Fixing a Balanced Knockout Tournament

Haris Aziz and Serge Gaspers and Simon Mackenzie and Nicholas Mattei
NICTA and UNSW, Sydney, Australia

Paul Stursberg
Technische Universität München, Germany
paul.stursberg@ma.tum.de

Toby Walsh
NICTA and UNSW, Sydney, Australia
toby.walsh@nicta.com.au

Abstract

Balanced knockout tournaments are one of the most common formats for sports competitions, and are also used in elections and decision-making. We consider the computational problem of finding the optimal draw for a particular player in such a tournament. The problem has generated considerable research within AI in recent years. We prove that checking whether there exists a draw in which a player wins is NP-complete, thereby settling an outstanding open problem. Our main result has a number of interesting implications on related counting and approximation problems. We present a memoization-based algorithm for the problem that is faster than previous approaches. Moreover, we highlight two natural cases that can be solved in polynomial time. All of our results also hold for the more general problem of counting the number of draws in which a given player is the winner.

Introduction

Balanced knockout tournaments are one of the most widely-used formats for sports competitions (Horen and Riezman 1985; Connolly and Rendleman 2011; Groh et al. 2012). A prominent example is the Wimbledon Men’s tennis tournament in which 128 players enter the tournament and the player who wins seven consecutive matches right from the first round to the final wins the tournament. The format is also used in certain elimination style election and decision-making. We consider the computational formats for sports competitions, and are also used in election.

A prominent example is the Wimbledon Men’s tennis tournament in which 128 players enter the tournament and the player who wins seven consecutive matches right from the first round to the final wins the tournament. The format is also used in certain elimination style election and decision-making. We consider the computational formats for sports competitions, and are also used in election.

Related counting and approximation problems. We present a memoization-based algorithm for the problem that is faster than previous approaches. Moreover, we highlight two natural cases that can be solved in polynomial time. All of our results also hold for the more general problem of counting the number of draws in which a given player is the winner.

Probabilistic Tournament Fixing Problem (PTFP)

Instance: Player set \( N \), pairwise comparison matrix \( P \), a distinguished player \( i^* \) in \( N \), and target probability \( q \in [0, 1] \).

Question: Does there exist a draw \( \sigma \) for the player set \( N \) for which the probability of \( i^* \) winning \( T(N, \sigma) \) is at least \( q \)?

PTFP was proposed by Vu, Altman, and Shoham (2009) and has been studied in numerous papers (see e.g., (Stanton and Vassilevska Williams 2011a; 2011c; 2011b)). It is a well-motivated problem in sports analytics (Silver 2011). Vu, Altman, and Shoham (2009) and Vassilevska Williams (2010) showed that PTFP is NP-hard for various restrictions. These results also imply that computing the maximum possible winning probability of a given player is NP-hard. Nevertheless, the computational complexity of a particularly natural and interesting problem,
the Tournament Fixing Problem (TFP), has remained a major open question. In the TFP, \( P \) is deterministic i.e., \( P_{jk} \in \{0, 1\} \) for all players \( j, k \in N \). We say that player \( j \) beats player \( k \) if \( P_{jk} = 1 \). The winner of each match is deterministically known beforehand and the question is whether there exists a draw for which a given player can win in the corresponding BKT.

**Tournament Fixing Problem (TFP)**

**Instance:** Player set \( N \), deterministic pairwise comparison matrix \( P \), and a distinguished player \( i^* \in N \).

**Question:** Does there exist a draw \( \sigma \) for the player set \( N \) for which \( i^* \) is the winner of \( T(N, \sigma) \)?

TFP is equivalent to checking whether there exists a seeding \( \pi \) for which \( i \) is the winner of \( T(N, \pi) \). We note that TFP is a special case of the problem with same name as defined in (Vassilevska Williams 2010).\(^1\) TFP is also a special case of \#TFP — the problem of counting the number of draws for which a given player is the winner. This count can be used to compute the probability of a player winning in a draw chosen uniformly at random. It can also be considered as the relative strength of the player.

