# Pairing Things Off: Counting Stable Matchings, Finding Your Ride, and Designing Tournaments

Robbie Weber

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Anna R. Karlin, Chair

Shayan Oveis Gharan, Chair

Dan Grossman

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

## Abstract

Pairing Things Off: Counting Stable Matchings, Finding Your Ride, and Designing Tournaments

Robbie Weber

Co-Chairs of the Supervisory Committee:
Professor Anna R. Karlin
Paul G. Allen School of Computer Science & Engineering

Assistant Professor Shayan Oveis Gharan
Paul G. Allen School of Computer Science & Engineering

This thesis discusses three problems around the common theme of pairing off agents. While the techniques vary, in each problem we observe that careful application of classical and well-understood techniques can still lead to progress.

In Chapter Two, we study the classical combinatorial problem of stable matchings. Stable matchings were introduced in 1962 in a seminal paper by Gale and Shapley. In this chapter, we provide a new upper bound on $f(n)$, the maximum number of stable matchings that an instance with $n$ men and $n$ women can have. The best lower bound prior to our work was approximately $2.28^n$, and the best upper bound was $2^{n \log n - \mathcal{O}(n)}$. We show that for all $n$, $f(n) \leq c^n$ for some universal constant $c$. Our bound matches the lower bound up to the base of the exponent.

In Chapter Three, we discuss online matching. The Min-Cost Perfect Matching with Delays Problem (MPMD) has been the subject of a recent flurry of activity in algorithm design. In the problem, a series of requests appear over time in a metric space, with the locations and timing determined by an adversary. The algorithm designer is charged with pairing off all the requests, attempting to minimize the sum of the amount of time the

requests have to wait (between appearance and being matched) and the distances between the matched requests.

We discuss our initial work on the problem, which shows that if the requests come from an unknown probability distribution (rather than from an adversary) a simple algorithm (that we call "ball growing") achieves excellent performance. We also adapt the core idea of the algorithm to the adversarial case, and show that a slight modification of the same simple idea suffices to match the best-known algorithms for MPMD.

In Chapter Four, we discuss a different type of matching – creating matchups in sports tournaments. Our work is inspired by an incident in the 2012 Olympic Badminton tournament where both teams playing in a match were incentivized to lose that match (and attempted to do so). In 2016, the tournament was redesigned, with the stated goal of eliminating misaligned incentives; we show the redesign failed in this goal. We then describe a minimally-manipulable tournament rule which could be reasonably implemented, while maintaining many of the subtler features of the current tournament that a designer would want.

# TABLE OF CONTENTS

# LIST OF FIGURES

# ACKNOWLEDGMENTS

This dissertation would not exist without both of my advisors. Shayan: thank you for your constant enthusiasm, for always pushing for stronger and better results, and for thinking about the big picture. Anna: thank you for your consistent patience, for keeping me on track when I needed it, and for always working out step one with me while Shayan worked on step three.

I am incredibly lucky that I have been able to mix teaching and research during my Ph.D. I am indebted to my advisors for letting me balance those two goals and to Dan Grossman for giving me a chance to teach. Our entire group of teaching faculty has been incredibly generous in mentoring me. Particular thanks go to Ruth Anderson and Lauren Bricker, for their advice on my trickiest teaching issues. To Brett Wortzman and Hunter Schafer, for making me feel included and welcome. And to Kasey Champion for her unmatched modeling of 'hustle' and prowess with powerpoint animations.

I would not have gotten though my Ph.D. without the support of my fellow students at UW. My thanks go in particular to Kira Goldner for giving me pep talks when I needed them most (and glares when I needed those most). To Jennifer Brennan, whose work ethic is inspiring. To John Thickstun, for being a sounding board and always finding the point in a complicated argument. To Swati Padmanabhan, for all of our coffee breaks and walk-and-talks. And to Sami Davies, Anna Kornfield Simpson, Leah Perlmutter, Christine Chen, and countless others for finding the time for practice talks and hallway conversations.

Finally, thanks to my family: my parents for always being there to listen, and to Kristi for ensuring our email correspondence was always at least weekly.

# Chapter 1

# INTRODUCTION

This thesis discusses three problems of pairing agents. In Chapter 2, we show that a stable matching instance with $n$ agents on each side can have at most $c^n$ stable matchings (where $c$ is a universal constant). Chapter 2 is based on a paper with Anna Karlin and Shayan Oveis Gharan that appeared in STOC 2018 [KOGW18].

