Topics in Fixing Knockout Tournaments: Bribery, NP-Hardness, and Parameterization

by

Christine Konicki

B.S., Massachusetts Institute of Technology (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 7, 2019

Author . . . .
Department of Electrical Engineering and Computer Science
May 21, 2019

Certified by . . . .
Virginia Vassilevska Williams
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Topics in Fixing Knockout Tournaments: Bribery, NP-Hardness, and Parameterization

by

Christine Konicki

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2019, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, we analyze the problem of fixing balanced knockout tournaments by arranging the tournament's initial seeding to guarantee one player's victory. We characterize the problem's hardness through a variety of perspectives.

First, we investigate the computational complexity of fixing the tournament while bribing players to lose matches they would typically win. We give a model of bribery in which one is given a pairwise comparison matrix whose entries contain the probability of the row player beating another column player in a match, where the organizer's ability to bribe players is constrained by the cost of each bribe and a fixed budget, and where the tournament seeding can be manipulated arbitrarily. We show that it is NP-hard to find a bribery scheme and a seeding under which a given player always wins the tournament, even when the original pairwise comparison matrix is monotonic; the hardness of fixing a tournament in this case without bribery is open. We also show that when the probability matrix is binary, for almost all n-player inputs generated by the Condorcet random model, if one bribes a specific number of the "top" $O(\log n)$ players, then there is an efficiently constructible winning seeding for any player.

Next, we investigate the relationship of the deterministic case of the tournament fixing problem to other NP-complete problems. We demonstrate the futility of constructing a reduction to it from certain well-known graph problems, showing why the features of these problems are ultimately incompatible. We analyze the blowup of the NP-hardness reduction to the problem from a restricted version of 3SAT and use it to give a direct NP-hardness reduction from 3SAT. Finally, we apply parameterized complexity to the deterministic tournament fixing problem, giving a simpler algorithm that matches the runtime of the fastest known algorithm, using the size of the input tournament graph's feedback arc set as a parameter.

Thesis Supervisor: Virginia Vassilevska Williams
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

This thesis is the culmination of my industry and research at MIT. It is a true intellectual marathon, if ever there was one. These are the people who made it all possible.

First and foremost, I would like to thank God and His Son for being at my side and silent in the background as I undertook this project, even when my back was turned. Thank You for giving me the brain, the curiosity, and the steel backbone that have spurred me forward for the past twenty-three years. This achievement would not have happened if not for your blessings.

A huge, huge thank you to my supervisor Virginia Vassilevska Williams for taking a chance on me. Through her, I have learned more about theoretical computer science, algorithms, and computational complexity than I ever imagined. She taught me to not second-guess myself when I was afraid of failure or risks, to not give up when I felt like a fumbling impostor, and to take myself seriously as a researcher even when I knew so little. She also fed me delicious chocolate during our weekly meetings, especially when she could tell that I was stressed out. Under her guidance, the idea machine that is my thesis transformed from a callow blob into a consort battleship and yielded our first publication together at AAMAS.

Thank you to Matthias Mnich from the University of Bonn's Institute of Computer Science and to Josh Alman from MIT's Theory Group for taking an interest in my work and for spending hours puzzling with Virginia and me. Thank you to Michael Sipser and Ryan Williams; my interest in this field skyrocketed after taking their respective classes, 6.840 (Theory of Computation) and 6.841 (Advanced Complexity Theory). I would not have undertaken this project if not for their passion for the material and their skill at teaching it. Thank you to my academic advisor Yury Polyanskiy for guiding me at the beginning of every semester to make sure I was on track to finish my degree on time. Thank you also to my freshman advisor Wayne Johnson for his smile and his unfailingly positive energy whenever we bumped into each other along the Infinite Corridor or on Massachusetts Avenue.

This degree would not have happened if not for Chris Terman, Adam Hartz, and Mardavij Roozbehani; thanks to them, I have had the opportunity to teach every single semester, and I would not have been able to finish this degree in a single contiguous run (or support myself) otherwise. Thank you to the 2017-2018 6.004 staff (Chris, Daniel, Silvina, Malek, Olivia, Hyung, Kat, Ariel, Rakesh, Andrew, Aaron, Rusch, Douglas), the Fall 2018 6.009 staff (Adam, Duane, Karl, Peter, Jeremy, Cavin, Amir, Jesse, Dylan, Luana, Joseph, Jisoo, Steven, Sunayana, Heather, Maha), and the Spring 2019 6.041 staff (Mardavij, Lizhong, Hajir, Sam, Bomin). Teaching with them was a true pleasure. Enormous thanks to all of my students (especially the ones I had more than once) for making it worthwhile, fun, and so much more than a job.

Thank you to all of the graduate students I have gotten to know: Andrea Lincoln, Dylan McKay, Luke Schaeffer, Siddartha Jayanti, Daniel Grier, Dhiraj Holden, Adam Sealfon, Andrew Xia, Josh Alman, Mina Dalirrooyfard, Quanquan Liu, Jayson Lynch, Govind Namarayan, and Nicole Wein. I am grateful to them for spending time with me in the open spaces of the sixth floor of Gates because I didn't have an office, and for answering my questions/hearing me rant when I didn't know what I was doing. Thank you to Daniel Grier also for helping me draft my thesis proposal, and to Mina Dalirrooyfard for sending me the template.

Thank you to all past and current members of TEP for being my family for part of my time as a graduate student. I especially want to single out some special people: Ivan for commiserating with me ever since we met as freshmen at the East Side Party; Bob, Carrie, and Skirma for making me laugh and forget about life for a while; Tuan, Dain, Dayna, and Miguel for always being contagiously adorable and sweet; Sara for talking sense into me; Jabari, Gabe, Meital, and Robby for keeping the door to 33 open; Jetson for helping me stay grounded about the future; and Blue for being my go-to person whenever I needed some understanding and good music. And, of course, thank you to the feline residents Eliot and Yashy for being reliable cuddlebugs.

Thank you to Barbie, Austin, Amelie, J, Dan, DGonz, and Adam for still being in Cambridge after all this time and for wearing far too many hats to count over the course of our friendship. For the past six years, they have all shared with me, hugged me, called me out on my faults, loved me, and grown with me–both up close and from afar. A gargantuan thank you to my best friend Nette for supporting me, listening to my (sometimes) annoying rambles, and being by my side in all of our battles together, even from Europe. Watching you grow and change with me has been an adventure, and I am grateful for you every day. May we continue to finish races together when we have hit our nineties, chafed thighs and all. Also, IngaBazinga!

Thank you to Chris, Lilly, Tim, and Pooka for being my housemates outside of MIT and for listening to me rant whenever I hit a low point. Thank you to the Boston chapters of Anonymous for the Voiceless and The Most Informal Running Club Ever for their interest in my work; answering your questions helped me stay focused and motivated. Thank you to Nishant, Laura, Michelle, Jon, Carolyn, Greg, Andi, Dahlia, Ivanna, Ebony, Eli, Lori, Sarah, Stefanie, and Larken for your capital activism and warm friendship.

Thank you to Boston and its surrounding metropolitan areas. I love this magical city like my own flesh, and I will always treasure it as a locus of profound growth in my life. There truly is no other city like it if you are a patriot, a runner, a student, or some combination of the three.

Finally, I want to mention that toward the end of 2018, part of my personal life imploded in spectacularly ugly fashion. To say “I hit a wall” or even “IHTFP” would be gross understatements. In retrospect, I almost lost my mind. Thank you to Steve Brown for dotingly taking me in even though we had never met. Thank you to Alex and Becca Brown for also being there, patiently listening to me when I was at my lowest. And a thousand times, thank you to Matt Brown for being my rock and champion during that time and ever since. You have been an inspiring fellow runner, a gentle catalyst for my renewed walk with God, a consistent source of kindness....put simply, the most unexpected and effervescent person at the most unexpected of times. Having my back against the warm, sure wall of your support has truly been a blessing, and this thesis would not have been finished without it. Because I can never thank the four of you enough, and words spoken only last for as long as memory allows, I dedicate this dissertation in perpetuity to you.

"Keep coming, and you sweep floors and scrub out ashtrays and fill stained steel urns with hideous coffee, and you keep getting ritually down on your big knees every morning and night asking for help from a sky that still seems a burnished shield against all who would ask aid of it...but the old guys say it doesn't yet matter what you believe or don't believe, Just Do It they say, and like a shock-trained organism without any kind of independent human will you do exactly like you're told, you keep coming and coming, nightly."

—David Foster Wallace, *Infinite Jest*

"The place of true healing is a fierce place. It's a giant place. It's a place of monstrous beauty and endless dark and glimmering light. And you have to work really, really, really hard to get there, but you can do it."

—Cheryl Strayed

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Knockout tournaments are a popular format for determining a winner from several choices. First, a specific permutation of the choices called the *seeding* is selected. Then, the tournament proceeds in sequential rounds. Each successive round uses a restricted subset of the current seeding, which contains only the alternatives that were not eliminated in previous rounds. The alternatives are then paired together: the first with the second, the third with the fourth, and so on. This ordered matching is known as a pairwise comparison between the alternatives. Only the winners of the current round's pairwise comparisons move on to the next round. Only one alternative reaches the final round, winning the tournament.

The knockout tournament structure is popular in sports competitions such as Wimbledon or the World Cup (where the alternatives are teams or players) and in social choice, as in the binary cup voting rule. Due to the sports connection, alternatives are often referred to as players, and pairwise comparisons as matches that the players win or lose. The tree that results from the sequential pairwise comparisons is known as the *bracket*. The popularity of these tournaments stems primarily from their lucidity, as the structure is simple and efficient once the players are arranged. More importantly, other social choice mechanisms can unintentionally give the players an incentive to attempt to "game the system" by engaging in unwanted behavior;

knockout tournaments have no such bug. Players in knockout tournaments are not on teams, so there is no reason for the players to work together. At best, they would be competing against each other in the penultimate round, which is not mutually beneficial as there is no reward for the loser who made it that far and then lost. Moreover, players who lose on purpose in a single match will be eliminated immediately instead of receiving a lower rank or a lower score, so there is no incentive for the players not to try their best.

Within computational social choice, several works investigate how much power a tournament organizer has in influencing the outcome of the tournament by selecting the initial seeding. This is the agenda control problem for knockout tournaments, also called the *Tournament Fixing Problem* (TFP): Can a self-interested manipulator set the initial seeding so that their favorite player has the best chance of winning? Stated more formally:

Definition 1.1.1. *The* **Tournament Fixing Problem** *(TFP) takes as input a rational number $\delta \in [0, 1]$, a favorite player $i^* \in [n]$, and an $n \times n$ matrix P that gives for every two players $i, j \in [n]$, the probability $P_{i,j}$ that i is preferred over j (i.e. that i* beats *j); and asks whether or not there exists a seeding s for which i^* wins the balanced knockout tournament with probability at least δ?*

The computational problem of fixing tournaments is interesting because its complexity can be used to show how difficult it really is for the organizer to find a permutation that would yield the outcome he desired, should he wish to underhandedly influence the results ahead of time. By extension, this demonstrates how much power the organizer has over who ultimately wins the entire tournament.

Tournaments in which n is a power of 2, known as *balanced knockout tournaments* and pictured in Figure 1-1, comprise a common format for sporting competitions, elections, and pairwise decision-making. In this thesis, we investigate the computational complexity of arranging the tournament's initial seeding using several techniques and problem variations.

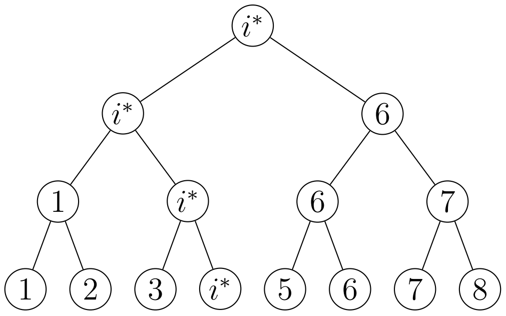

Figure 1-1: A balanced knockout tournament with $n = 8$ players and favorite player $i^* = 4$. The results by round are displayed as a binary tree.

1.2 Terminology and Notation

All tournaments in this thesis contain n players, where n is a power of 2. Balanced knockout tournaments adopt the structure of a complete binary tree. The arrangement of the leaf nodes represents the *seeding*, and players that correspond to sibling leaf nodes compete against each other in a match. The winner of each match moves up in the tree to the next round. The winner of the tournament is the player who reaches the root node. The potential match outcomes are described by a probabilistic pairwise comparison matrix $P \in \mathbb{Q}^{n\times n}_{[0,1]}$ where $P_{i,j}$ denotes the probability that player i will beat player j in a match, and $P_{i,j} + P_{j,i} = 1$. P can be used to compute the probability that a player i wins the tournament using dynamic programming given any initial seeding. We define various restrictions applied to P in this thesis.

Definition 1.2.1. *A probabilistic matrix P is* **monotonic** *if it satisfies the following conditions ([14], [15], [20]):*

1. $P_{i,j} \geq P_{j,i} \forall (i,j) : i \leq j$,

2. $P_{i,j} \leq P_{i,(j+1)} \forall (i,j)$.

Definition 1.2.2. *A probabilistic matrix P is* **ϵ-monotonic** *for $\epsilon \geq 0$ if it satisfies the following more relaxed conditions ([26]):*

1. $\forall (i,j) : i \leq j, P_{i,j} \geq P_{j,i}$;

2. $\forall (i,j,j') : j' > j, P_{i,j} \leq P_{i,j'} + \epsilon$.

Definition 1.2.3. *A probabilistic matrix P is* **deterministic** *if* $\forall(i,j) : i \neq j, P_{i,j} \in \{0,1\}$. *A deterministic probabilistic matrix can also be represented as a tournament graph by putting a directed edge from player i to player j whenever $P_{i,j} = 1$ for each (i,j).*

Table 1.1 summarizes some of the notation used to refer to players and their neighbors according to deterministic matrix P. Another common concept applied to tournament graphs is the notion of a *king*.

Notation
$N_{out}(v) = \{u : P_{v,u} = 1\}$
$N_{in}(v) = \{u : P_{v,u} = 0\}$
$out(v) = \lvert N_{out}(v)\rvert, out_X(v) = \lvert N_{out}(v) \cap X\rvert$
$in(v) = \lvert N_{in}(v)\rvert, in_X(v) = \lvert N_{in}(v) \cap X\rvert$

Table 1.1: Summary of the notation used in this paper for tournaments with deterministic P, i.e. tournament graphs.

Definition 1.2.4. *A player u is a* **king** *if for every other player v in the tournament, either $P_{u,v} = 1$ or $P_{u,w} = 1$ and $P_{w,v} = 1$ for some w in the tournament.*

1.3 Outline

In Chapter 2, we introduce into TFP the ability of the organizer to bribe players to lose to each other. We formulate a model of this bribery in which one is given a matrix with the entries equal to the probability of each row player beating the pairwise column player. Using this matrix, one can bribe players to decrease their probability of beating other players at a cost, without exceeding a budget, and where the tournament seeding can be manipulated arbitrarily.

We show that it is NP-hard to determine a bribery and a seeding under which a given player is guaranteed to win the tournament, even when the initial probability matrix is monotonic, and the probability matrix after bribery is ϵ-monotonic and very close to the initial one. In contrast, it is not known whether it is NP-hard to find a winning seeding for a given player without bribery when the probability matrix is monotonic. We also consider the case where the input is deterministic; that is, the probability matrix is binary, and one can bribe any player to assuredly lose to any other player. We show that for almost all possible n-player inputs generated by a well known model due to Condorcet, if one bribes the "top" $O(\log n)$ players, then there is an efficiently constructible seeding for which *any* player wins.

For the remaining chapters, TFP will be used specifically to refer to the case of deterministic TFP. In Chapter 3, we examine the relationship of TFP to other NP-complete problems. We show that the construction of a reduction from $CLIQUE$ or VC to TFP is infeasible and very unlikely to appear in the future. We summarize recent past work demonstrating why deterministic TFP is NP-complete using $3SAT2$ and analyze the implications of its blowup in problem size. We also create a direct reduction from $3SAT$ to TFP and then analyze its efficiency.

In Chapter 4, we examine the application of parameterized complexity to TFP. We outline our motivations for using parameterized complexity to reframe the runtime of the fastest parameterized algorithm for solving TFP. We then summarize this algorithm and give a more elegant one that matches the runtime. We also explain the hindrances indicating that this runtime is unlikely to drop below $2^{k \log k} n^{O(1)}$, where n is the number of nodes in the tournament graph and k the size of the feedback arc set in the input tournament graph.

In Chapter 5, we conclude and discuss several paths for future work on this problem and its variants.

Chapter 2

Bribery in Knockout Tournaments

2.1 Introduction

In this section, we consider the TFP problem when the tournament organizer is also allowed to bribe players. Notions of bribery in knockout tournaments have been studied before, e.g. by [13], [17], [19], and [22]. The results in this thesis extend what is known in novel ways.

We first formulate a new bribery version of TFP, extending a notion from [17]. The motivation of our model is as follows. Suppose we know from our input matrix P the probability $P_{i,j}$ that row player i will beat column player j. Suppose also that at a cost $C_{i,j}$ to the organizer, column player j will intentionally play worse, and will lose to row player i with probability $A_{i,j} > P_{i,j}$, where A and C are also $n \times n$ matrices. Now, as in TFP, we want our favorite player i^* to win the tournament. We have a budget B of how much money we want to spend on bribes. We want to bribe without exceeding the budget so that we can find a seeding for which i^* wins with probability at or above a given threshold δ, with the matches playing out according to the pairwise winning probabilities resulting from the bribes. Let us call this problem the *Tournament Fixing Problem with Bribery* (BTFP), which will be defined more formally in subsection 2.2.1. On input $(i^*, P, A, B, C, \delta)$, BTFP returns "yes" if and only if there is a bribery strategy and a seeding so that i^* wins with probability at least δ.

It is not hard to see that BTFP is NP-hard. All we have to do is reduce TFP to it: Given an instance (i^*, P, δ) of TFP, create an instance of BTFP $(i^*, P, A, B, C, \delta)$ by setting $B = 0$, $A = P$, and C arbitrarily. This transformation still works if P is structured. For instance, BTFP is NP-hard even if P is ε-monotonic due to the result of Vu, Altman, and Shoham (2009) [25].