**Contributions.** We first settle the computational complexity of TFP by showing that it is NP-complete. The problem was explicitly stated as an open problem a number of times (Vu, Altman, and Shoham 2009; Vassilevska Williams 2010; Russell and van Beek 2011; Stanton and Vassilevska Williams 2011b; 2011a; 2011c; Lang et al. 2012; Vu et al. 2013). As a corollary, we show that unless \( P = \text{NP} \), there exists no polynomial-time approximation algorithm for computing the maximal winning probability of a player. The inapproximability result provides additional motivation for the line of work in which heuristic algorithms have been proposed for PTFP (Vu et al. 2013). Another corollary is that there exists no fully polynomial time randomized approximation scheme (FPRAS) for counting the number of draws for which a player is the winner.

In view of these intractability results, we identify two natural cases for which even \#TFP can be solved in polynomial time. In the first case, the players can be divided into a constant number of player types. It appeals to the scenario where a player with lower ability beats a player with higher ability. The TFP instance consists of a set of players \( N \), and a distinguished player \( \sigma \in N \). We say that player \( \sigma \) is the winner of \( T(N, \sigma) \).

In the version of TFP defined in (Vassilevska Williams 2010), there can be additional constraints in which certain matches are disallowed.

The condition is quite natural since in many competitions 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 be the stronger player. For example, as of 15/01/2014, Nikolay Davydenko was the only tennis player among the men’s top 64 who had a winning head-to-head record against Rafael Nadal.

\(^1\)In the version of TFP defined in (Vassilevska Williams 2010), there can be additional constraints in which certain matches are disallowed.

\(^2\)The condition is quite natural since in many competitions 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. For example, as of 15/01/2014, Nikolay Davydenko was the only tennis player among the men’s top 64 who had a winning head-to-head record against Rafael Nadal.

**Related Work.** After the work of Vu, Altman, and Shoham (2009), PTFP and TFP have been studied in a number of AI papers. Vassilevska Williams (2010) identified various sufficient conditions for a player to be a winner of a BKT. In a followup paper, Stanton and Vassilevska Williams (2011c) focused on when weak players can possibly win a BKT. Stanton and Vassilevska Williams (2011b; 2011a) identified conditions in a probabilistic model under which the tournament organizer can fix the tournament with high probability. In (Vu and Shoham 2011), the problem of designing ‘fair’ draws was considered. Lang et al. (2007) and Lang et al. (2012) examined winner determination in voting trees that need not be balanced.

TFP can also be considered as the problem of checking whether a player is a possible winner in a BKT. Computing possible winner for other voting rules where the information on the preferences is not complete has been studied extensively (Xia and Conitzer 2011; Aziz et al. 2012). Another related problem is checking whether a sports team can still win a round-robin competition when all the matches have not yet been completed (Kern and Paulusma 2004; Gusfield and Martel 2002).

**TFP is NP-complete**

In this section, we settle the complexity of TFP. For convenience, we will represent the pairwise comparison matrix \( P \) as a graph where an edge from \( i \) to \( j \) exists iff \( i \) beats \( j \).

**Theorem 1.** TFP is NP-complete.

**Proof.** We reduce from the NP-hard variant of the 3SAT problem in which every literal appears at most twice. Given such a 3SAT instance \( F = (X, C) \) where \( X = \{x_1, \ldots, x_{|X|}\} \) is the set of variables and \( C \) the set of clauses, we build an instance of TFP where a draw exists such that a distinguished player \( m_1 \) wins iff \( F \) is satisfiable. The TFP instance consists of a set of players \( N = \{1, \ldots, n\} \) where \( n \) is the smallest power of 2 greater than \( 32 \cdot |X| \). The resulting knock-out tournament will thus consist of \( R := \lceil \log(32 \cdot |X|) \rceil - 1 \geq 5 \) rounds where the first (lowest) four rounds will be used to store the gadgets while the later rounds will enforce certain outcomes for the gadgets. We can decompose the set of players \( N \) as follows:

\[
N = M \cup S \cup G^X \cup G^{CG} \cup G^F
\]  

(1)

where players in \( G^X \) are used in the choice gadgets that will model the variable assignment, players in \( G^{CG} \) are used in the clause/garbage gadgets that will model the behaviour of the clauses, and players in \( G^F \) are used in filler gadgets that will be used to balance the BKT. Players in \( S \) are special players that will ensure the connection between choice and clause gadgets. Finally we will show that for \( m_1 \) to win the BKT, all \( k := \frac{n}{10} \) players in \( M = \{m_1, m_2, \ldots, m_k\} \) will have to proceed to the fifth round. We will use a total of
As the leftmost player, \( m \) match has spawned the right player, thus the left player wins.