In Chapter 3, we discuss a simple idea for solving online matching problems. We show that algorithms based on this idea can achieve excellent performance when requests are stochastic, and can match the best-known algorithms when requests are adversarial. Chapter 3 is based on a manuscript written with Anna Karlin, Shayan Oveis Gharan, and Alireza Rezaei.

In Chapter 4, we design a practical and approximately strategy-proof tournament for use in competitions like Olympic Badminton (which have experienced teams manipulating match results). Our tournaments match strategy-proofness results already in the literature while maintaining practical benefits that other tournaments have lacked. Chapter 4 is based on a manuscript written with Nick Liu, Jainul Vaghasia, and Sierra Wang.

## *1.1 Counting Stable Matchings*

A stable matching instance is formed by two sets of $n$ agents (commonly referred to as "men" and "women"), with each agent holding a preference list ordering (all of) the agents in the other set. A stable matching pairs each man to exactly one woman, such that there is no *blocking pair*. A blocking pair is a man-woman pair $(m, w)$ where $m$ and $w$ are not matched to each other, $m$ would rather be with $w$ than his partner, and $w$ would rather be with $m$ than her partner. Gale and Shapley's paper introducing stable matchings contains two surprising facts about this problem: regardless of the preference lists, a stable matching

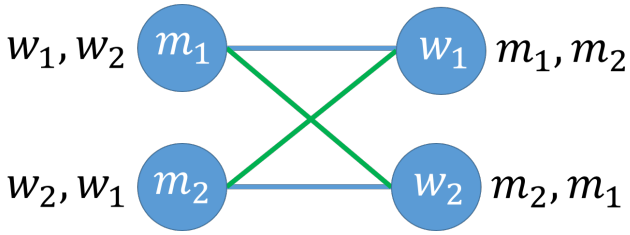



Figure 1.1: A stable matching instance with two possible stable matchings. The blue matching is stable because the men both have their first choices. The green matching is stable because the women both have their first choices.

always exists; and for some sets of preference lists more than one stable matching can exist. See Figure 1.1.

Once one realizes that the number of stable matchings can vary, it is a very natural question to ask how quickly the number of stable matchings can grow. The question has appeared in short lists of open problems on stable matchings created by Knuth [Knu76] and Gusfield and Irving [GI89].

We study the asymptotics of $f(n)$, the maximum number of stable matchings in an instance with $n$ agents on each side of the matching. For intuition, we recall the well-known fact that $f(n) \geq 2^{n/2}$. The instance consists of $n/2$ independent copies of the instance in Figure 1.1. That is, men $2i+1, 2i+2$ and women $2i+1, 2i+2$ have each other as their top choices (where the rankings in Figure 1.1 correspond to $i = 1$), and arbitrary lists thereafter. Observe that for each $i$, we can choose either the green matching or the blue matching, and there are no blocking pairs between the sub-instances. Thus we have $2^{n/2}$ matchings as claimed.

### *1.1.1 Tools*

Our main tool in this chapter is the "rotation poset," an object first described by Irving and Leather [IL86]. The rotation poset provides a succinct description of the set of stable

matchings. One can describe a set of polynomially-many "rotations" (intuitively, short descriptions of exchanges of partners that make one side slightly better-off at the expense of the other). It turns out these rotations are partially ordered, giving extra structure to the set.

### *1.1.2 Our Contribution*

We introduce the notion of a "mixing poset." It turns out that rotation posets can be decomposed into a small set of paths, such that these paths must intersect each other frequently. The notion of "mixing" is novel, though the proof that rotation posets have the mixing property is simply a combination of well-established observations about those posets.

It turns out that mixing posets always have an element that is somewhat "central" (that is one which both dominates and is dominated-by a large fraction of the elements). Using basic combinatorics, we can guarantee the existence of this central element. That central element can be used to build a recurrence, and thus a new bound on the number of stable matchings.

Specifically, we show in Theorem 2.1.1 that $f(n) \leq c^n$ for some constant $c$. Since there are lower bounds of the form $f(n) \geq c'^n$ for other constants $c'$, we now know that $f(n)$ grows as $2^{\Theta(n)}$.

## *1.2 Online Matching*

The Min-Cost Perfect Matching with Delays Problem has become a very popular subject of study in the online algorithms community. Inspired by ridesharing platforms, like Lyft and Uber, the problem asks the algorithm designer to pair requests appearing over time in a metric space. An algorithm matches requests irrevocably (i.e. once requests are paired off, they are permanently matched). The quality of a matching is the sum (over all matched pairs) of the distance between the paired requests and the time all requests had to wait to be matched.