We give a hardness result for BTFP for the case when P is monotonic. This does not follow immediately from the hardness of TFP, however, as TFP is not known to be NP-hard when P is monotonic. The monotonic case models the real world, and hence it is of interest since beating a stronger player is indeed harder than beating a weaker one. However, because a manipulator performing bribes would eschew getting caught, the matrix resulting from the bribes P' should ideally be almost monotonic and nearly identical to P.

Our first theorem is that BTFP is NP-complete, even if P is monotonic and no entry of the resulting probability matrix P' after bribery differs from the corresponding entry of P by more than an arbitrarily small threshold ε. The last assertion implies that P' must be ε-monotonic.

Next, we consider the case where for every pair of players i and j, the manipulator knows which one of them will win in a match. This corresponds to having P be a binary matrix. Bribery problems in this case of deterministic P have been proposed in the past in various formats. One version showed that given a seeding, one could efficiently find a set of matches such that a favorite player could win the tournament if the intended winner of each match according to P lost [22]. One could easily view this set of thrown matches as a set of bribes. [13] similarly applies polynomial-time operations to an existing winning seeding that was too obviously rigged to find a new ordering and a new set of bribes. However, since finding a winning seeding is typically a hard problem, it is much more interesting to incorporate bribery into the computational process of finding a winning seeding instead of working from an existing one.

Another case for deterministic P, Bribery-TFP, was proposed by Kim and Vassilevska W. [17]: Given P, a budget B, and i^*, determine if one can bribe at most

B players to each lose a prescribed match so that i^* wins the tournament. Notice that Bribery-TFP is the special case of BTFP in which P is binary, and the entires of both A and C are all 1's. [17] showed that this bribery problem is NP-hard when $B = (1-\varepsilon)\log n$ for any constant $\varepsilon > 0$. This is essentially tight since $\log n$ bribes will always suffice to fix i^* to be the winner: For any initial seeding and in each successive round, bribe the player that i^* is supposed to play.

However, there is an issue with this approach to bribing. The players that are bribed truly depend both on i^* and the given seeding, so the tournament outcome might look a bit suspicious, yielding the problem that [13] works to correct retroactively. Avoiding this problem in the first place would be preferable. Moreover, if we bribed some players to make i^* win for one seeding but decide later that we want some other player to win instead, we might have to bribe anew. It would be great if we could have a handful of players "in our pocket." With these players guaranteed to succumb to any bribe, neither the initial seeding nor the choice of which player we want to win will matter. We can just give the players in our pocket a match to throw, and then our favorite player will win. The notion of bribing only an elite set of players to throw matches in a tournament has been explored by [13], but the players that comprise that elite set vary depending on which player is the favorite and on the structure of P.

There is also an issue with the hardness results for both [13] and [17]. In most applications, the input matrix P is not arbitrary: For instance, stronger players tend to beat weaker players. Perhaps one can exploit this natural structure of P to bribe better. Indeed, our second result shows that if one assumes that P is generated from a standard generative model that follows this pattern (originally due to Condorcet), then one can always bribe the top $O(\log n)$ players so that regardless of which i^* we pick and for almost all P generated this way, one can efficiently find a winning seeding for i^* if the bribed players throw a match of our choosing. This result is intriguing. One can imagine for instance the tournament organizer paying the registration fees for the top $O(\log n)$ players, with the implicit understanding that if he asks, they have to return the favor and throw a match. Using this strategy, the tournament organizer can make even the weakest player win.

The Condorcet model is very natural: Given some "error" probability $p > 0$, there is a total ordering of the players 1 to n, where players with lower numbers are stronger than players with higher numbers. Additionally, for each $i < j$, a weaker player j beats a stronger player i with (small) probability $p \leq 1/2$; otherwise, the stronger player beats the weaker one with probability $1 - p$.

To gain some intuition of why a result such as this might be possible, imagine that $p = 0$. It follows that if $i < j$, then i always beats j. In this case, imagine that we bribe the top $\log n$ players to lose to the very weakest player n if they play against n for any round in the tournament. Then, n is a "superking" (as defined in [24]) because for every player j that n cannot beat, there are $\log n$ players that n beats and who beat j. [24] showed that for every superking, there exists an easily computable seeding for which the superking wins the tournament. When $p > 0$, the top $\log n$ players lose some matches with probability p, and we need to bribe slightly more than $\log n$ players to make sure that the weakest player can win. We provide a tight bound on just how many players we need to bribe.

2.2 NP-Completeness of BTFP

2.2.1 Bribery Model

We define a model for bribing players competing in a balanced knockout tournament. It is assumed that for any pairwise match, the probability of one player winning against the other is known, regardless of whether or not a bribe was made. We are given the following as inputs:

- A set of n players $S = [n]$.
- A favorite player $i^* \in S$.
- A probabilistic pairwise comparison matrix $P \in \mathbb{Q}_{[0,1]}^{n \times n}$.
- A matrix $A \in \mathbb{Q}_{[0,1]}^{n \times n}$ where $A_{i,j}$ denotes the probability that row player i will

beat column player j in a match that j has been bribed to throw. For all (i, j), $P_{i,j} \leq A_{i,j} \leq 1$.

- A budget $B \in \mathbb{Z}^+$, the total amount of money that can be spent on bribing players.
- A cost matrix $C \in \mathbb{Z}_+^{n \times n}$ where $C_{i,j}$ denotes how much column player j must be bribed to agree to throw the match to row player i.
- A threshold probability $\delta \in \mathbb{Q}^{[0,1]}$.

Note that when a player j is bribed to lose to a player i, the probability given by $A_{i,j}$ is used instead of the original probability given by $P_{i,j}$ to gauge who wins the match. The complement probability $P_{j,i}$ is likewise replaced with $1 - A_{i,j}$. We assume that we do not have to make two bribes in order to change both entries in P. With this model, we define the problem:

Definition 2.2.1. The Tournament Fixing Problem with Bribery (BTFP)*: "Given S, i^*, P, A, B, C, and δ as in our model above, decide whether there exists a seeding and bribery choices for a balanced knockout tournament described by P such that i^*'s probability of winning the tournament can be raised above δ by bribing specific players according to C without exceeding budget B."*

Note that there are two specific cases of P in which we apply the bribery model. In the *monotonic bribery model* for BTFP, P is strictly monotonic. In the *deterministic bribery model* for BTFP, P and A are deterministic, and for all (i, j), $A_{i,j} = C_{i,j} = 1$.

2.2.2 Hardness Proof

We state and prove the complexity of BTFP. It is not hard to show NP-completeness when P is ϵ-monotonic. Set $A = P$, $B = 0$, and $C = \mathbf{0}_{n,n}$. This makes BTFP equivalent to an instance of the tournament fixing problem without bribery, which has been proven to be NP-complete when P is ϵ-monotonic [26]. However, it is not known if TFP is NP-complete for monotonic P, so we have to find another approach in this case. We show that BTFP is NP-complete if P is restricted to being monotonic by demonstrating how to bribe so that the resulting matrix P' after bribery is ϵ-monotonic.

Theorem 2.2.2. *BTFP is NP-complete even if P is monotonic and the matrix P' after bribery is ϵ-monotonic.*

Proof. It is easy to see that BTFP $\in$ NP. A possible solution would consist of an initial seeding and denotations for what probabilities in P would be replaced by probabilities in A, depending on which of the associated matches involved bribes. A polynomial-time verification algorithm would consist of computing i^*'s new probability of winning the tournament with dynamic programming, comparing the result with δ, and checking that the total bribery cost does not exceed B.

To prove that BTFP is NP-hard, we reduce from *Vertex Cover* (VC): "Given a graph $G = (V, E)$ and an integer k, is there a subset $V' \in V$ such that $|V'| \leq k$ and every edge in E has at least one vertex in V' ?" We will show that one can choose an initial seeding for a tournament described by P with players that must be bribed so that a special player i^* will always win if and only if G has a vertex cover of size at least k. Our proof is based on past work proving that TFP is NP-complete when P is ϵ-monotonic by means of a reduction from VC [26].

Given an instance (G, k) of VC, we construct an instance of BTFP with the following players:

- Vertex players $\{v_i \in V\}$ where $|V| = n$. Also, a special new player $v_0 \notin V$ which does not cover any edges.
- Favorite player $v^* \notin V$, newly created.
- Edge players $\{e_i \in E\}$ where $|E| = m$.
- Filler players for vertex players: For each $v_i \in V \cup \{v^*, v_0\}$ and each $r \in \mathbb{N}$ such that $0 < r \leq \lceil \log(n-k) \rceil$, we have k filler players $\{f^r_{v_i}\}$.
- Filler players for edge players: For each $e_i \in E$ and each $r \in \mathbb{N}$ such that $\lceil \log(n-k) \rceil < r \leq \lceil \log(n-k) \rceil + \lceil \log(m) \rceil$, there are k filler players $\{f^r_{e_i}\}$.

- Holder players for edge players: For each $e_i \in E$, there are $2^{\lceil \log(n-k) \rceil} - 1$ associated edge holder players $h^t_{e_i}$.
- Holder players for filler players: For each filler player $f^r_{v_i}$ or $f^r_{e_i}$ that must be placed at round r (denoted f^r_i), there are $2^r - 1$ holder players $h^t_{f^r_i}$.
- Holder players for v^*. There are $2^N - 1$ special holder players in the tournament where $N = \lceil \log(n-k) \rceil + \lceil \log(m) \rceil + \lceil \log(k+1) \rceil + 1$.

Let P be as in Table 2.1, and let A be the matrix that is the same as P' in Table 2.2 above the diagonal and the same as P below the diagonal. Let W be the set of all players. Set B to be $|W|^2$, which is the size of P. For each pair (i, j) with $i < j$ (above the diagonal) in which P and P' (from the tables) differ, set the associated cost $C_{i,j} = 1$. For each remaining pair, set the associated cost $C_{i,j} = B + 1$.

The colored cells in the aforementioned tables denote the bribes that can be made to the column player to give the row player a quantitative advantage in the match. If all such bribes are made, P' is an ϵ-monotonic matrix of winning probabilities.

	v^*	v_j	e_j	$f_j^{r'}$	$h^{t'}_{e_j}$	$h^{t'}_{f_j^{r'}}$	$h^{t'}_*$
v^*	$-$	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$
v_i	ϵ	$1-\epsilon$ if $i < j$, ϵ o.w.	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$
e_i	ϵ	ϵ	ϕ	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$
f_i^r	ϵ	ϵ	ϵ	ϕ	ϕ	ϕ	$1-\epsilon$
$h^t_{e_i}$	ϵ	ϵ	ϵ	ϕ	ϕ	ϕ	$1-\epsilon$
$h^t_{f_i^r}$	ϵ	ϵ	ϵ	ϕ	ϕ	ϕ	$1-\epsilon$
h^t_*	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϕ

Table 2.1: Matrix P pre-bribery. The bribes are indicated by the colored cells

Lemma 2.2.3. *($\leftarrow$): If G has a vertex cover of size k, we can construct a seeding and players that must be bribed so that v^* will win with probability 1 using substitute probabilities from A that form an ϵ-monotonic P'.*

Proof. Given a graph G with a vertex cover V' of size k, we can create a seeding for which v^* wins with probability 1. As mentioned earlier, Table 2.1 gives the initial monotonic winning probabilities of P. The colors denote the entries that will be

	v^*	v_j	e_j	$f_j^{r'}$	$h^{t'}_{e_j}$	$h^{t'}_{f_j^{r'}}$	$h^{t'}_*$
v^*	$-$	1	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$	1
v_i	0	1 if $i<j$, 0 o.w.	1 if v_i covers e_j, $1-\epsilon$ o.w.	1	$1-\epsilon$	$1-\epsilon$	$1-\epsilon$
e_i	ϵ	0 if v_j covers e_i, ϵ o.w.	ϕ	1	1 if $i=j$, $1-\epsilon$ o.w.	$1-\epsilon$	$1-\epsilon$
f_i^r	ϵ	0	0	ϕ	ϕ	1 if $f_i^r = f_j^{r'}$, ϕ o.w.	$1-\epsilon$
$h^t_{e_i}$	ϵ	ϵ	0 if $i=j$, ϵ o.w.	ϕ	ϕ	ϕ	$1-\epsilon$
$h^t_{f_i^r}$	ϵ	ϵ	ϵ	0 if $f_i^r = f_j^{r'}$, ϕ o.w.	ϕ	ϕ	$1-\epsilon$
h^t_*	0	ϵ	ϵ	ϵ	ϵ	ϵ	ϕ

Table 2.2: Matrix P' post-bribery. The bribes are indicated by the colored cells

changed as a result of bribery. Note that $\epsilon \leq \phi \leq 1-\epsilon$. We bribe players so that our original matrix of winning probabilities P becomes the matrix P' containing our preferred probabilities given by the matrix in Table 2.2, with the same colors as before denoting the bribes. Again, note that $\epsilon \leq \phi \leq 1-\epsilon$. All colored entries can be bribed within our budget. Now, after we perform these bribes, the matrix P' from Table 2.2 is exactly the matrix that is obtained in [26] in the reduction from VC to TFP with an ϵ-monotonic P. Thus, we can just refer to the argument in [26] that shows how to construct a seeding so that v^* wins. □

Now we get to the crux of our proof, that in order for v^* to win with probability 1, the bribes chosen above must be made, and a small vertex cover must be recovered.

Lemma 2.2.4. *($\rightarrow$): If there is a bribery strategy and initial seeding such that v^* wins with probability 1, we can find a vertex cover in G of size k.*

Proof. We will show that all bribes in Tables 2.1 and 2.2 are necessary for v^* to win with probability 1. First, we examine the filler players f_i^r and their associated holder players $h^t_{f_i^r}$, proving that for v^* to win the tournament, the filler players must advance until round r exactly, at which point they are eliminated.

Claim 2.2.5. *Every filler player f_i^r can survive with probability 1 only as far as round r.*

Proof. Suppose that a filler player f_i^r survives until round t with probability 1. Then in each of the rounds $q < t$, f_i^r must have been matched to some holder player $h_{f_i^r}^q$ with probability 1 since this is the only matching that guarantees victory for f_i^r. Once bribes have been made, the holder players do not have any guaranteed victories according to Table 2.2. The only way for a holder player designated for f_i^r to be matched against f_i^r in round q with probability 1 is if the holder player won a subtournament of size 2^{q-1} against a subset of all of the holder players $h_{f_i^r}^t$. However, since the total number of holder players designated for filler player f_i^r is only $2^r - 1$ at the most, neither q nor t can exceed r. Therefore, filler player f_i^r can only advance as far as round r in the tournament with probability 1. □

Claim 2.2.6. *If v^* wins with probability 1, then every filler player f_i^r must survive until at least round r with probability 1.*

Proof. The only player that beats a holder player $h_{f_i^r}^t$ with probability 1 is f_i^r. So in order for v^* to win the tournament with probability 1, all such holder players must be eliminated in a subtournament that f_i^r wins with probability 1. However, there are $2^r - 1$ holder players $h_{f_i^r}^t$, which means that f_i^r must win a subtournament of size at least 2^r and therefore survive with probability 1 at least until round r. □

Combining Claims 2.2.5 and 2.2.6, we see that each filler player f_i^r survives exactly until round r with probability 1 and conclude that its holder players must have been bribed to lose to f_i^r, as in cells $(f_i^r, h_{f_j^{r'}}^{t'})$ and $(h_{f_i^r}^t, f_j^{r'})$ of Table 2.2. Next, we examine the edge players and their associated holder players.

Claim 2.2.7. *If v^* wins with probability 1, then all edge players must survive at least until round $\lceil \log(n-k) \rceil$ with probability 1.*

Proof. Using logic similar to the proof of Claim 2.2.6, e_i must win with probability 1 a subtournament that contains all its $2^{\lceil \log(n-k) \rceil - 1}$ holder players, as it is the only player that can eliminate them all with probability 1. Thus, e_i must survive at least until round $\lceil \log(n-k) \rceil$ with probability 1. □

Claim 2.2.7 allows us to conclude that the edge holder players must have been bribed to lose to the edge players, as in cells $(e_i, h^{r'}_{e_j})$ and $(h^r_{e_i}, e_j)$ of Table 2.2. Now, we examine v^*, the vertex players, and the filler players.

Claim 2.2.8. *If v^* wins with probability 1, then after round $\lceil \log(n-k) \rceil$, at most $k+1$ vertex players have survived with probability 1.*

Proof. According to Claim 2.2.7, if v^* wins with probability 1, then all edge players must advance at least until round $\lceil \log(n-k) \rceil$. So if a vertex player v_i survives until then with probability 1, then in every previous round r, v_i could not have played against any of the edge players. v_i could not have played against v^* either because v^* would lose with nonzero probability. v_i also could not have played against filler players f^q_i where $q \geq r$, since these players must survive round r with probability 1. v_i could not have played against filler players f^q_i where $q < r$ in round r either since v_i would beat f^q_i with probability 1, eliminating f^q_i too late according to Claims 2.2.5 and 2.2.6. Therefore, for each round r in the first $\lceil \log(n-k) \rceil$ rounds of the tournament, v_i must be matched with either another vertex player or a filler player f^r_i intended for that round specifically.

In the first round, we begin with n vertex players. In each round r, at most k vertex players can be matched against the k filler players f^r_i, so the rest must be matched against each other. Let n' be the number of players that advanced to round $r-1$ from the previous round. At least half of the vertex players matched against each other will be eliminated in each round, so the number of vertex players advancing to round r from the previous round is at most $k + \frac{n'}{2}$. Therefore, at the end of round $\lceil \log(n-k) \rceil$, we are left with at most $k + \frac{n-k}{2^{\lceil \log(n-k) \rceil}} = k+1$ vertex players. $\square$

Now, we show that at most k of the $k+1$ remaining vertex players form a vertex cover. From Claim 2.2.8, we know that during the first $\lceil \log(n-k) \rceil$ rounds of the tournament, these $k+1$ players were matched against either another vertex player or a filler player. The special vertex v_0 must be part of this set since it beats both vertex players and filler players with probability 1. In order for v^* to win the tournament with probability 1, all of the edge players must be eliminated with probability 1. Excluding

v_0, the remaining k vertex players are the only set of players in the tournament that (1) have survived through round $\lceil \log(n-k) \rceil$ and (2) will win against edge players with probability 1 if the vertices they represent are incident to the edges that the edge players represent. It follows that if v^* wins with probability 1, each edge must be incident to at least one of the k vertices remaining. Therefore, these k vertex players must form a vertex cover of G. $\square$

Therefore, BTFP is NP-hard when P is monotonic and P' is ϵ-monotonic. $\square$

2.3 Bribing a Few Top Players Is Sufficient

Here we consider the deterministic version of the tournament bribery problem: Given a favorite player i^* and a binary probability matrix T ($T_{i,j} \in \{0,1\}$ and $T_{i,j} = 1 - T_{j,i}$, $\forall i,j \in [n]$), we want to bribe a small number of players to each lose a single well chosen match at cost 1 each, and to find a seeding for which i^* wins. (We use T instead of P, since we will be interchangeably referring to T as a tournament graph and a matrix.) Here we show that for a natural generative model of binary tournaments, there exist $O(\log n)$ players (the "top" ones) so that for all i^* and almost all T, if one bribes these players, then i^* can always be made the winner.

