineering and Physical Sciences Research Council (grant numbers EP/V007793/1 and EP/V044621/1).

References

Alon, N.; Yuster, R.; and Zwick, U. 1995. Color-Coding. *J. ACM*, 42(4): 844–856.

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. AAAI Press.

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. ISBN 978-3-319-21274-6.

Gupta, S.; Roy, S.; Saurabh, S.; and Zehavi, M. 2018. 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 Kraus, S., ed., *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 322–328. ijcai.org.

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

Impagliazzo, R.; and Paturi, R. 2001. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2): 367–375.

Lang, J.; Pini, M. S.; Rossi, F.; Venable, K. B.; and Walsh, T. 2007. Winner Determination in Sequential Majority Voting. In Veloso, M. M., ed., *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 1372–1377.

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

Manurangsi, P.; and Suksompong, W. 2023. Fixing knockout tournaments with seeds. *Discret. Appl. Math.*, 339: 21–35.

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

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

Russell, T.; and van Beek, P. 2011. An Empirical Study of Seeding Manipulations and Their Prevention. In Walsh, T., ed., *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 350–356. IJCAI/AAAI.

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

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

Vassilevska Williams, 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*. AAAI Press.

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

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

Zehavi, M. 2023. Tournament Fixing Parameterized by Feedback Vertex Set Number Is FPT. In Williams, B.; Chen, Y.; and Neville, J., eds., *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, 5876–5883. AAAI Press.