Algorithms are evaluated using competitive analysis. The "competitive ratio" of an

algorithm is the worst possible ratio between the cost of the algorithm, and the cost of the best possible matching (i.e., one made with total foresight). The best-known competitive ratio for MPMD algorithms is $\mathcal{O}(\log n)$, shown by Azar, Chiplunkar, and Kaplan [ACK17].

We also discuss a stochastic version of the problem, where requests are drawn from a probability distribution (rather than being chosen by an adversary).

### *1.2.1 Tools*

There are two main threads in prior work on MPMD: one studying randomized algorithms, the other deterministic ones. Randomized algorithms frequently make use of "hierarchically separated trees (HSTs)." These trees are methods of approximating arbitrary discrete metric spaces (like those where our requests appear) with tree metrics. Embedding into HSTs alters the metric, which may increase error. However, the tree metric is much more structured than an arbitrary metric space. This simpler space lets us design better algorithms, compensating for the distortion between the original metric space and the one we optimize for.

Critical to the HST approach is that creating the HST is a random process, where the adversary cannot see the exact result.[1]

### *1.2.2 Our Contributions*

We make two contributions. The first is to discuss randomized MPMD algorithms through the lens of "ball-growing." Our new algorithms, and multiple prior algorithms in the literature can be viewed through this lens, and we emphasize this common thread. We think of every request steadily growing a "ball" around it (encompassing more of the metric space). Matching rules can then be phrased in terms of balls intersecting. This perspective naturally shows the goals of the algorithms – balancing the distance between points and the waiting time they are forced to incur. Our "odd-ball-growing" algorithm matches the best-known

[1] When embedding into the HST, the distances between some pairs are "distorted" more than others. If the adversary knew which pairs were most distorted, they could exploit the algorithm treating those pairs as more distant than they actually are.

behavior of existing algorithms for MPMD[2]. Compare Theorem 3.3.3 and Theorem 3.7.1.

We also show the versatility of the "ball-growing" perspective, by analyzing it in the stochastic case. We introduce a simple stochastic setting for MPMD, where the requests are drawn from probability distributions instead of chosen by an adversary. In this case a simplified version of the ball-growing approach achieves excellent performance: the expectation of our algorithm's cost differs from the optimal by only a constant factor (see Theorem 3.6.4).

### *1.3 Tournament Design*

Chapter 4 discusses a real-world use of creating pairings – designing (sports) tournaments. Our work is inspired primarily by the 2012 Olympic Badminton tournament. In that tournament, a quirk of the design (and an upset in an early round) produced a match where both teams were better-off losing the match they were playing than winning it.[3] The incident generated significant discussion as to whether the attempt to lose was unethical (see references in section 4.1). Regardless, it became clear that the tournament needed to be redesigned so as to not incentivize competitors to lose. It is not hard to design such a tournament, a traditional "single-elimination bracket" (pair teams to play a match, eliminate losers, recurse) would suffice.

The design is simple, and known to the designers of the Olympic tournament[4], yet by the following Olympics, they had instead tweaked the two-phase design they previously used. The decision to maintain the two phases shows they were trying to maintain other features of the tournament (besides attempting to restore teams' desire to win). As an example, the 2012 design ensures every team will play at least three matches, which is not matched by the single-elimination tournament.

---

[2] up to constant factors

[3] "Better-off" in the sense of increasing their chances of becoming champion.

[4] their 2012 tournament contains two phases; phase two is a single-elimination bracket

### *1.3.1 Tools*

While prior work on tournament design has occasionally found use for heavy machinery, the majority of work on these problems use only fundamental probability theory and combinatorics (though often with clever arguments). Our arguments end up requiring only these fundamentals.

### *1.3.2 Our Contribution*

We design a tournament which is provably "monotone" (that is, ensuring no team ever wants to lose a match), while also ensuring that every team is allowed to play two meaningful matches. Our design is closely based on one used in real-world tournaments, but it gives surprising insight into the monotonicity of tournaments. Prior to our tournament, the only proofs of monotonicity have followed straight from the definition of the tournament. While our proof is not difficult, it is the only non-trivial one we know of. Second, we show that common-in-practice modifications of our tournament (even naively increasing the number of participants) will break monotonicity, thus showing significant care must be taken in using these tournaments in practice.