The generative model we work with is the Condorcet Random model (CR Model). Given a probability $p \leq 1/2$ (possibly depending on n), an $n \times n$ matrix T is generated for the players $\{1, \ldots, n\}$ so that for all $i < j$, independently, $T_{i,j} = 1$ with probability $1-p$ and $T_{i,j} = 0$ with probability p [5] [18] [23] [24]. Intuitively, this means that a stronger player almost always beats a weaker player except with small probability p. Figure 2-1 visualizes this model as a directed graph.

Our main theorem is

Theorem 2.3.1. *Let n be a power of 2, $c > 1$, let $p \leq 1/2$ be arbitrary, and let $R = 2(1+\sqrt{2})^2 \approx 11.657$. Then, for at least a $1 - 1/n^{c-1}$ fraction of all possible $n \times n$ matrices T generated by the CR Model with probability p, and all players $i^* \in \{1, \ldots, n\}$, if one bribes players $1, \ldots, Rc\log n$ to lose to i^*, there is an efficiently*

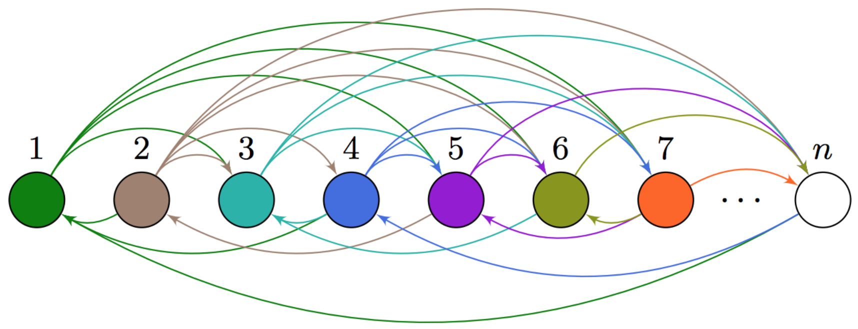

Figure 2-1: An example of a directed graph generated by the Condorcet Model for n players. The edges on the bottom result with small probability p, and the edges on the top result with probability $1-p$.

constructible seeding for which i^* wins.

Proof. We want to point out that the constant R in this theorem is not optimized; in the next subsection, we show how to bring it down to less than 6 for large enough n.

We begin with a theorem from [24].

Theorem 2.3.2. *Consider a tournament graph T on n players V with $\mathcal{K} \in V$. Let $A = N_{out}(\mathcal{K})$ and $B = V \setminus (A \cup \{\mathcal{K}\})$. Suppose that for every $j \in B$, $in_A(j) \geq \log n$. Then there exists an efficiently computable winning seeding for $\mathcal{K}$.*

Our proof will show that no matter what p is, for every player i^* that we could pick, if we bribe the set of players ranked $1, \dots, Rc \log n$ to lose to i^*, then i^* fulfills the conditions of the theorem above with high probability. We will prove the theorem for the weakest player $i^* = n$. The theorem easily follows for all players.

Additionally, we know from past work that if $p \geq c\sqrt{\log n/n}$ for large enough constant c, the theorem holds even without bribery [24], and thus it suffices to prove that the theorem holds when $p < c\sqrt{\log n/n}$.

We will also use the following well known fact which follows from Chernoff bounds:

Fact 2.3.3. *Suppose $X_1, \dots, X_m$ are Bernoulli random variables which are 1 with probability q and 0 otherwise. Then for all $C > 0$,*

$$Pr[\sum_{i=1}^{m} X_i \geq qm - 2\sqrt{C}\sqrt{qm}] \geq 1 - e^{-C}.$$

In our application of this fact, we will be able to set $C = c \ln n$ for constant $c > 1$, so that $e^{-C} = 1/n^c$

Having set $i^* = n$, suppose now that we bribe players $1, \ldots, Rc \log n$. Let T be the resulting tournament graph after generation via the CR model and after bribery. Let A be the set of players i^* can beat according to T, and let $B = V \setminus A \setminus \{i^*\}$. We will show that for any player $j \in B$, if $R \geq 2(1+\sqrt{2})^2$, then $in_A(j) \geq \log n$ with high probability. It follows that if Theorem 2.3.2 is true after bribing players to lose to i^*, then Theorem 2.3.1 must be true.

Claim 2.3.4. *Let $j \in B$. With probability at least $1 - 1/n^c$, $in_A(j) \geq \log n$.*

Proof. Due to the bribery, i^* now beats the top $Rc \log n$ players, which makes them part of A. We will consider how many of these players beat j. This bounds $in_A(j)$ from below. By Fact 2.3.3, with probability at least $1 - 1/n^c$, the number of players among the top $Rc \log n$ that beat j is at least

$$\begin{aligned} Q &:= (1-p)(Rc \log n) - 2\sqrt{c \ln n}\sqrt{(1-p)Rc \log n} \\ &\geq c \log n \left(R(1-p) - 2\sqrt{R(1-p)} \right) \\ &\geq \log n \left(R(1-p) - 2\sqrt{R(1-p)} \right). \end{aligned}$$

since $c > 1$.

We would like to pick R so that $R(1-p) - 2\sqrt{R(1-p)} \geq 1$, allowing us to bound Q so that $Q \geq \log n$, meaning at least $\log n$ of the top $Rc \log n$ players beat j. Now we solve for R by first setting $x = \sqrt{R(1-p)}$. This simplifies our equation, giving us $x^2 - 2x - 1 \geq 0$. The solutions to the quadratic equation are $x = 1 \pm \sqrt{2}$; putting x in terms of R again, and disregarding the negative solution for x, we now have

$$\sqrt{R(1-p)} \geq 1 + \sqrt{2}.$$

Solving this new equation, we get a closed formula for R:

$$R \geq (1+\sqrt{2})^2/(1-p).$$

Substituting $p = 1/2$, we see that $R \geq 11.657$ approximately; for $p < 1/2$, the lower bound for R is even tighter.

As a check, if we set $R = 2(1+\sqrt{2})^2$, as $p \leq 1/2$, $1/(1-p) \leq 2$, we get that

$$R \geq (1+\sqrt{2})^2/(1-p)$$

$$\implies R(1-p) - 2\sqrt{R(1-p)} \geq 1,$$

and thus $Q \geq \log n$. This proves the claim. □

We can conclude the proof of our theorem via a union bound: the probability that there is some $j \in B$ for which $in_A(j) < \log n$ is at most $|B|/n^c \leq 1/n^{c-1}$. Thus with probability at least $1 - 1/n^c$, i^* satisfies the conditions of the theorem, and one can always efficiently find a seeding for which i^* wins the tournament. □

2.3.1 Optimizing the Number of Top Players

We now prove our claim that R (as given in Theorem 2.3.1) can be reduced to less than 6 for sufficiently large n, meaning we can reduce the number of top players we expect to bribe by approximately half.

Theorem 2.3.5. *Let n be a power of 2, $c > 1$, let $p \leq 1/2$ be arbitrary, and let $R \approx 5.92$. Then, for at least a $1 - 1/n^{c-1}$ fraction of all $n \times n$ T generated by the CR model with probability p, and all players $i^* \in \{1, \ldots, n\}$, if one bribes players $1, \ldots, Rc\log n$ to lose to i^*, there is an efficiently constructible seeding for which i^* wins.*

Proof. We begin with a theorem from Kim et al. [18] which is a more stringent variation on Theorem 2.3.2.

Theorem 2.3.6. *Consider a tournament graph T on n players V where $\mathcal{K} \in V$ is a king. Let $A = N_{out}(\mathcal{K})$ and $B = V \setminus (A \cup \{\mathcal{K}\}) = N_{in}(\mathcal{K})$. Suppose that B is a disjoint union of three (possibly empty) sets H, I, J such that*

1. *$|H| < |A|$,*

2. $in_A(i) \geq \log n$ for all $i \in I$, and

3. $out(j) \leq |A|$ for all $j \in J$.

Then there is a winning seeding for $\mathcal{K}$ that can be computed in polynomial time.

We will show that for any player $i^* \in [n]$ that we could pick, if we bribe players $1, \dots, 6c \log n$ to lose to i^*, then i^* fulfills the conditions of the theorem above with probability at least $1 - 1/n^{c-1}$. More specifically, i^* is a king, and we can define H, A, I and J so that by the theorem above, i^* can be made to win. It follows that if Theorem 2.3.6 is true after bribing players to lose to i^*, then Theorem 2.3.5 must be true.

We will again apply Fact 2.3.3 to set $C = c \ln n$ for constant $c > 1$, so that $e^{-C} = 1/n^c$. We will also prove Theorem 2.3.5 for the case where i^* is the weakest player and $p < c\sqrt{\log n/n}$, so the theorem easily follows for all players and for all $p < 1/2$.

Suppose that we pick $i^* = n$ and we bribe players $1, \dots, 6c \log n$. Let T be the resulting tournament graph after generation via the CR model and after bribery. We will first show that the first condition of Theorem 2.3.6 is true following the bribery.

Claim 2.3.7. *With probability at least $1 - 1/n^d$ for $d \geq 1$, i^* is a king in T.*

Proof. As a result of the bribery, i^* beats the top $6c \log n$ players. Consider any player $j \in \{6c \log n + 1, \dots, n-1\}$. The probability that none of the top $6c \log n$ players beat j (meaning each of these players' outgoing edges to j was flipped) is $p^{6c \log n}$. Applying a union bound, we see that the probability that there exists some $j \in \{6c \log n + 1, \dots, n-1\}$ such that none of the top $6c \log n$ players beat j is at most $np^{6c \log n}$. We assumed that $p < c\sqrt{\log n/n}$, and so

$$np^{6c \log n} < n(c \log n/n)^{3c \log n} \leq n/n^{2 \log n} < 1/n^d,$$

for any $d \geq 1$. $\square$

We now divide all players other than i^* into the sets H, A, I, and J. Since A consists of all players that i^* can beat according to T, A clearly contains the top $6c\log n$ players. Let H contain all players in the ordering $\{1 + 6c\log n, \ldots, 12c\log n\}$ that are not already in A. It follows that all of the "second-best" $6c\log n$ players after the first $6c\log n$ players are either in A or in H, satisfying the first condition of Theorem 2.3.6. Finally, set $I = V \setminus (A \cup H \cup \{v^*\})$ and $J = \emptyset$. Our partition is depicted in Figure 2-2.

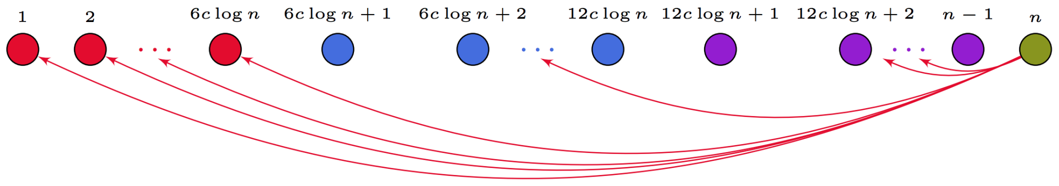

Figure 2-2: An n-player directed graph generated by our random model, post-bribery. $i^* = n$, denoted by the green player on the far right. The players in set A are red, the players in set H are blue, and the players in set I are purple. The red edges denote the winning matches for v^*, regardless of whether they flipped due to a bribe or the randomness of Condorcet.

Now consider a player $i \in I$. Clearly $i > 12c\log n$. We will show that $in_A(i) \geq \log n$ with high probability, satisfying the second condition of Theorem 2.3.6. Since $J = \emptyset$, proving that the first two conditions of the theorem are satisfied is sufficient.

Claim 2.3.8. *Let $i \in I$. With probability at least $1 - 1/n^c$, $in_A(i) \geq \log n$.*

Proof. We will only consider how many players among the first $2ac\log n$ are both beaten by i^* and beat i. We will try to minimize a and show that for large enough n, $a = 6$ suffices.

The top $ac\log n$ players are all beaten by i^*. From our proof of Claim 2.3.4, with probability at least $1 - 1/n^c$, the number of top $ac\log n$ players that beat i is at least

$$Q_1 := (1-p)ac\log n - 2\sqrt{c\ln n}\sqrt{(1-p)ac\log n} \geq .$$

$$c\log n\sqrt{1-p}(a\sqrt{1-p} - 2\sqrt{a}).$$

Now we consider the next $ac\log n$ players. The probability that one of these players is beaten by i^* and beats i is $p(1-p)$. Applying the same logic, we see that with probability at least $1 - 1/n^c$, the number of players from this group that both are beaten by i^* and beat i is at least

$$Q_2 := p(1-p)(ac\log n - 2\sqrt{c\ln n}\sqrt{p(1-p)ac\log n} \geq .$$
$$c\log n\sqrt{(1-p)}(ap\sqrt{(1-p)} - 2\sqrt{ap}).$$

It follows that with high probability, the total number of players from among the top $2ac\log n$ players that i^* beats and that beat i is

$$Q_1 + Q_2 \geq c\log n\sqrt{a(1-p)}\left((1+p)\sqrt{a(1-p)} - 2(1+\sqrt{p})\right).$$

We want $Q_1 + Q_2 \geq c\log n$, so we set

$$\sqrt{a(1-p)}((1+p)\sqrt{a(1-p)} - 2(1+\sqrt{p})) \geq 1.$$

Setting $x = \sqrt{a(1-p)}$, we can now solve the following simpler result: $(1+p)x^2 - 2(1+\sqrt{p})x - 1 \geq 0$. The solutions to the quadratic equation are

$$x = \frac{(1+\sqrt{p}) \pm \sqrt{2+2p+2\sqrt{p}}}{1+p}.$$

Putting x back in terms of a and p again, and disregarding the negative solution for x, we now have

$$x = \sqrt{a(1-p)} \geq \frac{(1+\sqrt{p}) + \sqrt{2+2p+2\sqrt{p}}}{1+p},$$

and hence
$$a \geq \left(\frac{(1+\sqrt{p}) + \sqrt{2+2p+2\sqrt{p}}}{(1+p)\sqrt{1-p}}\right)^2.$$

The right hand side above is a growing function of p in the interval between 0 and 1/2. Since we have assumed that $p < c\sqrt{\log n/n}$, it follows that $p < 7c\sqrt{\log n/n}$. For large enough n, this is smaller than 0.0001, yielding approximately $a \geq 5.92$; however, it suffices to set $a = 6$. This proves the claim. $\square$

We can conclude the proof of our theorem via a union bound as before. The probability that there is some $i \in I$ for which $in_A(i) < \log n$ is at most $|I|/n^c \leq 1/n^{c-1}$. Thus with probability at least $1 - 1/n^c$, i^* satisfies the conditions of the theorem, and one can always efficiently find a seeding for which i^* wins the tournament by bribing the top $6c \log n$ players. $\square$

Chapter 3

NP-Completeness of Deterministic TFP

3.1 Preliminaries

In this section we examine the relationship of deterministic TFP to other NP-complete problems, and the features of the problem that bode well or ill for reductions. We first restate our definition for TFP, which will be used as an abbreviation for the problem of fixing a *deterministic* tournament for both this chapter and the following chapter.

Definition 3.1.1. *A* **deterministic knockout tournament** *plays out as a balanced binary tree given an initial seeding of the set of players $N = \{1, 2, \ldots, n\}$ where $n = 2^m$ for some integer m. In the first seeding, the n sibling leaf nodes play a match against each other at level m of the tree. The winner of each match ascends the tree to the next round at level $m-1$. The winner of the tournament is the player who reaches the tree's root node at level 0. Match results are governed by a pairwise comparison matrix P where $P_{ij} \in \{0, 1\}$ for all players $i, j \in N$. If $P_{ij} = 1$, then row player i beats column player j in a match. Additionally, $P_{ij} + P_{ji} = 1$, as in the probabilistic case.*

In this chapter, the computational question of interest is whether or not there exists an initial seeding for which a particular player can win a tournament where the winner of each possible match is known with certainty ahead of time according to the dictations of P. We state the problem more formally:

Definition 3.1.2. The Tournament Fixing Problem (Deterministic) *(TFP): Given a set of players $N = [n]$, a deterministic pairwise comparison matrix P, and a favorite player $i^* \in N$, does there exist an initial seeding σ for N for which i^* wins the tournament?*

3.2 Reducing CLIQUE or VC to TFP Is Intractable

We show that two well-known NP-complete graph problems are surprisingly incompatible with the problem of fixing a deterministic tournament, which can be rephrased as a graph problem as we saw in section 2.3. Since one of the inputs of an instance of TFP is a directed graph, it is tempting to reduce other NP-complete graph problems to TFP in order to reinforce its complexity. What makes the construction of a polynomial-time reduction from either one of these problems to TFP very difficult (and untenable so far) is that one problem relies on the relationships within a certain group of nodes while another relies on a specific permutation of nodes. The translation of the former to the latter causes an exponential blowup in the number of nodes, which violates a cornerstone of proving NP-completeness. We demonstrate this for each problem investigated to show the untenability of a poly-time reduction. Recall the definition of *Vertex Cover* from section 2.2.

Definition 3.2.1. VERTEX COVER*(VC): Given a graph $G = \{V, E\}$ and an integer k, does there exist a subset $C \in V$ such that $|C| \leq k$ and C covers E?*

Building a reduction is a conundrum of translating the affirmative characteristics within an instance of one problem into the different but still affirmative characteristics within an instance of another problem. In this case, VC requires that the vertices in the cover collectively are connected to the entire edge set E through one or both endpoints. Translating this relationship into a deterministic tournament would require that players represent both the vertices and edges in the graph. For example, a vertex player in the cover could eliminate an edge player that it covers in a tournament match. This would require that vertex players not in the cover be eliminated before the edge players could be eliminated. It would also require that the vertex players in the cover all make it to a particular round, meaning the tournament must be comprised of at least k subtournaments. However, unlike the ϵ-monotonic comparison matrix used to represent VC that was described in section 2.2, this matrix completely eliminates the uncertainty that Claims 2.2.5 through 2.2.8 relied upon in order to prove one direction of the reduction.