**Lemma 1.** Let \( \sigma \) be a draw such that \( m_1 \) wins the BKT \( T(M, \sigma) \).

**Proof.** Seeding all players in \( M \) from left to right according to the spawning process makes \( m_1 \) win the tournament (see Figure 1): Whenever two players meet, the left player in the match has spawned the right player, thus the left player wins. As the leftmost player, \( m_1 \) wins the tournament.

**Global structure.** We now describe how the sets from (1) and (2) relate to each other. In many places, we will use the right-left-rule, that is elements from sets with a higher index will beat elements from sets with a lower index. For instance, for all \( j > j' \) and elements \( i \in \{m_j\} \cup G_j, i' \in \{m'_{j'}\} \cup G_{j'} \) where \( (i, i') \neq (m_j, m_{j'}) \) we have that \( i \) beats \( i' \). All members of \( S \) beat all other players unless explicitly stated otherwise. The set \( S \) can be partitioned into subsets \( S_j \) corresponding to each variable of the SAT instance, such that \( \forall j \in [|X|]: S_j = S_{x_j} \cup S_{\bar{x}_j} \cup \{s_j^c\} \) where

\[
\begin{align*}
G^X &= \bigcup_{j=1}^{\lfloor |X|/2 \rfloor} G_j; & G^{CG} &= \bigcup_{j=\lfloor |X|/2 \rfloor+1}^{2|X|} G_j; & G^F &= \bigcup_{j=2|X|+1}^{k} G_j
\end{align*}
\]

Note that the sets \( G_j \) have \( 10, 13 \) or \( 15 \) elements, depending on whether they are a subset of \( G^X, G^{CG} \) or \( G^F \), respectively.

**Set \( M \).** The relation between elements of the set of winners \( M \) is recursively defined via a linear ordering of players as follows: We start with player \( m_1 \). At each iteration, every player \( a \) spawns a new player \( b \) placed directly to his right. In the pairwise comparison graph, each player beats all players to his left in this construction except for the one that spawned him. This recursive construction is repeated until a total of \( k \) players are present (see Figure 1).

**Choice gadget.** For each \( j \in [|X|] \), the \( j \)-th choice gadget consists of player \( m_j \), all ten elements of \( G_j \) and all of \( S_j \). Note that some elements of \( S_j \) will appear again in the clause/garbage gadgets. The pairwise comparison graph for these elements (for fixed \( j \)) is shown in Figure 2. The choice gadget is structured in such a way that it is possible for \( m_j \) to win a subtournament composed of all elements in the gadget except two elements of either \( S_{x_j} \) or \( S_{\bar{x}_j} \) as illustrated in Figure 4. We will also show that this is the only way in which \( m_j \) can reach the fifth round in a tournament won by \( m_1 \).

**Clause/garbage gadget.** We now describe the internal structure of the clause/garbage gadgets. The \( j \)-th gadget consists of \( m_j \) and the 13 elements of \( G_j \), two of these are denoted \( c/g \). The pairwise comparison graph for these players is shown in Figure 3. For each clause \( c \in C \) we will call one of the players denoted \( c/g \) associated with clause \( c \), all remaining players \( c/g \) are garbage players. All players shown in the figure are beaten by all players in \( S \) with the following exceptions: (i) garbage players beat all players from \( S_p \), (ii) players associated with clause \( c \) beat all players from the set \( S_{x_j} \) or \( S_{\bar{x}_j} \) if \( x_j \) or \( \bar{x}_j \) occurs in clause \( c \), respectively.

The clause gadget is structured in such a way that it is possible for \( m_j \) to win a subtournament composed of all elements in the gadget with the addition of a compatible \( S_p \) element for both clause/garbage players included. We will
Lemma 3. For $m_1$ to win the tournament, all players $m_j \in M$ must reach the fifth round, winning a 16-player subtournament that contains all players from $G_j$ by playing against all four players from $G_j$ that they can beat.