# Chapter 2

# A SIMPLY EXPONENTIAL UPPER BOUND ON THE MAXIMUM NUMBER OF STABLE MATCHINGS

## *2.1 Introduction*

Stable matching is a classical combinatorial problem that has been the subject of intense theoretical and empirical study since its introduction in a seminal paper by Gale and Shapley in 1962 [GS62]. Variants of the algorithm introduced in [GS62] are widely used in practice, for example to match medical residents to hospitals. Stable matching is even the focus of the 2012 Nobel Prize in Economics [Ram12].

A stable matching instance with $n$ men and $n$ women is defined by a set of preference lists, one per person. Person $i$'s preference list gives a ranking over the members of the opposite sex. The *Stable Matching Problem* is to find a matching (i.e., a bijection) between the men and the women that is *stable*, that is, has no *blocking pairs*. A man $m$ and a woman $w$ form a blocking pair in a matching if they are not matched to each other, but both prefer the other to their partner in the matching. Gale and Shapley [GS62] showed that a stable matching always exists and gave an efficient algorithm to find one.[1] Since at least one stable matching always exists, a natural question is to determine the maximum number of stable matchings an instance of a given size can have. This problem was posed in the 1970s in a monograph by Knuth [Knu76], and was the first of Gusfield and Irving's twelve open problems in their 1989 textbook [GI89]. We denote the maximum number of stable matchings an instance with $n$ men and $n$ women can have by $f(n)$.

Progress on determining the asymptotics of $f(n)$ has been somewhat slow. The best lower

[1]Stable matching algorithms were actually developed and used as early as 1951 to match interns to hospitals [Sta53].

bound is approximately $2.28^n$, and the best upper bound prior to this paper was $2^{n \log n - \mathcal{O}(n)}$. See the related work section for a detailed history.

In this chapter, we present an improved upper bound.

**Theorem 2.1.1.** *There is a universal constant c such that $f(n)$, the number of stable matchings in an instance with n men and n women, is at most $c^n$.*

To prove this theorem, we use a result of Irving and Leather [IL86] that shows that there is a bijection between the stable matchings of an instance $\mathcal{I}$ and the downsets[2] of a particular partially-ordered set (poset) associated with $\mathcal{I}$ known as the *rotation poset*. We show that the rotation poset associated with a stable matching instance has a particular property that we call *n-mixing*, and that any poset with this property has at most $c^n$ downsets. All the steps in our proof are elementary.

The bound extends trivially to stable roommates instances. In the stable roommates problem, a set of $n$ agents rank the other $n-1$ agents in the set. The agents are paired off into roommate pairs, which are stable if no two agents would like to leave their partners and be matched to each other. A construction of Dean and Munshi [DM10], demonstrates that a stable roommates instance with $n$ agents can be converted into a stable matching instance with $n$ men and $n$ women, such that the stable roommate assignments correspond to a subset of the stable matchings in the new instance. Using this construction, we can apply our upper bound to Stable Roommates.

**Theorem 2.1.2.** *There is a universal constant c, such that the number of stable assignments in a stable roommate instance with n agents is at most $c^n$.*

---

[2]See section 2.2 for definitions of all the relevant terminology.

### *2.1.1 Related Work*

*Lower Bounds*

It is trivial to provide instances with $2^{n/2}$ stable matchings by combining disjoint instances of size 2 (see section 1.1). Irving and Leather constructed a family of instances [IL86] which has since been shown by Knuth[3] to contain at least $\Omega(2.28^n)$ matchings. Irving and Leather's family only has instances for $n$ which is a power of 2. Benjamin, Converse, and Krieger also provided a lower bound on $f(n)$ by creating a family of instances with $\Omega(2^n\sqrt{n})$ matchings [BCK95]. While this is fewer matchings than the instances in [IL86], Benjamin et al.'s family has instances for every even $n$, not just powers of 2. In 2002, Thurber extended Irving and Leather's lower bound to all values of $n$. For $n$ powers of 2, Thurber's construction exactly coincides with Irving and Leather's. For all other $n$, the construction produces a lower bound of $2.28^n/c^{\log n}$ for some constant $c$ [Thu02]. To date, this lower bound of $\Omega(2.28^n)$ is the best known. We refer the reader to Manlove's textbook for a more thorough description of the history of these lower bounds [Man13].