This issue poses a serious hindrance when we attempt to prove that a vertex cover of size k can be found if a favorite player such as i^* wins with probability 1. So if a vertex player survives until a particular round with probability 1, for example, it is possible that it advanced that far through the tournament by eliminating edge players too early or by eliminating anoher vertex player that it should not have. It is also possible also that one vertex player in the cover could be eliminated by an edge player that it did not cover so that the edge player could advance to a particular round. In other words, there are ways for i^* to win that are not contingent upon the existence of a vertex cover of size k in the instance of VC.

Although TFP and VC are both graph problems, they are ultimately incompatible for a poly-time NP-hardness reduction because it is impossible to guarantee (1) that the edge players made it to a certain point without eliminating vertex players in the cover and (2) that the vertex players in the cover made it to a certain point without eliminating each other. Furthermore, the scope of the tournament requires several additional holder and filler players for the edge players, the vertex players, and the favorite player. Forcing deterministic relationships between these players so that an instance of TFP can be constructed makes it impossible to guarantee that a vertex cover can be found if and only if the favorite player wins the tournament, at least not without an exponential size blowup.

A reduction was attempted for the known NP-complete problem CLIQUE as well. We define the problem formally.

Definition 3.2.2. **CLIQUE**: *Suppose that a set S of a given graph $G = \{V, E\}$ form a clique if each vertex in S is connected to every other vertex with an edge. Given G and an integer k, does there exist a clique in G of size k?*

It is clear that CLIQUE requires that each vertex in the clique be connected to every other vertex in the clique by way of an edge. The problem also requires that no other vertex outside of the clique is connected to all of the clique vertices through an edge; otherwise, the clique would have size $k + 1$. It seems reasonable that the resulting tournament graph for an instance of TFP would have one node to represent each vertex in the CLIQUE input graph, for a total of $|V|$ nodes in the tournament graph. However, we must translate the affirmative relationships between the nodes in the CLIQUE graph that yield an accepted instance of the problem into a valid tournament graph that a certain favorite player wins. This process would require many more nodes. For example, one might intuitively require two additional nodes for each edge (u, v) so that u could eliminate one copy of the edge in a match and v could eliminate the other in a different match. This adds $2|E|$ nodes to the tournament graph. However, this would require the elimination of (1) all other edges and (2) all other vertices not in the clique so that only the ones that comprise the clique would remain. A deterministic tournament makes it impossible to distinguish between an edge being eliminated by a vertex not in the clique and an edge being eliminated by a vertex in the clique, so this would not work in a polynomial-time reduction.

One might also design a tournament graph containing a single node for each vertex in the CLIQUE graph and multiple copies of the edges. In order for i^* to win, the graph might require each of the clique vertices to eliminate a single copy of the edge connecting it to the other clique vertices, a process which would take at least k rounds. The directed graph may also be set up so that a favorite player wins if and only if all k of the clique vertices make it to a particular round, and all the others are eliminated. Both of these cases require that these vertices each win a subtournament of at least k rounds, requiring a total of approximately $k \cdot 2^{O(k)}$ nodes to be added to the directed graph. The remaining $|V| - k$ vertices and their edges must also be eliminated, which further adds to the now too-high complexity

Ultimately, this attempt to represent a relationship between a group of nodes in the CLIQUE problem with a directed graph defined by binary edge relationships that manifests as a tournament is very difficult to represent with just a polynomial number of nodes as a function of $|V|$, $|E|$, and k. This is because a group relationship like this tends to manifest itself in a tournament as a series of rounds, which causes the reduction blowup to increase dramatically since the number of rounds is a logarithmic function of the total number of nodes in the tournament graph. Furthermore, the tournament graph is constructed with an order in mind for how the matches should play out and when certain players should be eliminated over the course of the game. A problem like CLIQUE ignores order altogether, so reducing CLIQUE to a problem like TFP where the correct permutation of the matches is critical for an affirmative instance adds an unforeseen amount of complexity. Despite both TFP and CLIQUE being graph problems, they ultimately are incompatible for a poly-time NP-hardness reduction.

3.3 NP-Completeness of TFP

We now summarize the proof demonstrating that TFP is NP-complete [3].

Lemma 3.3.1. *TFP is NP-complete.*

Proof. It is easy to see that TFP is in NP. To show its NP-hardness, one can reduce to it from 3SAT2, which is a variant of the 3SAT problem in which every literal appears no more than twice. Given a 3SAT2 instance $F = (X, C)$ where $X = \{x_1, \ldots, x_{|X|}\}$ is the set of variables and C the set of clauses, we summarize how to construct an instance of TFP where a favorite player who can win the tournament via some initial seeding if and only if F is satisfiable.

The instance of TFP has a set of players N of size n, where n is the smallest power of 2 that is greater than or equal to $32 \cdot |X|$. The tournament instance will have a total of $\log n \geq 5$ rounds. N can be grouped into a series of disjoint sets and gadgets. We briefly enumerate each of them:

- Player set $M = \{m_1, \ldots, m_k\}$ where $k = \frac{n}{16}$ and m_1 is the favorite player who is ultimately intended to win the tournament. In order for this to happen, all players from M must make it to the fifth round.

- Choice gadget set $G^X = \{G_1, G_2, \ldots, G_{|X|}\}$

- Clause/garbage gadget set $G^{CG} = \{G_{|X|+1}, G_{|X|+2}, \ldots, G_{2|X|}\}$

- Filler gadget set $G^F = \{G_{2|X|+1}, G_{2|X|+2}, \ldots, G_k\}$

- Superset $S = \{S_1, \ldots, S_{|X|}\}$ where each set S_j corresponds to the variable x_j of the 3SAT2 instance, $S_j = \{S_{x_j}, S_{\bar{x}_j}, s_j^*\}$, and $|S_{x_j}| = |S_{\bar{x}_j}| = 3$.

Figure 3-1 depicts the directed graph relating the elements of M. It represents a recursive spawning process so that m_1 wins the subtournament when the nodes in level 3 are seeded from left to right, in ascending order $m_1, m_2, \ldots, m_k$. The full proof demonstrates that this is the only way for m_1 to win the tournament, which means that all k players in M must make it to the fifth round so that they can be seeded in this fashion [3].

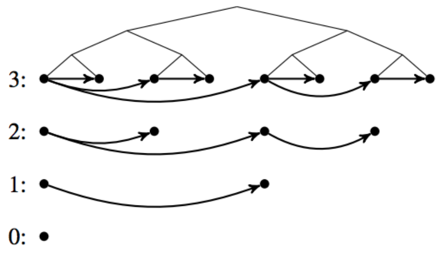

Figure 3-1: The recursive spawning process for the case $k = 8$, where the players in level 3 are arranged in ascending order $m_1, \ldots, m_8$ from left to right. If the players are seeded according to this permutation, then m_1 will win the tournament.

Figure 3-2 depicts the directed graph relating the players between different gadgets and sets. Unless indicated otherwise, all players from sets with a higher index will beat elements from sets with a lower index. In particular, S can be divided into the set of special players $\{s_1^*, s_2^*, \ldots, s_{|X|}^*\}$ and the set of players S^p that interact with players in the choice and garbage gadgets. For $j > j'$, elements from S_{x_j} and

$S_{\bar{x}_j}$ beat elements from $S_{x_{j'}}$ and $S_{\bar{x}_{j'}}$, and each element s^*_j is beaten by all players $\{s^*_{j'} \mid j+1 \leq j' \leq |X|\}$.

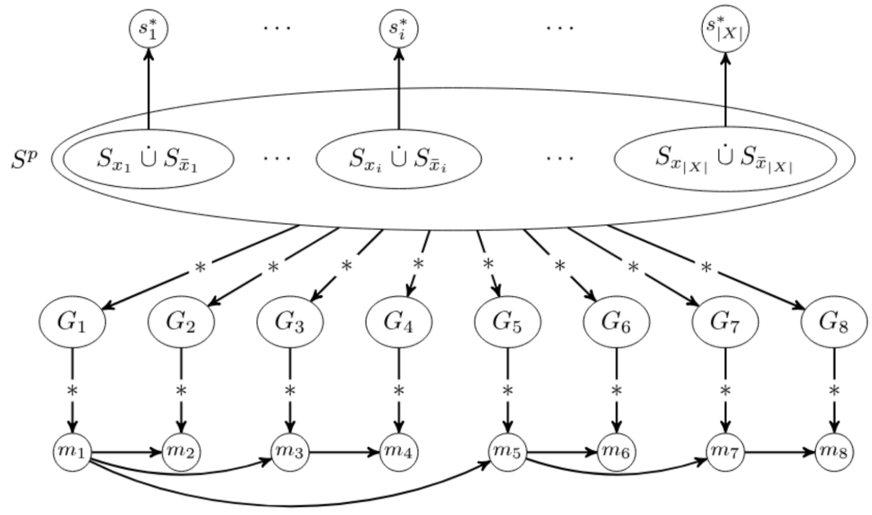
s^*_1
s^*_i
$s^*_{|X|}$
S^p
$S_{x_1} \dot\cup S_{\bar{x}_1}$
$S_{x_i} \dot\cup S_{\bar{x}_i}$
$S_{x_{|X|}} \dot\cup S_{\bar{x}_{|X|}}$
G_1 G_2 G_3 G_4 G_5 G_6 G_7 G_8
m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8

Figure 3-2: The global structure of the tournament, comprising all n players and divided by the aforementioned sets and gadgets. All vertical arrows not shown point down, and all horizontal arrows not shown point from right to left. The arrows accompanied by an asterisk indicate an exception: Every player m_i beats exactly four players in the gadget G_i, and some players in G_i beat a subset of players in S^p [3].

Each choice gadget consists of player m_j, all of S_j, and all ten elements of G_j for $1 \leq j \leq |X|$. The pairwise comparison graph for these players is shown in Figure 3-3, and it is structured in such a way that it is possible for m_j to win a subtournament composed of all elements in the choice gadget except for two elements of either S_{x_j} or $S_{\bar{x}_j}$, as depicted in Figure 3-4. The full proof demonstrates that this is the only way in which m_j can reach the fifth round in a tournament ultimately won by m_1.

Each clause/garbage gadget consists of player m_j and the thirteen elements of G_j (two of which are denoted by c/g) for $|X|+1 \leq j \leq 2|X|$. The pairwise comparison graph for these players is shown in Figure 3-5. For each clause of the 3SAT2 instance $c_i \in C$, one of the players denoted c/g is associated with clause c_i while all the other players c/g are garbage players. All players shown in the figure are beaten by all players in S except for (1) all garbage players beating all players from S^p, and (2) players associated with clause c_i beat all players from either the set S_{x_j} or the set $S_{\bar{x}_j}$ if x_j or $\bar{x}_j$ occcurs in clause c_i, respectively. The clause gadget is designed so that m_j can win a subtournament containing all players in the gadget with the addition

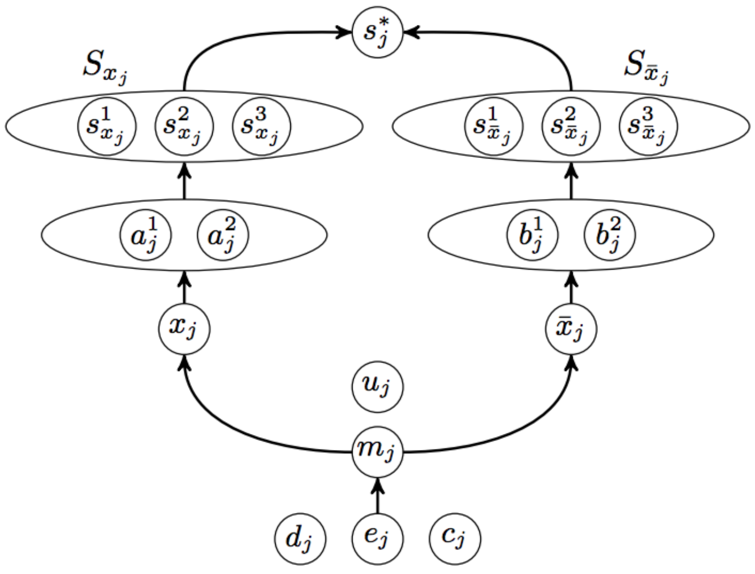

Figure 3-3: Directed graph relating the players in choice gadget G_j. All vertical arrows not shown in the graph point down, and all horizontal arrows not shown point from right to left. Players grouped together in ovals have the same relationship with players outside the ovals [3].

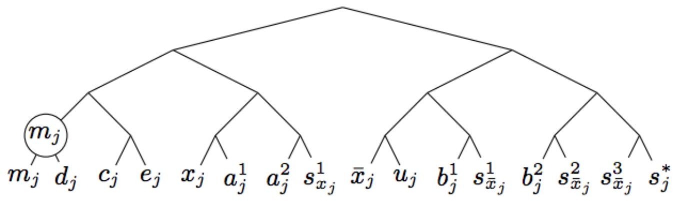
m_j
m_j d_j c_j e_j x_j a_j^1 a_j^2 $s_{x_j}^1$ $\bar{x}_j$ u_j b_j^1 $s_{\bar{x}_j}^1$ b_j^2 $s_{\bar{x}_j}^2$ $s_{\bar{x}_j}^3$ s_j^*

Figure 3-4: The subtournament for choice gadget j in the case where $x_j = True$ [3].

of a compatible player in S^p for both c/g players. The full proof demonstrates that this is the only way for m_j to reach the fifth round in order for m_1 to win the entire tournament.

Each filler gadget consists of m_j and the fifteen elements of G_j for $2 \cdot |X| + 1 \leq j \leq k$. The directed graph for these players is shown in Figure 3-6. Note that for all $j \in [k]$, player m_j beats exactly four players not in the set M. This manifests in the following three cases:

1. If $1 \leq j \leq |X|$, then the associated gadget is a choice gadget. The 16-player subtournament allowing m_j to reach the fifth round thus consists of m_j, the players of G_j, and the players of S_j excluding two elements of either $S_{x_j} \cup \{s_j^*\}$ or $S_{\bar{x}_j} \cup \{s_j^*\}$.

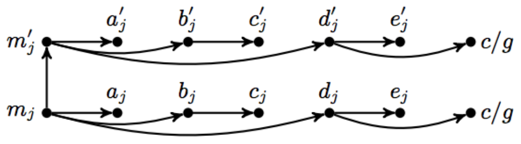

Figure 3-5: Directed graph relating the players within a clause/garbage gadget through the aforementioned recursive spawning process. All vertical arrows not shown in the graph point down, and all horizontal arrows not shown point from right to left [3].

2. If $|X| + 1 \leq j \leq 2 \cdot |X|$, then the associated gadget is a clause/garbage gadget. The 16-player subtournament allowing m_j to reach the fifth round thus consists of m_j, the players of G_j, and one additional player from the set S^p for each clause/garbage player c/g to beat.

3. If $2 \cdot |X| + 1 \leq j \leq k$, then the associated gadget is a filler gadget. The 16-player subtournament allowing m_j to reach the fifth round thus consists of m_j and the players of G_j.

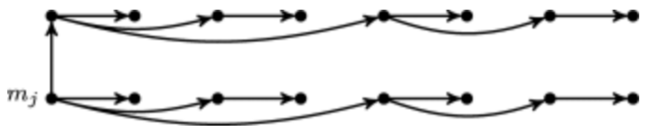

Figure 3-6: Directed graph relating the players in a filler gadget through the aforementioned recursive spawning process. All vertical arrows not shown in the graph point down, and all horizontal arrows not shown point from right to left [3].

The complete proof demonstrates through induction that for each of these subtournaments, m_j makes it to the fifth round if and only if a specific seeding is used. Likewise, once all players m_j make it to the fifth round, the aforementioned seeding depicted in Figure 3-1 must be followed so that m_1 wins. It also demonstrates that a draw in which each player $m_j \in M$ wins a 16-player soubtournament with exactly the players stated exists if and only if the 3SAT2 instance is satisfiable, satisfying the conditions for a correct NP-hardness reduction. $\square$

3.4 Size Blowup for 3SAT2 $\leq_m$ TFP

In this section, we give a lower bound on the blow-up of the size of the reduction described in the previous section. The blow-up comes from the size of the minimal feedback arc set within the new instance of TFP relative to the size of the input $F = (X, C)$.

Theorem 3.4.1. *The reduction from 3SAT2 to TFP cannot be accomplished with a size blowup tighter than $O(|X|^2)$.*

Proof. Within each of the $|X|$ clause/garbage gadgets, there are either one or two garbage players g which collectively beat all players from S^p. This means that there are $O(|X|)$ outgoing edges from each of the $O(|X|)$ garbage players to the players in S^p. Because the remaining players in S^p beat all other players to the left of the c/g players as depicted in Figure 3-5, several cycles are created within this gadget. The size of the minimal feedback arc set therefore is at least quadratic in size. The filler and choice gadgets each have blowups of size $O(|X|)$, so the tightest bound we can place on the reduction's blowup is $O(|X|^2)$. □

If we were able to solve TFP in $2^{o(n)}$ time, then this quadratic blowup means that 3SAT2 would have to be solvable in $2^{o(n)}$ time. However, since 3SAT has no such algorithm according to the exponential time hypothesis, neither does 3SAT2, and an algorithm solving TFP therefore must require at least $2^{O(n)}$ time.

3.5 Reducing 3SAT Directly to TFP

We now show how to expand this reduction in order to create a polynomial-time reduction from the general version of 3SAT to TFP.

Theorem 3.5.1. *3SAT can be reduced directly to TFP in polynomial time.*

Proof. Figures will be updated. Let L be the maximum number of times that any of the literals appears in our instance $F = (X, C)$ of 3SAT. We have to modify each choice gadget G_j so that the number of players and rounds reflects the case where

$L \neq 2$. More specifically, we must structure the subtournament ultimately won by m_j so that it consists of m_j, the players of G_j, and the players of S_j excluding L elements of either S_{x_j} or $S_{\bar{x}_j}$. We do this by recursively breaking down L into two types of 5-round subtournaments: one that excludes two elements of either S_{x_j} or $S_{\bar{x}_j}$, which is already demonstrated in Figure 3-3; and one that excludes exactly one element of either S_{x_j} or $S_{\bar{x}_j}$, as demonstrated in Figure 3-7. The tree that results from the latter subtournament is depicted in Figure 3-8.