Proof. Let $M_j$ denote the set of $2^j$ players from $M$ that are present after the $\ell$-th iteration of the spawning process. Furthermore, denote by $\ell_j$ the smallest $\ell$ such that $m_j \in M_\ell$. We use induction over the set $M$. The induction will proceed from left to right according to the order specified in the description of the spawning process. First, note that for all $j \in [k]$, player $m_j$ beats exactly $(R - 4 - \ell_j)$ players from $M$ that are to his right. The induction hypothesis is as follows.

In order for $m_3$ to win the tournament, the following must hold for all $j \in [k]$:

(i) In some round, $m_j$ plays against the player that spawned him, (ii) $m_j$ must reach round $(R + 1 - \ell_j)$ and be beaten in that round, (iii) $m_j$ plays against all players from $G_j$ that he can beat, (iv) the subtournament consisting of the first four rounds that $m_j$ wins contains all players from $G_j$.

For the base case, consider $m_1$. For (i) and (ii) there is nothing to show, as $m_1$ must win the tournament. As $m_1$ only beats a total of $R$ players, he needs to play against all of these, which proves Part (iii). Part (iv) is implied by Lemma 2.

We now show that for a player $m_j \in M$, the induction hypothesis holds, provided that it holds for all players $m_{ji} \in M$ with $j' < j$. Consider the player $m_{ji}$ that spawned $m_j$.

For (i), note that by the induction hypothesis, all players from $G_{ji}$ with $j' < j^*$ have to be placed in the subtournament won by $m_{ji}$. Similarly, all players $m_{ji}$ with $j' < j^*$ have to play (and lose) against the player that spawned them, therefore they cannot play against $m_{ji}$ (as they would lose). Thus, $m_{ji}$ only play against players from $G_{ji}$ and players $m_r$ from $M$ to $m_{ji}$’s right and the respective sets $G_r$. He beats exactly four players in $G_{ji}$, none in the other $G_j$ and $(R - 4 - \ell_{ji})$ from $M$ to his right. By the induction hypothesis, he thus needs to play against all of these (including $m_{ji}$) to proceed to round $(R + 1 - \ell_{ji})$.

For (ii), note that to the left of $m_{ji}$ there are $(R - 4 - \ell_j)$ other players that are spawned by $m_{ji}$. Denote them by $m_{ji}, m_{ji}, \ldots, m_{ji,i+4-\ell_j}$ and note that $R + 1 - \ell_{ji}^* = i + 4$. Thus, $m_{ji}$ cannot face $m_{ji}$ in round $5, 6, \ldots, R - \ell_{ji}$ (as other players need to play against $m_{ji}$ in these rounds) and has to proceed at least until round $(R + 1 - \ell_{ji})$. By the same argument as for $m_{ji}$ above, before $m_{ji}$ faces $m_{ji}$, he can only play against players from $G_j$ and players $m_r$ from $M$ to $m_{ji}$’s right and the respective sets $G_r$. Player $m_{ji}$ only beats a total of $R - \ell_{ji}$ players among these, thus $m_{ji}$ must be beaten in round $R - \ell_{ji}$. Furthermore, he must play against all of the above players that he can beat, which settles Part (iii).

Again, Part (iv) is implied by Lemma 2 as by the argument above, $m_{ji}$ needs to play against all four players from $G_j$ that he can beat.

We can now finally show the following lemma.

Lemma 4. There exists a draw such that $m_1$ wins the tournament if and only if the SAT instance is satisfiable.

Proof. First, note that by Lemma 1 and Lemma 3, $m_1$ can win the tournament iff all players from the set $M$ reach the
fifth round. As only $|M|$ players can reach the fifth round in total, $m_1$ can win the tournament iff each player $m_j \in M$ can win a 16-player subtournament that consists exactly of the players stated in Lemma 2. We thus have to show that a draw in which each player $m_j \in M$ wins a 16-player subtournament with exactly the players stated in Lemma 2 exists iff the 3SAT instance is satisfiable.