*Upper Bounds*

Trivially, there are at most $n!$ stable matchings (as there are at most $n!$ bijections between the men and women). The first progress on upper bounds that we are aware of was made by Stathoupolos in his 2011 Master's thesis [Sta11], where he proves that the number of stable matchings is at most $\mathcal{O}(n!/c^n)$ for some constant $c$. A more recent paper of Drgas-Burchardt and Świtalski shows a weaker upper bound of approximately $\frac{3}{4}n!$ [DBŚ13]. All previous upper bounds have the form $2^{n\log n-\mathcal{O}(n)}$.

*Restricted Preferences*

The number of possible stable matchings has also been studied under various models restricting or randomizing the allowable preference lists. If all preference lists are equally

[3]Personal communication, as described in [GI89].

likely and selected independently for each agent, Pittel shows that the expected number of stable matchings is $\mathcal{O}(n \log n)$ [Pit89]. Applying Markov's Inequality shows that the number of stable matchings is polynomial in $n$ with probability $1 - o(1)$. Therefore, the lower bound instances described above are a vanishingly small fraction of all instances. Work of Hoffman, Levy, and Mossel (described in Levy's PhD thesis [Lev17]) shows that under a Mallows model [Mal57], where preference lists are selected with probability proportional to the number of inversions in the list, the number of stable matchings is $C^n$ with high probability (where the constant $C$ depends on the exact parameters of the model).

The number of attainable partners[4] a person can have has also been the subject of much research. Knuth, Motwani, and Pittel show that the number of attainable partners is $\mathcal{O}(\log n)$ with high probability if the lists are uniformly random [KMP90]. Immorlica and Mahdian show that if agents on one side of the instance have random preference lists of length $k$ (and consider all other agents unacceptable) the expected number of agents with more than one attainable partner depends only on $k$ (and not on $n$) [IM05]. Ashlagi, Kanoria, and Leshno show that if the number of men and women is unbalanced, with uniformly random lists, the fraction of agents with more than one attainable partner goes to 0 as the size of the market grows [AKL17]. Intuitively, the advantage to being on the proposing side "disappears" when the market becomes unbalanced (and sufficiently large). Follow-up work of Pittel finds the expected number of stable matchings in this setting and related results [Pit17a].

Various related questions have been examined for alternative notions of stability and in the stable roommates setting. See [Pit17b, Pit17c] and references therein.

*Counting*

A natural computational problem is to count the number of stable matchings in a given instance as efficiently as possible. Irving and Leather show that finding the exact number of matchings is $\#P$-complete [IL86], so finding an approximate count is a more realistic goal.

[4] Woman $w$ is an attainable partner of man $m$ if there is a stable matching in which they are matched to each other.

Bhatnagar, Greenberg, and Randall consider instances where preference lists come from restricted models [BGR08]; for example, those in which the preference lists reflect linear combinations of $k$ "attributes" of the other set, or where every agent appears in a "range" of $k$ consecutive positions in every preference list. In both of these cases, they show that a natural Markov Chain Monte Carlo approach does not produce a good approximation (as the chain does not mix efficiently). As part of their proof, they show the number of stable matchings can still be as large as $c^n$ for some constant $c < 2.28$, even in these restricted cases.

A formal hardness result was later shown by Dyer, Goldberg, Greenhill, and Jerrum [DGGJ04]. They show approximately counting the number of stable matchings is equivalent to approximately counting for a class of problems, canonically represented by #BIS.[5] This hardness result was strengthened by Chebolu, Goldberg, and Martin [CGM12] to hold even if the instances come from some of the restricted classes of [BGR08].

The heart of all of these results is the rotation poset (originally developed in [IL86]), which we use and describe in section 2.4.

*Stable Matching in General*

We refer the reader to books by Roth and Sotomayor [RS92], Gusfield and Irving [GI89], Manlove [Man13], and Knuth [Knu97] for more about the topic of stable matching. For many examples of stable matching in the real world, see [Rot15].

## *2.2 Preliminaries and main technical theorem*

In this section, we review standard terminology regarding partially ordered sets, describe the key property of a poset we will use, and state our main technical theorem (Theorem 2.2.5).

[5] More specifically, they show an FPRAS for the number of stable matchings exists if and only if one exists for #BIS, approximately counting the number of independent sets in a bipartite graph. Goldberg and Jerrum conjecture that no such FPRAS exists [GJ12]. See [CGM12] for formal definitions.