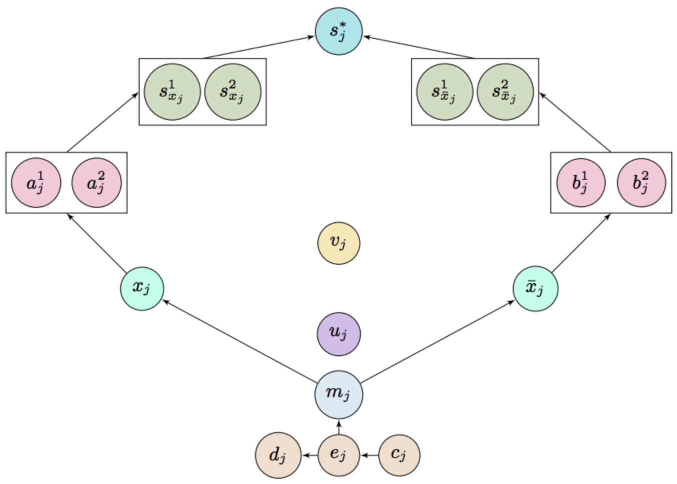

Figure 3-7: Directed graph relating some of the players in choice gadget G_j. All vertical arrows not shown in the graph point down, and all horizontal arrows not shown point from right to left. If $x_j = True$, then one player from the set $\{s^1_{x_j}, s^2_{x_j}\}$ will be unused in the resulting choice gadget subtournament and be used instead in the clause gadget subtournaments. If $x_j = False$, then one player from the set $\{s^1_{\bar{x}_j}, s^2_{\bar{x}_j}\}$ will be unused in the resulting choice gadget subtournament and be used instead in the clause gadget subtournaments.

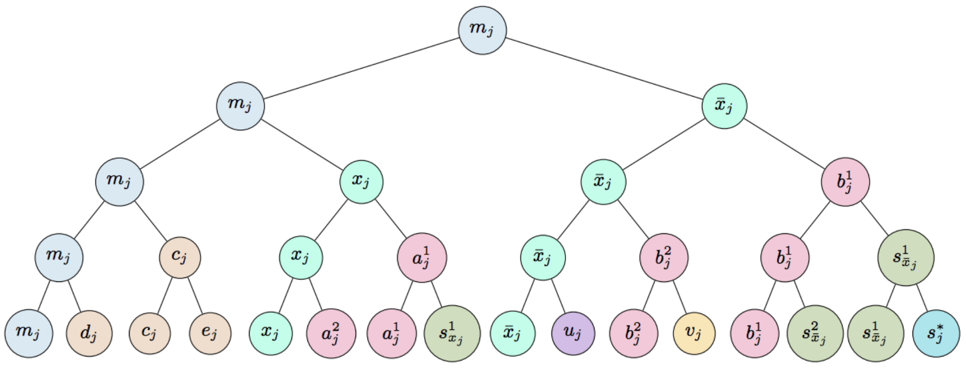

Figure 3-8: The subtournament for choice gadget j in the case where x_j or $\bar{x}_j$ appears at most once in F ($L = 1$) and $x_j = True$.

In the case where $L = 3$, we would combine a 5-round subtournament that ultimately excludes either two elements of S_{x_j} or two elements of $S_{\bar{x}_j}$ with a 5-round subtournament that excludes either a single element of S_{x_j} or a single element of $S_{\bar{x}_j}$, for a total of 3 elements excluded from S_{x_j} or $S_{\bar{x}_j}$. Note that the single element excluded by one subtournament cannot be either of the two elements excluded by the other subtournament. The complete directed graph is given in Figure 3-9. The resulting trees indicating how the tournament plays out round by round are given in Figure 3-10. This makes sense, since the assignment $x_j = 1$ or $x_j = 0$ must be consistent for all four appearances of the literal. Then, m_j would ultimately win a 6-round subtournament.

Similarly, if $L = 7$, we would combine three 5-round subtournaments that each exclude 2 elements of the same set with one 5-round subtournaments that exclude a single element of that same set, for a total of 7 elements excluded from S_{x_j} or 7 elements excluded from $S_{\bar{x}_j}$. Player m_j would ultimately win a 7-round subtournament. The complete directed graph is given in Figure 3-11, and the resulting tournament tree is depicted in Figures 3-12 and 3-13. As a final example, if $L = 13$, we would combine five 5-round subtournaments that each exclude 2 elements of the same set with three 5-round subtournaments that each exclude a single element of that same set, for a total of 13 elements excluded from S_{x_j} or $S_{\bar{x}_j}$. Player m_j would ultimately win an 8-round subtournament.

We now give a brief proof of correctness for this reduction, relying heavily on what we know from the proof given in section 3.3 [3].

Theorem 3.5.2. *There exists a draw such that m_1 wins the tournament if and only if the* 3SAT *instance is satisfiable.*

Proof. We already know that by a simple extension of the 3SAT2 reduction's proof of correctness that m_1 can win the tournament if and only if all the players from M reach round $\lceil \log L \rceil + 4$. We therefore need to show that there exists a draw that enables all players $m_j \in M$ to win their respective $16L$-player subtournaments with exactly the players described if and only if the 3SAT instance is can be satisfied.

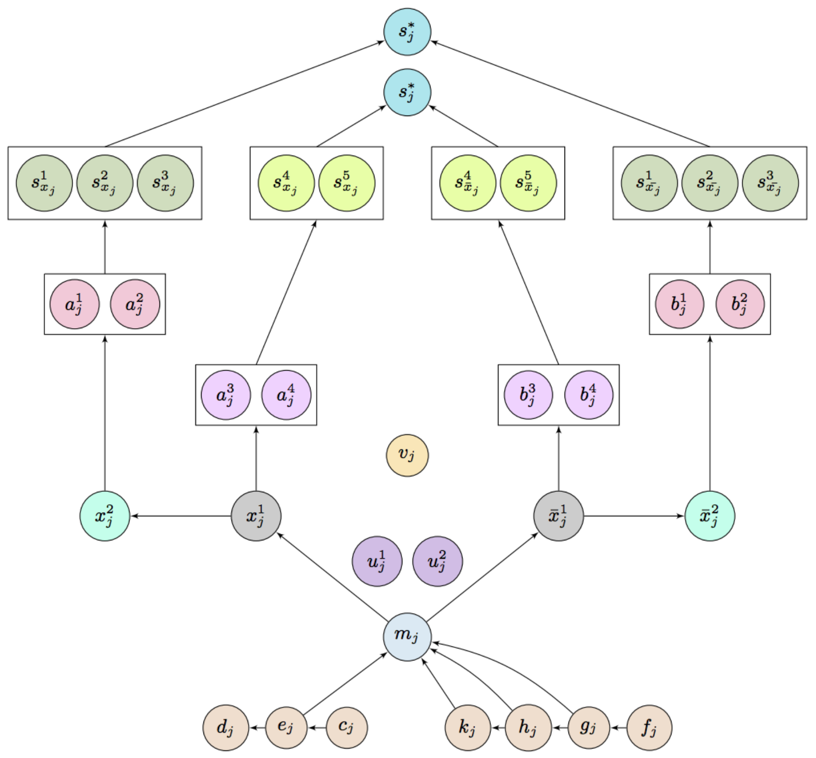

Figure 3-9: Directed graph relating some of the players in choice gadget G_j in the case where $L = 3$. All vertical arrows not shown in the graph point down, and all horizontal arrows not shown point from right to left. If $x_j = True$, then two players from the set $\{s_{x_j}^1, s_{x_j}^2, s_{x_j}^3\}$ and one player from the set $\{s_{x_j}^4, s_{x_j}^5\}$ will be unused in the resulting choice gadget subtournament and be used instead in the clause gadget subtournaments. If $x_j = False$, then two players from the set $\{s_{\bar{x}_j}^1, s_{\bar{x}_j}^2, s_{\bar{x}_j}^3\}$ and one player from the set $\{s_{\bar{x}_j}^4, s_{\bar{x}_j}^5\}$ will be unused in the resulting choice gadget subtournament and be used instead in the clause gadget subtournaments.

Lemma 3.5.3. *($\leftarrow$): If the formula F is satisfied by an assignment ψ, then there exists a draw such that m_1 wins the tournament.*

Proof. Suppose that an assignment ψ satisfies the formula F. For all j where gadget G_j is a choice gadget, put all players from $G_j \cup S_j \cup \{m_j\}$ into the same subtournament for the first $\lceil \log L \rceil + 4$ rounds, with an exception. If $\psi(x_j) = True$, then exclude L players from from S_{x_j}; if $\psi(x_j) = False$, then exclude L players from $S_{\bar{x}_j}$. By a logical extension of one of the lemmas in [3], we see that each of these subtournaments is really a binary tree of 4-round subtournaments in which 1 or 2 players from S_{x_j} ($S_{\bar{x}_j}$) are excluded if $\psi(x_j) = True$ ($\psi(x_j) = False$), and the tournament results from round 4 onward depend on which players make it to round 4. It is clear to see also that m_j will only win the subtournament if the players excluded across the

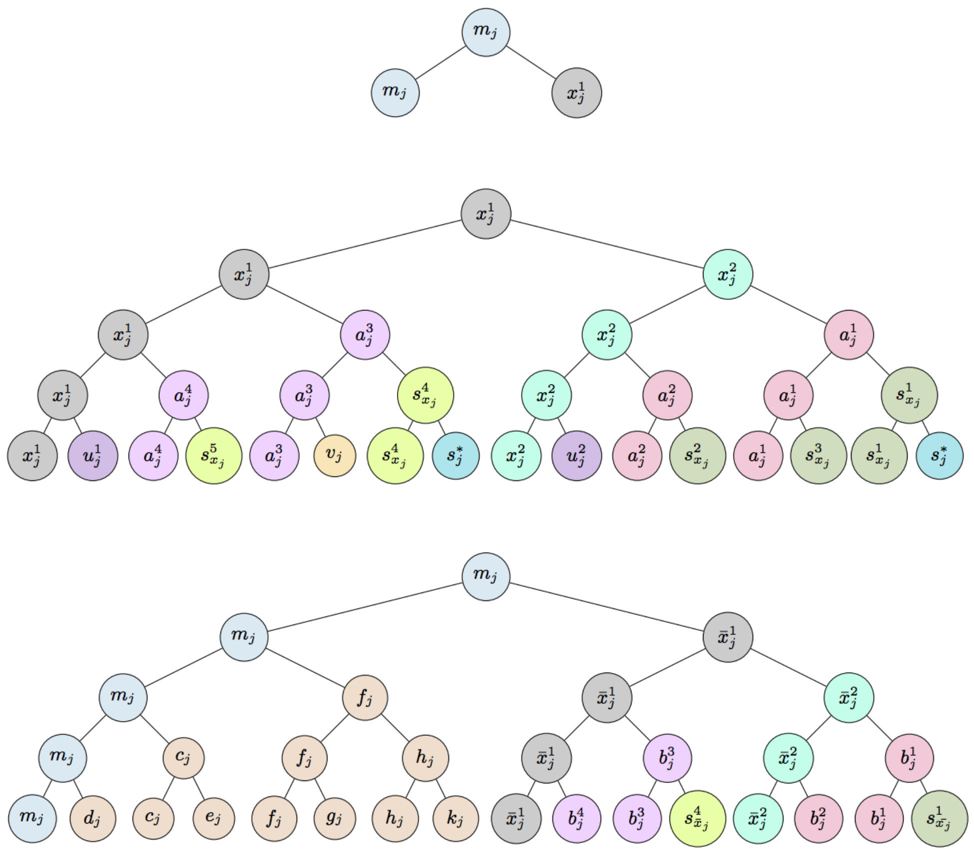

Figure 3-10: The 6-round subtournament for choice gadget j in the case where x_j or $\bar{x}_j$ appears at most three times in F ($L = 3$) and $x_j = False$. On the top is the subtournament consisting of rounds 5 and 6 that allows m_j to ultimately win the 6-round subtournament. In the middle is the 5-round subtournament that x_j^1 wins, and on the bottom is the 5-round subtournament that m_j wins.

subtournaments come from either S_{x_j} or $S_{\bar{x}_j}$. Therefore, we can choose the draw definitively so that m_j wins the subtournament.

For every literal that evaluates to true, L players remain that are not part of the choice gadget subtournament and can be beaten by every player associated with a clause $c_i \in C$ that contains that literal. We know that each clause must evaluate to true, and meaning at least one literal in the clause evaluates to true. For a player c/g associated with a clause $c_i \in C$, select one of the literals in c_i that evaluates to true. If $x_j \in c_i$ and $x_j = True$, assign one of the aforementioned L excluded players from S_{x_j} to player c/g. Likewise, if $\bar{x}_j \in c_i$ and $x_j = False$, assign one of the aforementioned L excluded players from $S_{\bar{x}_j}$ to player c/g. To all garbage players, assign any remaining player from the superset S. Extending the logic of the hardness

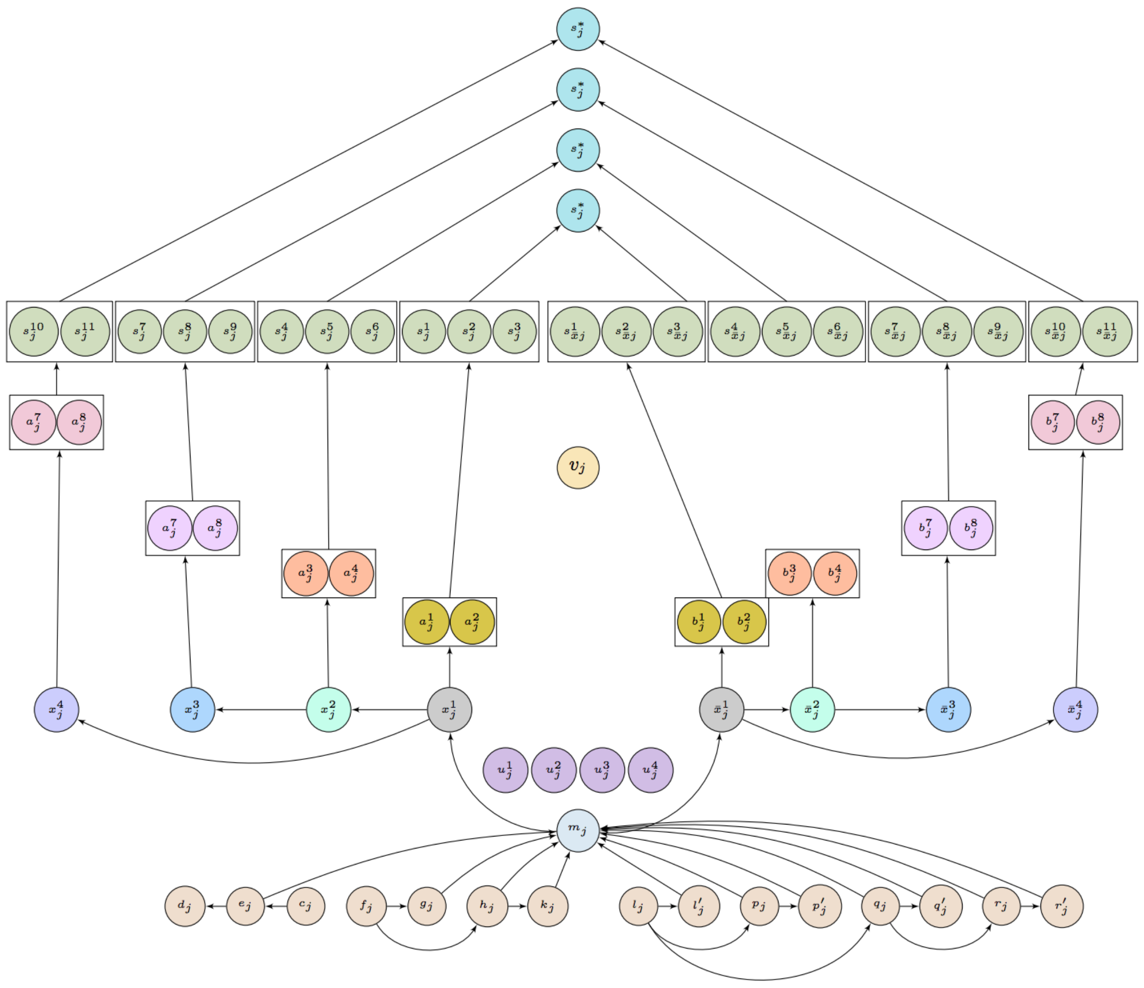

Figure 3-11: Directed graph relating some of the players in choice gadget G_j in the case where $L = 7$. All vertical arrows not shown in the graph point down, and all horizontal arrows not shown point from right to left. If $x_j = True$, then two players from the set $\{s^1_{x_j}, s^2_{x_j}, s^3_{x_j}\}$, two players from $\{s^4_{x_j}, s^5_{x_j}, s^6_{x_j}\}$, two players from $\{s^7_{x_j}, s^8_{x_j}, s^9_{x_j}\}$ and one player from $\{s^{10}_{x_j}, s^{11}_{x_j}\}$ will be unused in the resulting choice gadget subtournament and be used instead in the clause gadget subtournaments. If $x_j = False$, then two players from the set $\{s^1_{\bar{x}_j}, s^2_{\bar{x}_j}, s^3_{\bar{x}_j}\}$, two players from $\{s^4_{\bar{x}_j}, s^5_{\bar{x}_j}, s^6_{\bar{x}_j}\}$, two players from $\{s^7_{\bar{x}_j}, s^8_{\bar{x}_j}, s^9_{\bar{x}_j}\}$ and one player from the set $\{s^{10}_{\bar{x}_j}, s^{11}_{\bar{x}_j}\}$ will be unused in the resulting choice gadget subtournament and be used instead in the clause gadget subtournaments.

proof in [3], we see that after the choice gadget subtournaments were constructed and the remaining players were assigned to the clause/garbage gadget subtournaments, a draw allowing m_j to win the subtournament for every j has been found. □

Lemma 3.5.4. *($\rightarrow$): If there exists a draw such that m_1 wins the tournament, then there exists an assignment ψ that satisfies the formula F.*

Proof. Suppose that we are given a draw such that m_1 wins the tournament. We know