$(\Leftarrow)$ Let the formula $F$ be satisfied by a truth assignment $A$. We construct a BKT over $N$ in which $m_1$ wins. For all $j$ where the $j$-th gadget is a choice gadget, we place all players from $\{m_j\} \cup G_j \cup S_j$ with the exception of two players from $S_j$, if $A(x_j) = true$ or $S_j$, if $A(x_j) = false$ in one subtournament for the first four rounds. These are exactly 16 players and by Lemma $2b)$, we can choose the draw in such a way that $m_j$ wins the subtournament.

Note that the choice gadgets use all players from $S$ except for two players of each set $S_{x_j}$ (or $S_{\bar{x}}_j$), where $x_j$ (or $\bar{x}_j$) is a literal that evaluates to true. In other words, for every literal that evaluates to true, two players remain that can be beaten by every player associated with a clause in which the literal players appear.

For the clause/garbage gadgets, to use Lemma $2b)$ we need for each $c/g$ player one additional player that he beats. If $c/g$ is associated with a clause $c_i \in C$, pick one of the literals in $c_i$ that evaluates to true, say $\bar{x}_j$, and assign one of the two players from $S_{\bar{x}_j}$ mentioned above to this $c/g$ player. Note that at most two such players are required for any literal, as every literal only appears twice. To all garbage players, assign an arbitrary remaining player from set $S$. Note that, after composing the subtournaments for all choice gadgets, $2|X|$ players from the set $S$ were left. As there are $2|X|$ $c/g$ players, we can assign a player from $S$ to each of these. For all $j$ where the $j$-th gadget is a clause/garbage gadget, we now place all players from $\{m_j\} \cup G_j$ in one subtournament for the first four rounds, together with the two players that are assigned to the $c/g$ players in $G_j$. By Lemma $2b)$, we can choose a draw such that $m_j$ wins this subtournament. For all $j$ where the $j$-th gadget is a filler gadget, we place all players from $\{m_j\} \cup G_j$ in one tournament for which, by Lemma $2b)$, we can find a draw where $m_j$ wins.

$(\Rightarrow)$ Let a draw be given such that all players in $M$ win a 16-player subtournament consisting exactly of the players stated in Lemma 2. First, note that whenever a player $s^*_i$ is not placed in the $i$-th choice gadget, we get a contradiction as by Lemma $2$, this player cannot be placed anywhere else while all players from $M$ still reach the fifth round. The following truth assignment is thus well-defined: $A(x_j) = true$ if $m_j$'s subtournament includes all players from $S_j$ except two players from $S_{x_j}$, $A(x_j) = false$ if $m_j$'s subtournament includes all players from $S_{\bar{x}_j}$ except two players from $S_{\bar{x}_j}$. We now show that $A$ satisfies all clauses. Let $c_i \in C$ be a clause and $c/g$ the player associated with that clause. Let $j^*$ be the 16-player subtournament including $c/g$. By the assumption, $m_{j^*}$ wins the subtournament, thus Lemma $2$ implies that it contains a player that $c/g$ beats, i.e., a player from set $S_{x_j}$ or $(S_{\bar{x}_j})$, where $x_j$ or $\bar{x}_j$ occurs in clause $C_i$. By the definition of $A$, that player can only be used in the subtournament if the corresponding literal evaluates to true (as otherwise, the player is used in $m_j$'s subtournament).

This concludes the proof of the theorem.

Implications of Theorem 1

Theorem 1 simultaneously implies a number of results from the literature, in particular Theorem 1, 2, and 3 in (Vu, Altman, and Shoham 2009) and Theorem 1 in (Vassilevska Williams 2010). It also yields some further corollaries. For PTFP and $\alpha \geq 1$, an algorithm is called an $\alpha$-approximation algorithm for the maximum winning probability if it can find a draw in which the winning probability of the given player $i$ is at least $\text{mpw}(i, N, P)/\alpha$.

Corollary 1. Unless $P = NP$, there exists no polynomial-time algorithm for PTFP that approximates the maximum winning probability of a player within any given factor.