**Definition 2.2.1** (Poset)**.** *A partially ordered set (or poset)* $(V, \prec)$*, is defined by a set* $V$ *and a binary relation,* $\prec$*, on* $V$ *satisfying:*

- **Antisymmetry:** *For all distinct* $u, v \in V$ *if* $u \prec v$ *then* $v \not\prec u$*, and*
- **Transitivity:** *for all* $u, v, w \in V$ *if* $u \prec v$ *and* $v \prec w$ *then* $u \prec w$*.*

Two elements $u, v \in V$ are *comparable* if $u \prec v$ or $v \prec u$. They are *incomparable* otherwise. If $u \prec v$, we say $u$ is *dominated* by $v$ and $v$ *dominates* $u$.

**Definition 2.2.2** (Chain)**.** *A set* $S$ *of elements is called a* chain *if each pair of elements in* $S$ *is comparable. In other words, for* $\ell > 0$*, a chain of* length $\ell$ *is a sequence of elements* $v_1 \prec v_2 \prec v_3 \prec \cdots \prec v_\ell$*.*

A set of elements is called an *antichain* if they are pairwise incomparable.

**Definition 2.2.3** (Downset)**.** *A* downset *of a partial order is an antichain and all elements dominated by some element of that antichain.*

Observe that a downset is closed under $\prec$. That is, for any downset $S$, if $v \in S$ and $u \prec v$ then $u \in S$.

The following is the key property of the posets associated with stable matching instances that we will use in the proof.

**Definition 2.2.4** ($n$-mixing)**.** *A poset* $(V, \prec)$ *is* $n$*-mixing if there exist* $n$ *chains* $C_1, \ldots, C_n$ *(not necessarily disjoint) such that*

*i) Every element of* $V$ *belongs to at least one of the chains,*
*i.e.,* $\cup_{i=1}^{n} C_i = V$*,*

*ii) For any* $U \subseteq V$*, at least* $2\sqrt{|U|}$ *of the* $C_i$ *contain an element of* $U$*.*

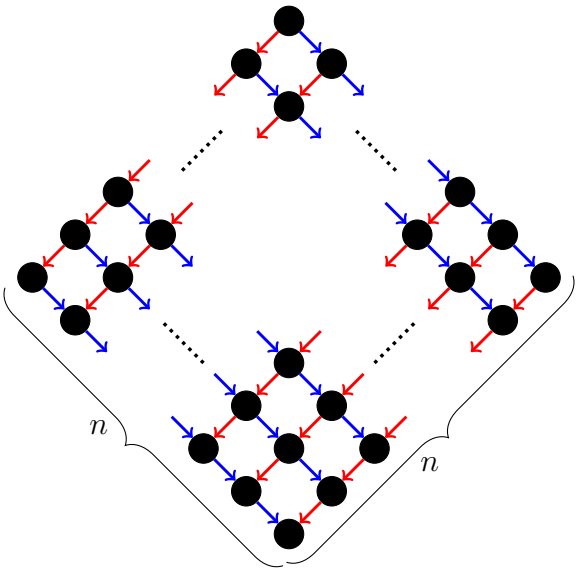



Figure 2.1: A $2n$-mixing poset with respect to chains defined by the red and blue paths. A path in the graph from $v$ to $u$ indicates that $u \prec v$. That is, the poset is the transitive closure of the arrows shown. For any set $U$ of $k$ elements, there are at least $2\sqrt{k}$ chains that contain one of these elements.

Observe that if a poset is formed by $n$ disjoint chains, each of length $\ell$, it has about $\ell^n$ downsets, and $\ell$ could be arbitrarily bigger than $n$. But such a poset is not mixing, since taking $U$ to be the set of elements on one of the chains violates property ii) of mixing. For an example of a mixing poset, see Figure 2.1. We can now state our main technical theorem.

**Theorem 2.2.5.** *There is a universal constant $c$, such that if a poset is $n$-mixing, then it has at most $c^n$ downsets.*

Note that the $n$-mixing property immediately implies that the poset has at most $n^2$ elements; just let $U = V$ in the above definition. A poset with $n^2$ elements covered by $n$ chains can have at most $(n+1)^n$ downsets (this is achieved for $n$ equal length chains). So, the main contribution of the above theorem is to improve this trivial upper bound to $c^n$ for some constant $c$.

Theorem 2.2.5 is the main technical contribution of this work. The proof is contained in section 2.3. To complete the proof of Theorem 2.1.1 we use the following theorem relating mixing posets to stable matchings.