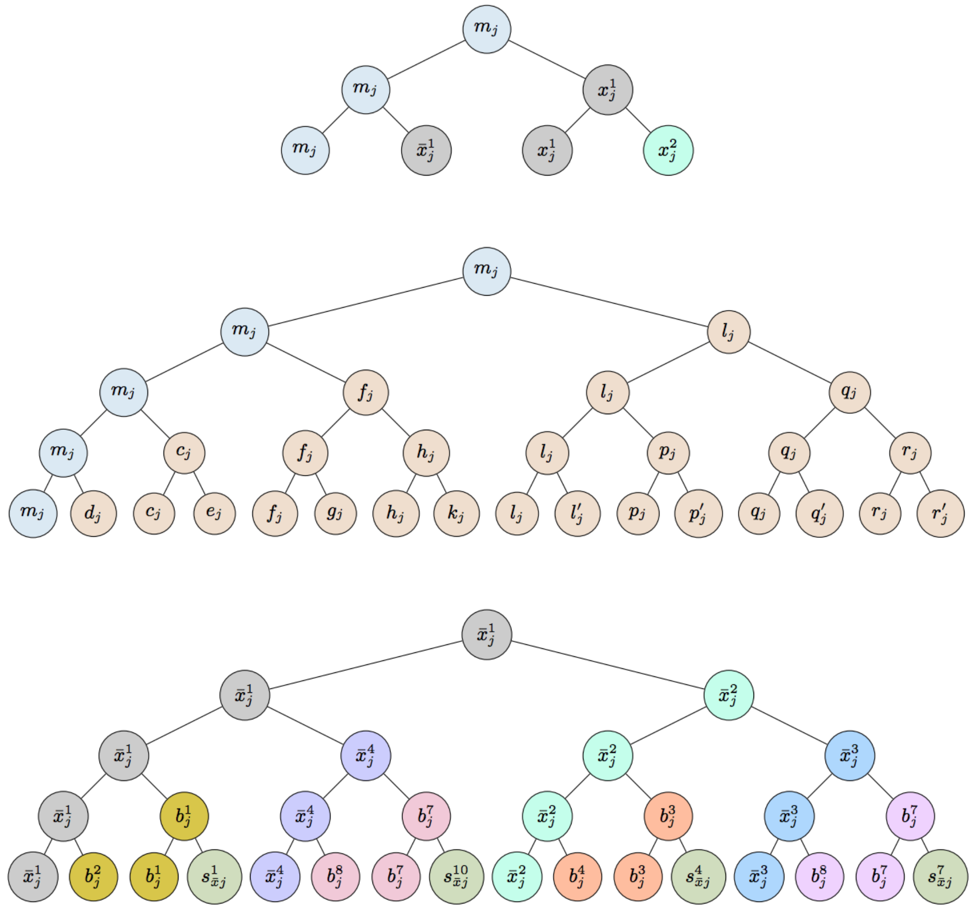
m_j
m_j x_j^1
m_j $\bar{x}_j^1$ x_j^1 x_j^2
m_j
m_j l_j
m_j f_j l_j q_j
m_j c_j f_j h_j l_j p_j q_j r_j
m_j d_j c_j e_j f_j g_j h_j k_j l_j l'_j p_j p'_j q_j q'_j r_j r'_j
$\bar{x}_j^1$
$\bar{x}_j^1$ $\bar{x}_j^2$
$\bar{x}_j^1$ $\bar{x}_j^4$ $\bar{x}_j^2$ $\bar{x}_j^3$
$\bar{x}_j^1$ b_j^1 $\bar{x}_j^4$ b_j^7 $\bar{x}_j^2$ b_j^3 $\bar{x}_j^3$ b_j^7
$\bar{x}_j^1$ b_j^2 b_j^1 $s_{\bar{x}j}^1$ $\bar{x}_j^4$ b_j^8 b_j^7 $s_{\bar{x}j}^{10}$ $\bar{x}_j^2$ b_j^4 b_j^3 $s_{\bar{x}j}^4$ $\bar{x}_j^3$ b_j^8 b_j^7 $s_{\bar{x}j}^7$

Figure 3-12: Part of the 7-round subtournament for choice gadget j in the case where x_j or $\bar{x}_j$ appears at most three times in F ($L = 3$) and $x_j = False$. On the top is the subtournament consisting of rounds 5 through 7 that allows m_j to ultimately win the entire 7-round subtournament. In the middle is the 5-round subtournament that m_j wins, and on the bottom is the 5-round subtournament that $\bar{x}_j^1$ wins.

this means that all players $m_j \in M$ won different $\lceil \log L \rceil + 4$-round subtournaments consisting exactly of the players described earlier. We know that if a player s_j^* is placed anywhere other than in the choice gadget subtournament for player m_j, this yields a contradiction because this means that another player in M would be beaten by s_j^* and fail to advance to round $\lceil \log L \rceil + 4$. Therefore, we have a clear bijection between the players contained in the choice gadget subtournaments and the truth assignment for F: $\psi(x_j) = True$ if m_j's choice gadget subtournament includes all but L players of S_j, and $\psi(x_j) = False$ if m_j's choice gadget subtournament includes all but L players of $S_{\bar{x}_j}$. By a logical extension of the Aziz proof, we see that in order

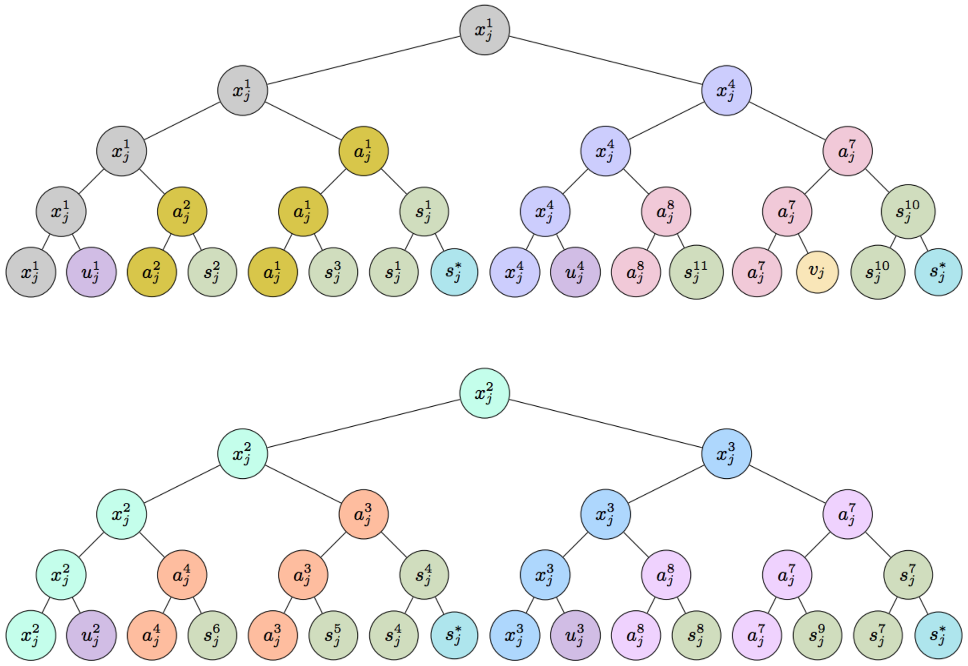
x_j^1
x_j^1 x_j^4
x_j^1 a_j^1 x_j^4 a_j^7
x_j^1 a_j^2 a_j^1 s_j^1 x_j^4 a_j^8 a_j^7 s_j^{10}
x_j^1 u_j^1 a_j^2 s_j^2 a_j^1 s_j^3 s_j^1 s_j^* x_j^4 u_j^4 a_j^8 s_j^{11} a_j^7 v_j s_j^{10} s_j^*
x_j^2
x_j^2 x_j^3
x_j^2 a_j^3 x_j^3 a_j^7
x_j^2 a_j^4 a_j^3 s_j^4 x_j^3 a_j^8 a_j^7 s_j^7
x_j^2 u_j^2 a_j^4 s_j^6 a_j^3 s_j^5 s_j^4 s_j^* x_j^3 u_j^3 a_j^8 s_j^8 a_j^7 s_j^9 s_j^7 s_j^*

Figure 3-13: Continuation of the 7-round subtournament for choice gadget j described by Figure 3-12. On the top is the 5-round subtournament that x_j^1 wins, and on the bottom is the 5-round subtournament that x_j^2 wins.

for player m_ℓ to win the subtournament associated with clause/garbage gadget G_ℓ and clause c_i, the subtournament must contain a player that c/g beats. This player represents one of the literals contained in clause c_i. If $x_j \in c_i$, and $\psi(x_j) = True$, then player c/g beats a player from set S_{x_j}. Likewise, if $\bar{x}_j \in c_i$ and $\psi(x_j) = False$, then player c/g beats a player from set $S_{\bar{x}_j}$. This matching corresponds directly with a satisfiable truth assignment ψ for F. □

Therefore, 3SAT can be reduced directly to TFP in polynomial time. □

We now analyze the size blowup of this reduction. The number of rounds in each of the subtournaments won by players $m_j \in M$ required to isolate all players $m_1, \ldots, m_k$ so that they are the only ones left in the tournament is bounded from below by $\lceil \log L \rceil + 4$, creating a binary tree that recursively breaks down into 5-round subtournaments that correspond to a literal appearing either once or twice. This reduction ultimately adds $2^{\lceil \log L \rceil + 4} < 2L \cdot 16$ nodes to each subtournament. Since the number of rounds in the entire tournament ultimately goes up, the total number of additional nodes becomes $2(32) \cdot |X| \cdot L = 64L \cdot |X|$. We can assume that $L <= |X|$, which means the number of additional nodes is $O(|X|^2)$. However, because we know from Theorem 3.4.1 that we need to add $O(|X|^2)$ edges, from each of these nodes to the others currently in the graph, the reduction blowup becomes $O(|X|^4)$. □

3.6 Comparing the Direct and Indirect Reductions to TFP

We now show that the above reduction is less time-efficient than reducing from 3SAT to 3SAT2 to TFP.

Theorem 3.6.1. *The size blowup created by reducing 3SAT to 3SAT2 to TFP is tighter than the size blowup created by directly reducing 3SAT to TFP.*

Proof. The reduction from 3SAT2 to TFP was given in section 3.3, and its size blowup was given in section 3.4, so we only need to find the size blowup for the reduction from 3SAT to 3SAT2. We first show how to reduce 3SAT to 3SAT2.

Lemma 3.6.2. *There exists a polynomial-time reduction from 3SAT to 3SAT2.*

Proof. Let ϕ be an instance of 3SAT with a set of literals $\{x\}$ such that each x appears $k > 2$ times in ϕ. Replace the first occurrence of x with a literal y_1, the second by y_2, and so on, where $y_1, y_2, \ldots, y_k$ are k new variables. Note that if the opposite literal $\bar{x}$ also appeared $k' > 2$ times in ϕ, we would replace each occurrence with a new literal z so that $z_1, z_2, \ldots, z_{k'}$ are k' new variables.

Now, we add to ϕ the expressions

$$(\bar{y}_1 \vee y_2) \wedge (\bar{y}_2 \vee y_3) \wedge \cdots \wedge (\bar{y}_{k-1} \vee y_k) \wedge (\bar{y}_k \vee y_1),$$
$$(\bar{z}_1 \vee z_2) \wedge ((\bar{z}_2 \vee z_3) \wedge \cdots \wedge (\bar{z}_{k'-1} \vee z_{k'}) \wedge (\bar{z}_{k'} \vee z_1)$$

for x and $\bar{x}$, respectively. Let ϕ' be the result of modifying ϕ in this fashion for each literal appearing more than twice. Clearly, ϕ' has the desired properties of an instance of 3SAT2; however, ϕ' must also be a satisfiable instance of 3SAT2 if and only if ϕ is a satisfiable instance of 3SAT. This is true because for each literal x appearing $k > 2$ times in ϕ, the assignments of the additional variables $y_1, \ldots, y_k$ are the same as the satisfying assignment of x. The expressions above only yield *True* if all k variables have the same assignment (i.e. all *True* or all *False*). It is easy to see that at most $O(n)$ new literals and clauses are added to ϕ in order to create the 3SAT2 instance ϕ', where n is the number of literals in ϕ and each literal appears a constant number of times. $\square$

The number of nodes required for the middle-man reduction becomes $O(n) \cdot O(n^2) = O(n^3)$, while the number of nodes required for the direct reduction is $O(n^4)$. Therefore, it is more efficient to include 3SAT2 in the complete reduction. $\square$

Chapter 4

Parameterized Complexity of Deterministic TFP

4.1 Overview and Motivation

In this chapter, we analyze past work on the application of parameterized complexity to TFP in the deterministic case, as defined in the previous chapter. In particular, we give a more lucid algorithm that matches the tightest known bound on runtime for a given parameter and discuss why a better runtime is currently unlikely to be found.

After a series of smaller publications [1] [6], one of the first comprehensive published works on parameterized complexity appeared in 1999 [7]. But in the past decade, the field has experienced a prodigious transformation into one of the richest and most useful areas of theoretical computer science. In this branch of complexity theory, the inherent difficulty of computational problems is measured with respect to more than one parameter describing the input or output. This allows a finer, more specific classification of NP-hard problems than before. It also motivates the design of algorithms that have an efficient runtime if the parameters of a problem's input are relatively small, regardless of the input size; problems like this are known as *fixed-parameter tractable*, defined below:

Definition 4.1.1. *Consider Σ^*, a set of all possible input strings that can be generated using an alphabet Σ, and some decision problem $A \subseteq \Sigma^*$ with input instance $x \in \Sigma^*$ and parameter k. We say that A is* **fixed-parameter tractable (FPT)** *if there exists an algorithm that decides whether or not x belongs to the set of "yes" or "accept" instances of A in time $f(k) \cdot |x|^{O(1)}$ for some computable function f.*

Several NP-hard problems are indeed fixed-parameter tractable, such as MINIMUM VERTEX COVER, LONGEST PATH, and DISJOINT TRIANGLES [9]. What makes fixed-parameter tractability so attractive is that it precisely cleaves the computational difficulty in the complexity contribution of the parameters so that it is obvious where the congestion(s) in hardness are. For example, the runtime of an FPT algorithm for A from our definition might demonstrat that the parameter k contributes a super-exponential blowup in algorithmic runtime while the original size of the input $|x|$ contributes little blowup by comparison.

Parameters like k tend to arise naturally from the formal structure of the problem in question. For example:

- In a parallel processing system, the runtime of an algorithm running on input data of size n could be parameterized by setting k equal to the number of processors in the system.
- In a database of size n, the time required to answer a query could be parameterized by having k represent the size of the query.
- Many game problems that ask whether a given player has a winning strategy given a particular game, a set of players, and a set of rules are PSPACE-complete. However, many are FPT when parameterized not just by the size n of the input game but also by the number of possible moves k at each step.

The practical world is full of computational problems governed by all sorts of parameters with a variety of bounds. Designing algorithms with runtimes such as $2^k\, n$ or $2^k\, poly(n)$ to solve these problems would be very useful because we could then argue that a problem is solvable in a reasonable amount of time if $k << n$.

4.2 Prior Work

We first must discuss the concept of a binomial arborescence and why it is relevant to both TFP and our work. An example of how a seeding for a balanced knockout tournament is translated into a labeled binomial arborescence is given in Figure 4-1.

Definition 4.2.1. *A* **binomial arborescence** $T = (V(T), E(T))$ *rooted at* $u \in V(T)$ *is defined recursively as follows:*

- *A single node* u *is a binomial arborescence rooted at* u*;*
- *If* $|V(T)| = 2^i$ *for some* $i > 0$*, then* T *is a binomial arborescence rooted at* u *if* $(u, v) \in E(T)$ *for another* $v \in V(T)$ *such that* T_v *is a binomial arborescence rooted at* v *of size* 2^{i-1} *and* $T_u = T\ T_v$ *is a binomial arborescence rooted at* u *of size* 2^{i-1}*.*

Furthermore, given a directed graph $G = (V, E)$*,* $T = (V(T), E(T))$ *is a* **spanning binomial arborescence** *for* G *if* $V(T) = V$*,* $E(T) \subseteq E$*, and* T *is a binomial arborescence.*

It is already known that TFP can be reduced to the problem of finding a spanning binomial arborescence with a specific root. [24] demonstrated that given a balanced knockout tournament G with favorite player $v^* \in V(G)$, there exists a seeding of the vertices in G that results in a victory for v^* if and only if G has a spanning binomial arborescence rooted at v^*. For the rest of this chapter, we will be investigating TFP using this connection between finding a winning seeding for v^* and spanning binomial arborescences.

The fastest FPT algorithm for solving TFP runs in time $2^{O(k \log k)} n^{O(1)}$ where n is the number of players and k is the size of the minimal feedback arc set [12]. Recall that in a directed graph $G = (V, E)$, a subset $F(E) \subseteq E$ is a minimal feedback arc set if and only if the tournament obtained after reversing the arcs in $F(E)$ is acyclic. This set can be found in time $2^{O(\sqrt{k})} n^{O(1)}$ [8] [16]. The algorithm for solving TFP begins by guessing a template tree that consists of all possible ways to place in a blank binomial arborescence the nodes in the feedback vertex set $F(V)$, which is the set of vertices

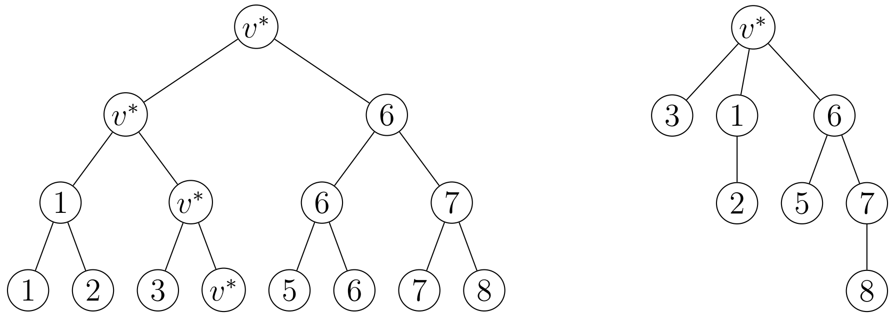

Figure 4-1: A balanced knockout tournament with $n = 8$ players and $v^* = 4$ with the results by round displayed as a binary tree (left) and a binomial arborescence (right).

adjacent to arcs in $F(E)$. There are $O((\log n)^k) = 2^{O(k \log k)}$ different permutations of binomial arborescences to iterate over. Once a guessed template has been verified as a valid partial labeling of a binomial arborescence rooted at the favorite player according to the dictations of the input directed graph, the algorithm iterates over the paths and subtrees not already determined by the guess and fills them up in a greedy fashion.