It immediately follows from Theorem 1 that #TFP is NP-hard. We next highlight that even randomisation is not very helpful for #TFP. Let $\Gamma$ be a finite alphabet in which we agree to describe our problem instances and solutions. A fully polynomial time randomized approximation scheme (FPRAS) for a function $f : \Gamma^* \rightarrow \mathbb{Q}$ is an algorithm that takes input $x \in \Gamma^*$ and a parameter $\epsilon \in \mathbb{Q}_{\geq 0}$, and returns with probability at least $\frac{1}{2}$ a number between $f(x)/(1 + \epsilon)$ and $(1 + \epsilon)f(x)$. Moreover, an FPRAS runs in time polynomial in the size of $x$ and $1/\epsilon$. RP is the complexity class consisting of problems that can be solved in randomized polynomial time. The statement below follows from Proposition $8$ in (Welsh and Merino 2000) and Theorem $1$.

Corollary 2. Unless NP = RP, there is no FPRAS for #TFP.

Hence, unless NP = RP, there also does not exist an FPRAS for computing the probability of a player winning a BKT for a draw chosen uniformly at random.

Tractable Cases

We first examine a natural case in which players are divided into player types: All players of one type beat exactly the same subset of players of other types. The result of matches between players of the same type is irrelevant as we do not care which player in each type wins. In this variant that we call #TFP-types, the objective is to count the number of draws for which a player of a given type wins. We first adapt the concept of the pairwise comparison matrix. For two player types $i$ and $j$, define $P_{ij} = 1$ if an $i$-player wins a match between $i$ and $j$ and $P_{ij} = 0$ otherwise. Note that this definition is chosen such that $P_{ii} = 1$ for all player types $i$.

Theorem 2. #TFP-types are polynomial-time solvable if there are a constant number of player types.

Proof. We will use a dynamic programming scheme. All vectorial inequalities assume equal dimensions and are meant component-wise. Let $N_x := \{\eta = (\eta_1, \ldots, \eta_k) \geq 0 | \eta_1 + \cdots + \eta_k = 2^x\}$ and denote by #TFP($i, x, \eta \in N_x$) the number of draws for an $x$-round tournament involving $\eta_j$ players of type $j$ for all $j \in [k]$ in which a player of type $i$ wins the tournament. We will assume that the players of one type are not distinguishable. For fixed $i$ and $x$, there are
Thus, \( \#\text{TFP}(i, x, \eta) \) is computed via the following recursion.

\[
\#\text{TFP}(i, x, \eta) = \sum_{\eta' \in N_{i-1}} \#\text{TFP}(i, x - 1, \eta') \cdot \sum_{j \in [k]} \#\text{TFP}(j, x - 1, \eta - \eta')
\]  \hspace{1cm} (3)

The base cases are given by

\[
\#\text{TFP}(i, 0, \eta) = \begin{cases} 1 & \text{if } i = 1 \text{ and } \forall j \neq i : \eta_j = 0 \\ 0 & \text{otherwise} \end{cases}
\]  \hspace{1cm} (4)

for all \( i \in [k] \). Eq. (3) only uses values of \( \#\text{TFP}(j, y, \eta') \) with \( y < x \), thus every problem only needs to be solved once. For constant \( k \), solving a problem requires \( O(n^k) \) operations and for constant \( k \) it holds that \( \binom{k+2^n}{k} \in O(n^k) \). Thus, \( O(\log(n)n^{2k}) \) operations are necessary to compute \( \#\text{TFP}(i, \log(n), (r_1, \ldots, r_k)) \).

The second tractable case that we identify is:

**Theorem 3.** \( \#\text{TFP} \) is polynomial-time solvable if there is a linear ordering of strengths of the players with at most a constant number of pairwise relations flipped.

**Proof.** Let \( b \) be the number of backwards arcs that do not respect the linear ordering. We can show by induction that the problem can be reduced to \( \#\text{TFP} \)-types with at most \( 4b + 3 \) player types. Since the number of player types is constant, the theorem follows from Theorem 2. \( \square \)

### An Exponential Time Algorithm for \( \#\text{TFP} \)

\( \#\text{TFP} \) can be trivially solved in time \( 2^{O(n \log n)} \) via a brute-force enumeration of all possible draws. In exponential time algorithms (Fomin and Kratsch 2010), the aim is to design algorithms solving the problem exactly with worst-case running times outperforming the brute-force solution. In this section we give an algorithm for \( \#\text{TFP} \) running in time \( O(2.8285^n) \) using space \( O(1.7548^n) \). If only polynomial space is available, the running time becomes \( 4^{n+o(n)} \), and we give a range of possible time-space trade-offs. The algorithm is based on the recursion formula (3) and memoization used at various levels of the recursion. We use \( \text{poly}(n) \) to stand for a polynomial function in \( n \).