**Theorem 2.2.6.** *For every stable matching instance $\mathcal{I}$ with a total of $n$ men and women, there exists an $n$-mixing poset $(\mathcal{R}, \prec)$, called the rotation poset, such that the number of downsets of the poset is equal to the number of stable matchings of $\mathcal{I}$.*

Note that Theorem 2.2.5 and Theorem 2.2.6 immediately imply Theorem 2.1.1. We prove Theorem 2.2.6 in section 2.4, by combining existing observations about the rotation poset.

### *2.3 Proof of main technical theorem*

In this section we prove Theorem 2.2.5. The proof proceeds by finding an element $v$ of the poset which dominates and is dominated by many elements. We then count downsets by considering those downsets that contain $v$ and those that do not. Since $v$ dominates and is dominated by many elements, the size of each remaining instance is significantly smaller, yielding the bound.

Formally, we say an element is $\alpha$-*critical* if it dominates $\alpha$ elements and is dominated by $\alpha$ elements. The key lemma is that there is always an $\Omega((|V|/n)^{3/2})$-critical element.

**Lemma 2.3.1.** *Let $(V, \prec)$ be an $n$-mixing poset with respect to chains $C_1, \ldots, C_n$, and define $d = \frac{|V|}{n}$. For some universal constants $d_0 > 1$ and $c_0 > 0$, there is an element $v \in V$ such that $v$ is $(c_0 d^{3/2})$-critical as long as $d \geq d_0$.*

We prove Lemma 2.3.1 in subsection 2.3.1 via a counting argument.

In the rest of this section we prove Theorem 2.2.5 using Lemma 2.3.1. We bound the number of downsets by induction on $d$.

Our base case is when $d = d_0$. In this case, the number of downsets is maximized when the chains are all the same length, so we have an upper bound of $(d_0 + 1)^n$.

For larger $d$, first we identify a $\left(c_0 d^{3/2}\right)$-critical element $v$. The number of downsets containing $v$ is equal to the number of downsets in the poset remaining after we delete $v$ and everything it dominates. Similarly, the number of downsets not containing $v$ is the number of downsets in the poset remaining after we delete $v$ and everything dominating $v$. In both cases, the resulting poset is still $n$-mixing[6], so we can induct. We call such a step (choosing a critical element) an *iteration*.

It remains to bound the number of downsets that this process enumerates. By the $n$-mixing property, there are at most $n^2$ elements in the initial poset. We partition the iterations into phases, where in phase $i$ we reduce the size of the poset from $\frac{n^2}{2^i}$ elements to $\frac{n^2}{2^{i+1}}$ elements. By definition, in phase $i$, $d \geq \frac{n}{2^{i+1}}$. So, we can bound the number of iterations required in phase $i$ (call it $k_i$) by:

$$c_0 \left(\frac{n}{2^{i+1}}\right)^{3/2} k_i > \frac{n^2}{2^{i+1}}.$$

Rearranging, we see it suffices to choose $k_i = 2^{(i+1)/2}\sqrt{n}/c_0$. We continue until $d = d_0$. Summing across all phases, the number of choices to make is at most

$$\frac{\sqrt{n}}{c_0} \sum_{i=0}^{\log n} 2^{(i+1)/2} < 5\frac{n}{c_0}.$$

[6]with respect to the (what remain of) the same chains

So, the algorithm enumerates at most $(d_0 + 1)^n$ downsets in the base case and it makes at most $5n/c_0$ choices during the inductive process. Thus, the number of downsets is at most $(d_0 + 1)^n 2^{5n/c_0} = c^n$ as required.

### *2.3.1 Proof of Main Technical Lemma*

Finally, we prove Lemma 2.3.1. That is, we show that any $n$-mixing poset $(V, \prec)$ contains a $c_0 d^{3/2}$-critical element as long as $d \geq d_0$. (Recall that $d = |V|/n$.)

We make use of the standard graph representation of the partial order: In this graph there is a node for each element of the partial order, and a directed edge from $v$ to $u$ if $u \prec v$. Of course, this directed graph is acyclic. Henceforth, we refer only to this DAG rather than to the poset. We partition the nodes of the DAG into *levels* as follows: Level 1 nodes are those with no outgoing edges, i.e., sinks of the DAG, and level $i$ nodes are those whose longest path to a sink (i.e., a level 1 node) has exactly $i$ nodes. Note that each level is an antichain.