In order to understand the basics of this algorithm, as well as why the one we give later on in section 4.4 is simpler, we must first understand the relationship between a labeled spanning binomial arborescence and a "template" (as it is called in [12]).

Definition 4.2.2. (**LCA closure**)*: For a rooted tree T and a subset $S \subseteq V(T)$, the* **least common ancestor-closure (LCA-closure)** $\mathsf{LCA}(S)$ *is obtained by the following process:*

1. *Set $S' = S$.*
2. *While there are still vertices $x, y \in S'$ whose least-common ancestor w in T is not in S, add w to S'.*
3. *Finally, when this process is complete, output $S' = \mathsf{LCA}(S)$. Note that $\mathsf{LCA}(S) \leq 2|S|$.*

Having defined a least common ancestor-closure, which is crucial to understanding how a template is generated from a binomial arborescence, we restate the formal definition of a template and the procedure by which a binomial arborescence T is dissected and altered in order to create a template given a subset F of the nodes in the arborescence.

Definition 4.2.3. (**Template Trees**)*: Let T be a spanning binomial arborescence rooted at v^*, let $F \subseteq V(T)$, and let V_F be the set of vertices adjacent to the arcs in F (a.k.a. the feedback vertex set). We construct a* **template tree** *T' of F from T:*

1. *Set $T' = T$.*
2. *For each $u \in \mathsf{LCA}(F)$, delete from T' all vertices that do not lie on any of the paths in T' from v^* to u.*
3. *While there there is still a vertex $u \in V(T') \setminus \mathsf{LCA}(F)$ with exactly one in-neighbor v and exactly one out-neighbor w, delete u from T' and replace it and the arcs (v, u) and (u, w) with the arc (v, w).*
4. *Delete the labels of all nodes in T' except for those of the vertices in F.*
5. *The resulting tree T' is a template tree of F.*

A given tree T' is a template tree of F if it is a template tree of F in at least one spanning binomial arborescence T rooted at v^*.

Figures 4-2 and 4-3 give an example of how the a template T' of $F(V) \cup \{v^*\}$ might be generated from a spanning binomial arborescence T of 16 nodes, related to each other by a directed graph $G = (V, E)$, with root v^* and $F(V) = \{v_2, v_8, v_{10}, v_{13}, v_{15}\}$.

We now describe the intuition for this algorithm. Let (G, v^*, k) be an accepted instance of TFP, meaning that there exists a spanning binomial arborescence T rooted at v^*. First, iterate over all possible template trees of $F(V) \cup \{v^*\}$ in T. The total number of template trees to check is $k^{O(k)}$, and this superset of possible trees enumerated in time $k^{O(k)}$. In addition to guessing a template T' that can be generated from the arborescence T (which we do not know), we also have to guess the modifications that were made to it in order to turn T into T'. This is done in two ways:

1. For each arc $(u, v) \in E(T')$, we guess all possible $\ell \in \{0, \cdots, \log n\}$. ℓ represents the number of arcs that were removed from the path in T from u to v.

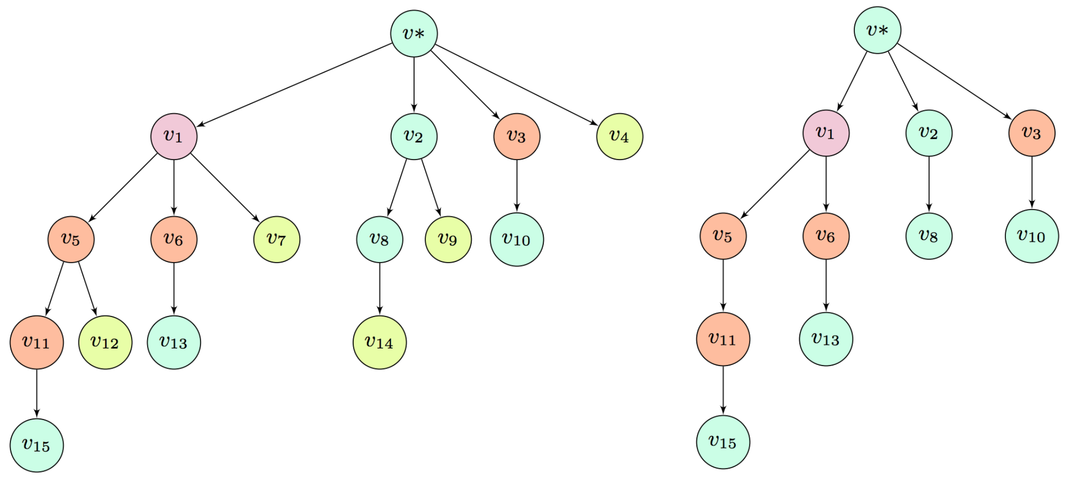

Figure 4-2: On the left is the original spanning binomial arborescence $T' = T$ rooted at v^*. The blue vertices belong to the set $F(V) \cup \{v^*\}$, the pink vertex v_1 belongs to the set $\mathsf{LCA}(F(V) \cup \{v^*\}) \setminus (F(V) \cup \{v^*\})$ whose label is removed in Step 4 of the template-generating process, the orange vertices belong to the set of vertices removed from T' in Step 2, and the green vertices belong to the set of vertices removed from T' in Step 3. On the right is tree T' after the orange vertices have been removed, in completion of Step 2 in the template-generating process.

2. For each vertex $u \in F(V) \cup \{v^*\}$, we guess all possible $m \in \{2^0, 2^1, \cdots, 2^{\log n}\}$. m represents the size of the subtree rooted at u in T.

The process of guessing a valid template takes time $2^{O(k \log k)} n^{O(1)}$. Each guess is then checked to see if a binomial arborescence rooted at v^* that represents a winning seeding could be constructed from the guessed template. This verification takes polynomial time, so these first two stages are completed in time $2^{O(k \log k)} n^{O(1)}$.

As soon as a template tree T' of $F(V) \cup v^*$ in T is found, we greedily fill in the missing nodes and arcs, starting at the top of T' and moving downward. First, the missing arcs between paths are restored, reversing Step 3 of the template-building process described in the definition. Then, the remaining missing nodes are restored, reversing Step 2 of the process. Both of these restorative steps are done in polynomial time. The result is a spanning binomial arborescence rooted at v^*.

The runtime blowup of this algorithm comes from the first stage, when we have to enumerate all possible permutations of the set $F(V) \cup \{v^*\}$ in the binomial arborescence. Once a valid labeling of these nodes in the arborescence has been found, the remaining nodes can be labeled efficiently.

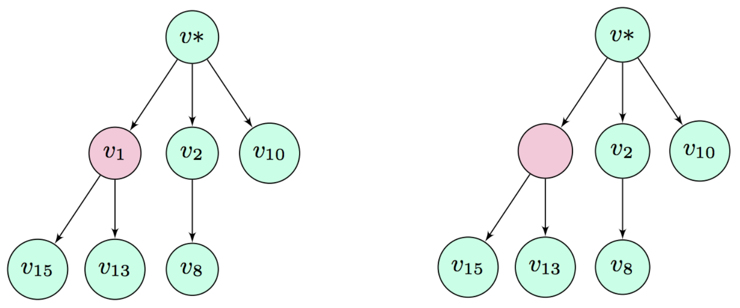

Figure 4-3: On the left is tree T' after the green nodes have been removed, in completion of Step 3 in the template-generating process. On the right is tree T' after the the pink vertex's label has been deleted, in completion of Step 4 in the template-generating process. The result is the new template tree T' of $F(V) \cup \{v^*\}$ created from arborescence T.

4.3 Towards a Lower Parameterized Complexity

An FPT algorithm for solving TFP that runs in time $2^{O(k)}n^{O(1)}$ would be much more attractive. At first glance, it seems possible that we could use a graph problem with some parameter k that can be solved in this runtime to come up with such an algorithm. One work that was investigated to see if this was possible is Alon et al. [2], which presented efficient, FPT algorithms for finding simple paths and cycles in graphs using the color-coding method. Suppose we have a graph $G = (V, E)$ with a random coloring of the vertices of G with k colors. A path in G is said to be *colorful* if each vertex on it is assigned a distinct color. A colorful path in G of length k can be found in $2^{O(k)} \cdot |E|$ worst-case time. Similarly, given a directed or undirected forest H on k vertices and a graph $G = (V, E)$, a subgraph of G isomorphic to H, if one exists, can be found in $2^{O(k)} \cdot |E|$ expected time in the directed case, and in $2^{O(k)} \cdot |V|$ expected time in the undirected case.

It is not hard to see that the binomial arborescence and the tournament graph given in Figure 4-1 are isomorphic to each other. Using this fact, several attempts were made to reduce the problem of finding a spanning binomial arborescence rooted at the node representing our favorite player to the problem of finding a subgraph of a graph G isomorphic to a directed forest H. More specifically, we attempted to reduce the problem of finding a valid labeling of the nodes $F(V) \cup \{v^*\}$ in the arborescence to the color-coding problem, where the H is the forest denoted by the edges in the tournament graph T connecting the vertices of $F(V) \cup \{v^*\}$. The number of colors used k' would be equal to $O(k)$ where k is the size of the feedback arc set, and $|F(V)| \leq 2k$. This way, the parameterized complexity of finding a valid enumeration would be tightened to $2^{O(k)}n^{O(1)}$ rather than $2^{O(k \log k)}n^{O(1)}$.

However, the binomial arborescence rooted at v^* representing a winning seeding is constructed with the consideration of the order of the vertices in the arborescence. That is, in order to be correct, the labels must be added so that each successive level and/or subtree can still have a valid labeling. Verifying the correctness of a guessed labeling of $F(V) \cup \{v^*\}$ in the blank arborescence can be done in polynomial time, which is optimal. The decisions made by the algorithms presented in Alon et al. rely only on the vertices having distinct colors, regardless of their ordering or arrangement in the input graph [2]. The necessity of considering the ordering of nodes in the graph when labeling $F(V) \cup \{v^*\}$ is a feature of TFP whose additional intricacy and complexity is incompatible with those algorithms. As an analogy, recall that a single combination (n, m) indifferent to its elements' ordering can be broken down into a series of permutations whose union creates that combination. Attempting to represent a single one of those permutations with some group of combinations is already difficult, but the complexity of representing an exponential number of permutations in this way is even harder, and completely contradicts what we attempted to do: tighten the hardness lower bound for finding the correct permutation from $2^{O(k \log k)}n^{O(1)}$ down to $2^{O(k)}n^{O(1)}$.

We investigated other publications that gave FPT algorithms for graph problems that were parameterized by the size of the input graph's feedback arc set. The problems included finding k-cuts of graphs, finding pathwidth, and finding the optimal linear arrangement of the graph's vertices. Some of the algorithms investigated had runtimes that were slower than $2^{O(k)}n^{O(1)}$ [11] [21]. In the cases where the algorithms were fast enough, we attempted to reduce TFP to the problem(s) being investigated in the publication, but again the aforementioned incompatibility of ignoring the ordering reared its head [4] [10].

It is still possible that a parameterized algorithm with runtime $2^{O(k)}n^{O(1)}$ can be found for TFP, with k equal to the size of the feedback arc set in the input tournament graph. However, we can conclude from our investigations that this algorithm must cut down the number of guessed arrangements of the vertices that comprise $F(V) \cup \{v^*\}$ in the arborescence dramatically. Furthermore, if this algorithm relies upon a reduction from the problem of finding a valid arrangement of those nodes to another graph problem (which may or may not be parameterized according to the size of the feedback arc set), the latter problem and the algorithm that solves it must be compatible with TFP's intricate reliance upon permutations and ordering in the binomial arborescence. The FPT algorithm given in the previous section makes the process of finding a correct permutation even more complicated by guessing a correct template, rather than the positions of the nodes in $F(V) \cup \{v^*\}$ in the arborescence, but without reducing the complexity of this guessing stage. It would be advisable for a future algorithm to avoid unnecessary changes such as this.

4.4 Our FPT Algorithm

We give a simpler FPT algorithm for TFP with runtime $2^{O(k \log k)} n^{O(1)}$, where n is the number of players and k is the size of the minimal feedback arc set. As in Gupta et. al [12], our input is a favorite player v^*, and a directed graph $G = (V, E)$. We will also assume that the feedback arc set $F(E)$ (and consequently the feedback vertex set $F(V)$) is known. First, the algorithm will iterate over all possible arrangements of the feedback vertex set $F(V)$ in the blank binomial arborescence B. However, instead of deleting blank nodes and formulating these permutations as "templates," we will instead treat them as spanning binomial arborescences over all n nodes in G with only the positions of the vertices $v \in F(V) \cup \{v^*\}$ labeled and with v^* at the root. There are three combinatorial pieces that determine the number of iterations:

- Which arcs in $F(E)$ are included. There are k to choose from, so there is a total of 2^k possibilities.

- For each arc $(u, v) \in F$, guess an integer in $\{0, \dots, \log n\}$ representing the distance between u and v in B. We enumerate over all possible lengths, yielding a total of $(\log n^{2k}) \cdot 2^k = 2^{k \log k}$ iterations.

- For each vertex $v \in F(V)$, guess the size of the subtree rooted at v in B, ranging from $\{2^0, \ldots, 2^{\log n}\}$. This also yields a total of $2^{k \log k}$ iterations.

Given a guessed partial labeling S of a blank spanning binomial arborescence B with labeled nodes $V(S) = F(V) \cup \{v^*\}$ and edges $E(S)$, a favorite player v^*, $F(V)$, and the original tournament graph $G = (V, E)$, we must verify whether or not the labeling is valid (i.e. whether nor not it can yield a winning binomial arborescence). We do this using the procedure Verify.

Algorithm 1 Verify$(S, F(V), v^*)$

```
1: $i = 0$, $paths = \{\}$
2: for $w \in out(v^*)$ do
3:     if $w \notin F(V)$ then
4:         add $(v^*, w)$ to $paths$
5:     end if
6:     if $w \in F(V)$ and $(v^*, w) \in E(S)$ then
7:         add $(v^*, w)$ to $paths$
8:     end if
9: end for
10: for $i \in [1, \log n]$ do
11:     $newPaths = \{\}$
12:     for $path \in paths$ with terminus $u \in V$ do
13:         for $w \in out(u)$ do
14:             if $w \notin F(V)$ then
15:                 $path = path + (u, w)$
16:                 add $path$ to $newPaths$
17:             end if
18:             if $w \in F(V)$ and $path$ is existable in $S$ with $w$ at distance $i$ from $v^*$ then
19:                 $path = path + (u, w)$
20:                 add $path$ to $newPaths$
21:             end if
22:         end for
23:     end for
24:     $paths = newPaths$
25: end for
26: ACCEPT if all vertices in $V$ are visited, REJECT otherwise.
```

This verification algorithm runs in time $poly(n)$. Once we have guessed a labeling S and the verification algorithm accepts the guess as a feasible partial labeling of the spanning binomial arborescence, we fill in the at most $n - k$ remaining blank nodes. This too can be done in polynomial time not by greedily filling in a more complicated template but instead by setting boundaries on what vertices can go where, and then filling in the blank nodes starting with the top of the tree and slowly moving down.

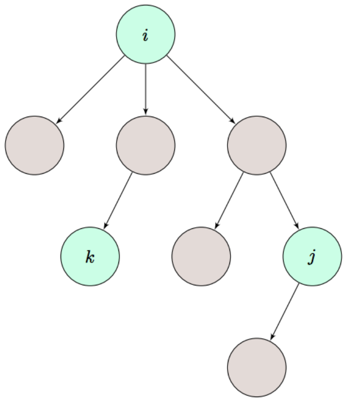

Figure 4-4: Spanning binomial arborescence rooted at i and labeled only with a guess for $F(V) = \{i, j, k\}$. The blue-green nodes are labeled with a guessed permutation of the nodes of $F(V)$, and the off-white nodes are at present unlabeled.

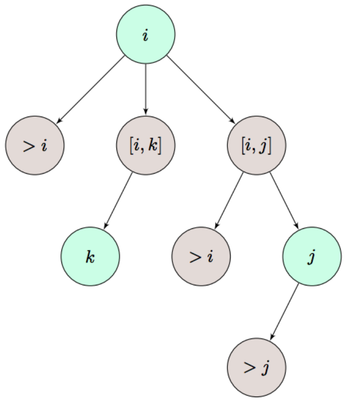

Figure 4-5: Spanning binomial arborescence rooted at i. The blue-green nodes are labeled with $i, j, k \in F(V)$, and the off-white nodes are labeled with the bounds on topological rank that the vertex placed there must satisfy.

Figure 4-4 shows an 8-node spanning binomial arborescence rooted at i. We will treat this arborescence as a subtree of a valid partial labeling S that includes the nodes $i, j, k \in F(V)$. The remaining nodes are blank. To understand how the interval method of labeling a spanning binomial arborescence works, consider that all vertices in V are labeled 1 to n in a topological ordering where if $u < v$, u beats v. This is represented in the input tournament graph by an edge $(u, v) \in E$. The vertices in $F(V)$ violate this ordering because they each have at least one edge in the

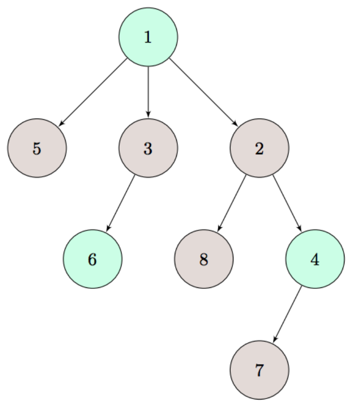

Figure 4-6: Possible correct labeling of a spanning binomial arborescence rooted at i with $1, 4, 6 \in F(V)$.

feedback arc set, and if $F(E)$ were removed from E, G would be acylic. Therefore, the remaining vertices in $V \backslash F(V)$ can all be placed in S according to the dictations of this topological ordering as well as the intervals created by where the nodes in $F(V)$ have already been placed. Figure 4-5 shows an 8-node spanning binomial arborescence with $i, j, k \in F(V)$ labeled and the remaining nodes marked according to which intervals their topological ranking should fall in. A node labeled $> i$ can only be labeled with available vertices $v \in V$ such that $\sigma(v) > i$. For example, if we let $i = 1, j = 4, k = 6$, then a possible correct winning labeling for i that satisfies the bounds on topological rank set in Figure 4-5 is given by Figure 4-6.