**Theorem 4.** For every \( y, 2 \leq y \leq \log n \), there is an algorithm solving \( \#\text{TFP} \) in time \( T(n) = \text{poly}(n) \cdot \prod_{y=0}^{y-1} \frac{2^n}{2^n} \cdot 2^{n/2^y} \) and space \( S(n) = \text{poly}(n) \cdot (2^n)^{n/2^y} \cdot \left( \frac{2^y}{2^y-1} \right)^{n-n/2^y} \).

**Proof Sketch.** The algorithm recursively evaluates the formula (3) for the special case where each player type consists of one player, starting from \( \#\text{TFP}(i, \log n, (1, \ldots, 1)) \). It uses memoization for all recursive calls \( \#\text{TFP}(\cdot, x, \cdot) \) where \( x \leq (\log n) - y \). That is, the algorithm uses a table indexed by players, level, and player type vector. To evaluate \( \#\text{TFP}(\cdot, x, \cdot) \) with \( x \leq (\log n) - y \), the algorithms first checks in this table whether this recursive call has already been evaluated. Only if the value has not yet been computed, it computes the result recursively, and stores it in the table. Then, it returns the result that is stored in the table. The number of table entries used by memoization is upper bounded by \( n \) times the number of subsets of size at most \( n/2^y \). Using Stirling’s approximation for factorials, the space usage of the algorithm can be upper bounded by \( S(n) = \text{poly}(n) \cdot (2^n)^{n/2^y} \cdot \left( \frac{2^y}{2^y-1} \right)^{n-n/2^y} \).

The running time of the algorithm is the time used for the recursion without memoization, \( \text{poly}(n) \cdot \prod_{y=0}^{y-1} \frac{2^n}{2^n} \cdot 2^{n/2^y} \) plus the time for the part with memoization, which is upper bounded by \( S(n) \cdot n \cdot 2^{n/4} = O(2.0868^n) \). Thus, for any \( y, 2 \leq y \leq \log n \), we obtain an algorithm with running time \( T(n) = \text{poly}(n) \cdot \prod_{y=0}^{y-1} \frac{2^n}{2^n} \cdot 2^{n/2^y} \) using space \( S(n) = \text{poly}(n) \cdot (2^n)^{n/2^y} \cdot \left( \frac{2^y}{2^y-1} \right)^{n-n/2^y} \).

Various time and space requirements of the algorithm are reported in Table 1. Using the rule of thumb that for current computing architectures, the space requirements of an algorithm become a bottleneck if they exceed the square root of the time requirements, the analyses for \( y = 2 \) and \( y = 3 \) currently seem the most relevant.

<table>
<thead>
<tr>
<th>( y )</th>
<th>( \text{time } T(n) )</th>
<th>( \text{space } S(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>( O(2.8285^n) )</td>
<td>( O(1.7548^n) )</td>
</tr>
<tr>
<td>3</td>
<td>( O(3.3636^n) )</td>
<td>( O(1.4576^n) )</td>
</tr>
<tr>
<td>4</td>
<td>( O(3.6681^n) )</td>
<td>( O(1.2634^n) )</td>
</tr>
</tbody>
</table>

**Table 1:** Running times and space requirements for the algorithm of Theorem 4 for various time-space trade-offs.

In this paper we considered problems related to tournament fixing. Although being able to change draws is not always realistic, the computational problems that are considered have been analyzed in post analysis of tournament draws and also shed light on the relative strengths of players. Our main result is that \( \text{TFP} \) is NP-complete. We discussed a number of implications of the result. We complement the computational hardness result in the paper by presenting algorithms for \( \text{TFP} \) — both for the general case as well as restricted cases. A possible future direction is to propose parametrized algorithms for \( \text{TFP} \).

**Conclusions**

**Acknowledgments**

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. Serge Gaspers is the recipient of an Australian Research Council Discovery Early Career Researcher Award (project number DE120101761).
References