We now give the main algorithm CompleteSBA for labeling the off-white nodes using the topological ranking σ, the feasible partial labeling S, and G. Upon the algorithm's completion, B is completely and correctly labeled with v^* at the root. We also give the helper algorithm UpdateBounds for updating the upper and lower bounds of the available intervals for the unlabeled nodes in B S, where B is the entire arborescence and S is the set of labeled nodes. We repeatedly call this function as B is labeled from the top down.

To show CompleteSBA in action, consider a graph $G = (V, E)$, a topological ranking $\sigma(v_i) = i$, and a binomial arborescence B with a feasible partial labeling $S = F(V) \cup \{v^*\}$ where $n = 16$, $v^* = v_2$ and $F(V) = \{v_1, v_8, v_{10}, v_{13}, v_{15}\}$. Note that for all pairs v_i, v_j with $i < j$ whose edge is not in feedback arc set, $(i, j) \in E$. The

Algorithm 2 CompleteSBA$(B, G, F(V), \sigma, v^*)$

```
1: $L = \min(V \setminus (V(F) \cup \{v^*\}))$
2: $S = V(F) \cup \{v^*\}$
3: $i = 1$
4: UpdateBounds$(B, G, S, \sigma, v^*, L)$
5: while $B \setminus S \neq \emptyset$ and $i \leq \log n$ do
6:     for $u \in B[i]$ where $B[i]$ is the set of unlabeled nodes in $B$ at level $i$, sorted by increasing interval size do
7:         if $\exists$ node $v \in B \setminus S$ with interval range $R$ such that $|R| = 1$ then
8:             Label $v$ with $r \in R$.
9:             $S = S \cup \{v\}$
10:            UpdateBounds$(B, G, S, \sigma, v^*, L)$
11:        end if
12:        Label $u$ with the smallest $\sigma(j) > L$
13:        $S = S \cup \{v\}$
14:        $L = L + 1$
15:    end for
16:    UpdateBounds$(B, G, S, \sigma, v^*, L)$
17:    $i = i + 1$
18: end while
19:
20: return $(B, S)$
```

Algorithm 3 UpdateBounds$(B, G, S, \sigma, v^*, L)$

```
1: for unlabeled $u \in B \setminus S$ do
2:     if $in(u) > L$ and $in(u) \in S$ then
3:         add the lower bound $(L, \cdot)$ to $u$
4:     else
5:         add the lower bound $(in(u), \cdot)$ to $u$
6:     end if
7:     if $\exists v \in out(u)$ such that $v \in S$ then
8:         add the upper bound $(\cdot, v)$ to $u$
9:     else
10:        add the upper bound $(\cdot, n)$ to $u$
11:    end if
12: end for
```

algorithm proceeds in Figures 4-7 through 4-12.

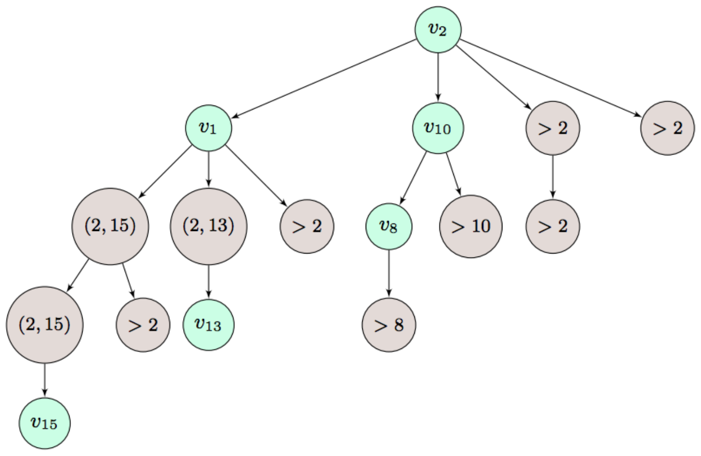

Figure 4-7: Spanning binomial arborescence B rooted at $v^* = v_2$. The blue-green nodes are labeled with $F(V) = \{v_1, v_8, v_{10}, v_{13}, v_{15}\}$, and the off-white nodes are labeled with the bounds on topological rank $i = \sigma(v_i)$. At this point $L = 2$.

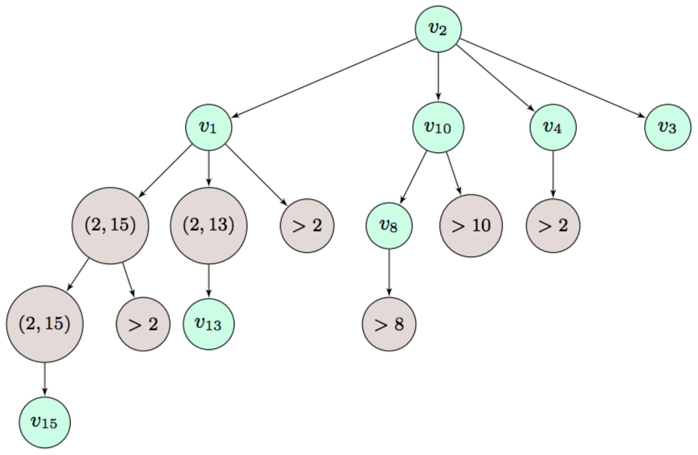

Figure 4-8: Partial labeling of B resulting from the iteration $i = 1$, filling in the nodes at the first level. Now $S = \{v_1, v_2, v_3, v_4, v_8, v_{10}, v_{13}, v_{15}\}$ and $L = 4$.

It is not hard to see that each call to UpdateBounds is completed in linear time. The number of unfilled nodes in B at level i is bounded from above by $\binom{\log n}{i}$. Because v^* has already been labeled at the root, we can set the boundary $i \in [1, \log n]$. UpdateBound is called once at the end of each iteration. During each visit, we check to see if there exists a node in $B \setminus S$ with an interval size equal to 1 (meaning there exists only one feasible label for it). Therefore, each visit to an unlabeled node also

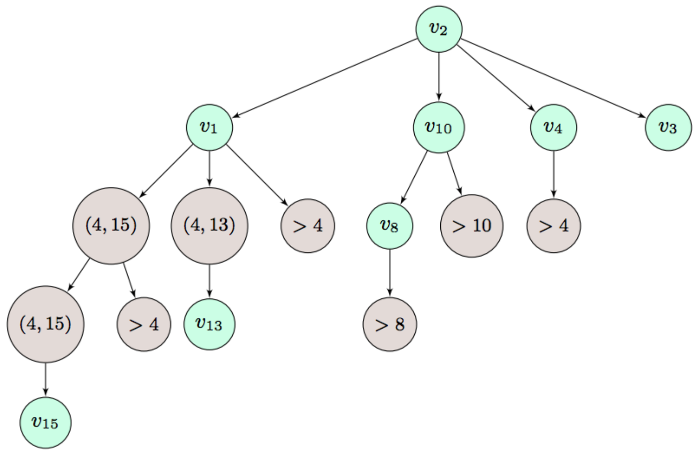

Figure 4-9: Update of the lower bounds in the remaining nodes in $B \setminus S$ given the the new labels of Figure 4-8.

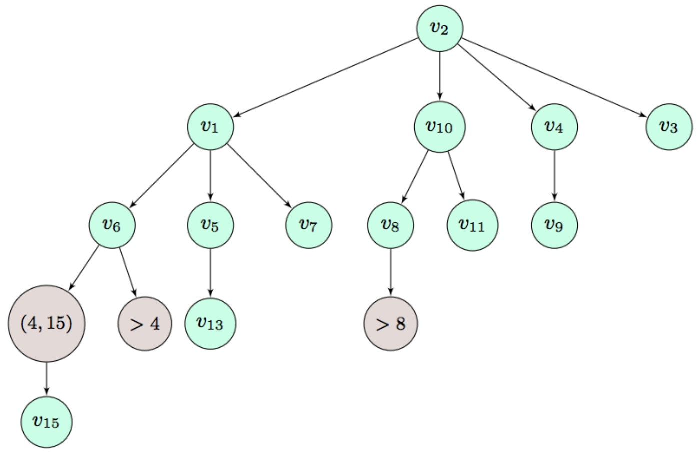

Figure 4-10: Partial labeling of B resulting from the iteration $i = 2$, filling in the nodes at the second level. Now $S = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}\}$ and $L = 11$.

takes linear time. So the total runtime of CompleteSBA is

$$\sum_{i=1}^{\log n} \left(\binom{\log n}{i} \cdot O(n) + O(n) \right) = O(n^2) + O(n \log n) = O(n^2)$$

The total runtime of our algorithm for solving TFP is therefore $2^{k \log k} n^{O(1)}$, matching the runtime of the algorithm given by [12], but with a much cleaner and less convoluted implementation. Because the runtime is exponential in our parameter k and polynomial in the size of the input n, it is also fixed-parameter tractable.

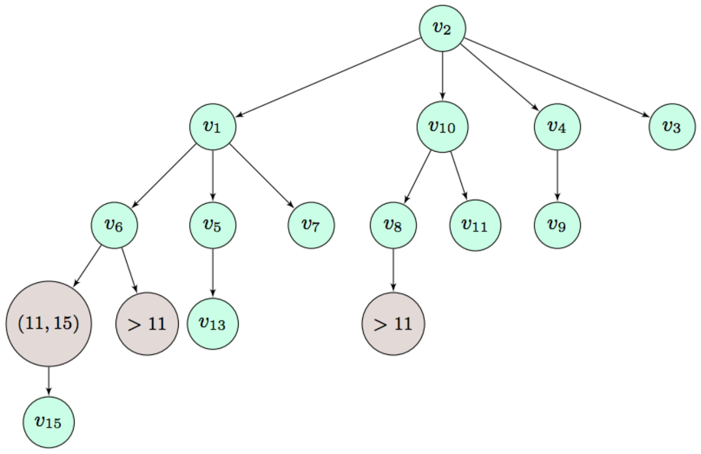

Figure 4-11: Update of the lower bounds in the remaining nodes in $B \setminus S$ given the new labels of Figure 4-10.

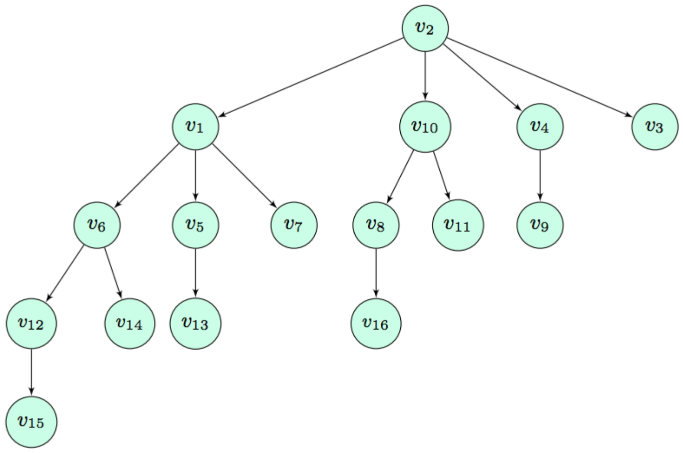

Figure 4-12: Complete correct labeling for B, S at the conclusion of CompleteSBA. Because the lone node at the level $i = 4$ has already been labeled by $v_{15} \in F(V)$, the algorithm will terminate as soon as all of the nodes at level $i = 3$ have been assigned labels because $B \setminus S = \emptyset$.

Chapter 5

Conclusions and Future Work

This thesis examined the problem of fixing tournaments using several techniques. In this section, we summarize some specific directions and open problems for future work. We demonstrated that even a knockout tournament described by a monotonic probabilistic matrix can be fixed with the addition of bribery, yielding an NP-complete problem from an open problem. However, the question of whether or not it is NP-hard to fix a tournament in the monotonic case without the aid of bribery still remains open and acute. Worthwhile future work would include finding a polynomial-time reduction to this problem from a known NP-hard problem. Another possible direction would be to parameterize it according to one or more parameters that quantify the monotonicity of the matrix P since it seems that the inclusion of monotonicity specifically encumbers the establishment of TFP's complexity.

We also showed that a knockout tournament described by the Condorcet random model will almost always have an optimally small set of top players that can be bribed so that a winning seeding can be found efficiently for any player in the tournament. Having a small set of players whom the organizer will always be able to bribe that is (1) independent of the precise structural details of the tournament graph, which are subject to the noise of the Condorcet model, and (2) available before any winning seedings have been found is new and compelling. Possible future work on this idea would include applying it to deterministic tournaments with more complicated structures or parameterizing the complexity of various bribery problems with respect to the size of this set. It would also be interesting to apply this idea to knockout tournaments described by probabilistic matrices rather than deterministic ones and see how the number of players needed in the set changes.

This thesis also examined the implications of the known NP-completeness of deterministic TFP, beginning with attempts to reduce other NP-complete problems to it. Possible future work would be to find other Boolean decision problems with constraints similar to the one imposed on 3SAT2 that ultimately allowed it to be reduced successfully to TFP in polynomial time. I considered investigating whether or not HAMPATH could be reduced to TFP, but I suspect that the relationship between vertices and edges in a correct instance of HAMPATH would cause a massive blowup in the reduction size to the point where a poly-time NP-hardness reduction would be infeasible. Possible future work would include seeing whether or not this is the case for other NP-complete problems. The most attractive problems would have an inherent language restriction comparable to that of 3SAT2. As we have seen, restricting each literal in a satisfiable instance of 3SAT2 to appear at most twice allowed us to keep the tournament graph constructed from the instance to polynomial size. It follows that problems that have similarly strict features would be more likely to yield efficient reductions to TFP.

Finally, this thesis examined the application of parameterized complexity to deterministic TFP by attempting to lower the tightest known bound on TFP's complexity using the size of the minimal feedback arc set as a parameter. Possible future work includes finding a more efficient way to iterate over the possible partial labelings of spanning binomial arborescences, since this is what causes the running time of the iterative stage (and therefore the entirety) of both the original algorithm [12] and the simpler algorithm presented in this thesis to skyrocket. If the number of iterations could be reduced from $2^{O(k \log k)}$ to $2^{O(k)}$, this would yield a much better runtime of $2^{O(k)}n^{O(1)}$. Another future direction would be choosing a different property of the input tournament graph as a parameter and establishing a (possibly) tighter and more attractive lower bound that way.

Bibliography

[1] K.R. Abrahamson, M.R. Fellows, J.A. Ellis, and M.E. Mata. On the complexity of fixed parameter problems. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*. IEEE, 1989.

[2] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[3] Haris Aziz, Serge Gaspers, Simon Mackenzie, Nicholas Mattei, Paul Stursberg, and Toby Walsh. Fixing a balanced knockout tournament. In *Proc. AAAI*, pages 552–558, 2014.

[4] Florian Barbero, Christophe Paul, and Michal Pilipczuk. Exploring the complexity of layout parameters in tournaments and semi-complete digraphs. In *ArXiv*, 2017.

[5] M. Braverman and E. Mossel. Noisy sorting without resampling. In *Proc. SODA*, pages 268–276, 2008.

[6] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness. *SIAM Journal on Computing*, 24(4):873 – 921, 1995.

[7] Rod G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer, 1999.

[8] Feige. Faster fast, 2009.

[9] Michael R. Fellows. Parameterized complexity: Main ideas, connections to heuristics and research frontiers. In *Proceedings of ISAAC*, volume 2223 of *Lecture Notes in Computer Science*, pages 291–307. Springer-Verlag, 2001.

[10] Fedor V. Fomin and Michal Pilipczuk. Subexponential parameterized algorithm for computing the cutwidth of a semi-complete digraph. In *Proceedings of the Twenty-First European Symposia on Algorithms (ESA 2013)*, pages 505–516, 2013.

[11] Archontia C. Giannopoulou, Michal Pilipczuk, Jean-Florent Raymond, Dimitrios M. Thilikos, and Marcin Wrochna. Cutwidth: obstructions and algorithmic aspects. In *ArXiv*, 2017.

[12] 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)*, pages 275–281, 2018.

[13] Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Winning a tournament by any means necessary. In *Proc. IJCAI*, pages 282–288, 2018.

[14] J. Horen and R. Riezman. Comparing draws for single elimination tournaments. *Op. Research*, 33(2), 1985.

[15] F. K. Hwang. New concepts in seeding knockout tournaments. *The American Mathematical Monthly*, 89(4):235–239, 1982.

[16] Karpinski and Schudy. Faster algorithms for feedback arc set tournament, kemeny rank aggregation and betweenness tournament. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC (2010)*, pages 3–14, 2010.

[17] M. P. Kim and V. Vassilevska Williams. Fixing tournaments for kings, chokers, and more. In *Proc. IJCAI*, pages 561–567, 2015.

[18] Michael P. Kim, Warut Suksompong, and Virginia Vassilevska Williams. Who can win a single-elimination tournament? *ArXiv*, 2015.

[19] N. Mattei, J. Goldsmith, and A. Klapper. On the complexity of bribery and manipulation in tournaments with uncertain information. In *Proceedings of the 25th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2012)*, 2012.

[20] J. W. Moon and N. J. Pullman. On generalized tournament matrices. *SIAM Review*, 12(3):384–399, 1970.

[21] Michal Pilipczuk. Computing cutwidth and pathwidth of semi-complete digraphs via degree orderings. In *ArXiv*, 2012.

[22] T. Russell and T. Walsh. Manipulating tournaments in cup and round robin competitions. In *Proc. ADT*, 2009.

[23] I. Stanton and V. Vassilevska Williams. Manipulating stochastically generated single-elimination tournaments for nearly all players. In *Proc. WINE*, pages 326–337, 2011.

[24] V. Vassilevska Williams. Fixing a tournament. In *Proc. AAAI*, pages 895–900, 2010.

[25] T. Vu, A. Altman, and Y. Shoham. On the complexity of schedule control problems for knockout tournaments. In *Proc. AAMAS*, pages 225–232, 2009.

[26] T. Vu, N. Hazon, A. Altman, S. Kraus, Y. Shoham, and M. Wooldridge. On the complexity of schedule control problems for knock-out tournaments. *Submitted to JAIR (Preliminary versions in AAMAS 2009 and COMSOC 2008)*, 2010.