Next, we create $n$ disjoint subchains $S_1, \ldots, S_n$. The subchain $S_i$ will be a subset of the nodes in $C_i$. We perform the assignment of nodes to subchains by processing up the DAG level by level. Initialize every subchain $S_i$ to be empty. For each node $u$, consider the set of indices $I(u) = \{j : u \in C_j\}$. Assign $u$ to the subchain for an index in $I(u)$ which currently has the fewest nodes among those chains, i.e. $\arg\min_{j \in I(u)} |S_j|$, breaking ties arbitrarily. If $u$ is the $k^{\text{th}}$ node assigned to a subchain $S_i$, then we say the *height* of $u$ is $k$. By construction, the following properties hold:

(a) The $S_i$'s are chains, since $S_i \subseteq C_i$.

(b) The $S_i$'s are disjoint.

(c) If $u$ has height $h$, then $u$ dominates at least $h - 1$ nodes in each of the subchains $S_j$ such that $j \in I(u)$ (i.e., $\{j : u \in C_j\}$).

**Claim 2.3.2.** *Let $\overline{D}$ be the set of nodes of height at least $\lfloor d/2 \rfloor$. Each node $u \in \overline{D}$ dominates at least $c_0 \cdot d^{3/2}$ nodes, for $d \geq d_0$, where $c_0$ and $d_0$ are universal constants.*

*Proof.* Suppose that $u \in S_i \cap \overline{D}$ is at height $\ell \geq \lfloor d/2 \rfloor$, and let $D(u)$ be the nodes in $S_i$ of height $\lceil \ell/2 \rceil$ through $\ell$. By the mixing property, these nodes lie on at least $2\sqrt{\lfloor \ell/2 \rfloor}$ chains $\mathcal{C}$. Moreover, by construction, on each subchain $S_j \in \mathcal{C}$, at least $\lfloor \ell/2 \rfloor - 1$ nodes are dominated by some node in $D(u)$ and hence are dominated by $u$. Therefore, $u$ dominates at least $(\lfloor \ell/2 \rfloor - 1) \cdot 2\sqrt{\lfloor \ell/2 \rfloor} + (\lfloor \ell/2 \rfloor - 1) = \Omega(d^{3/2})$ nodes. □

Since the number of nodes in the DAG is $dn$, we conclude:

**Corollary 2.3.3.** *There is a set $\overline{D}$ of strictly more than $|V|/2$ nodes, that each dominate $\Omega(d^{3/2})$ nodes.*

A symmetric argument in which subchains are built starting from the sources of the DAG shows that there is a set $\underline{D}$ of strictly more than $|V|/2$ nodes that are dominated by $\Omega(d^{3/2})$ nodes. Therefore, there is some node $v$ in the intersection of $\overline{D}$ and $\underline{D}$. This is the $c_0 d^{3/2}$-critical node we seek.

**Remark 2.3.4.** *A crude analysis shows that $c_0 \geq 1/8$ when $d_0 > 25$.*

### *2.4 Rotations and the rotation poset*

In this section we present a key theorem of Irving and Leather [IL86], and use it to prove Theorem 2.2.6.

**Theorem 2.2.6.** *For every stable matching instance $\mathcal{I}$ with a total of $n$ men and women, there exists an $n$-mixing poset $(\mathcal{R}, \prec)$, called the rotation poset, such that the number of downsets of the poset is equal to the number of stable matchings of $\mathcal{I}$.*

We begin with the definitions needed to prove Theorem 2.2.6.

**Definition 2.4.1** (Rotation)**.** *Let $k \geq 2$. A* rotation $\rho$ *is an ordered list of pairs*

$$\rho = ((m_0, w_0), (m_1, w_1), \ldots, (m_{k-1}, w_{k-1}))$$

*that are matched in some stable matching $M$ with the property that for every $i$ such that $0 \leq i \leq k-1$, woman $w_{i+1}$ (where the subscript is taken mod $k$) is the highest ranked woman on $m_i$'s preference list satisfying:*

*i) man $m_i$ prefers $w_i$ to $w_{i+1}$, and*

*ii) woman $w_{i+1}$ prefers $m_i$ to $m_{i+1}$.*

*In this case, we say $\rho$ is* exposed *in $M$.*

We will sometimes abuse notation and think of a rotation as the set containing those pairs. Also, we will need the following facts about rotations later.

**Lemma 2.4.2** ([IL86, Lemma 4.7])**.** *A pair $(m, w)$ can appear in at most one rotation.*

**Lemma 2.4.3** ([GI89, Lemma 2.5.1])**.** *If $\rho$ is a rotation with consecutive pairs $(m_i, w_i)$, and $(m_{i+1}, w_{i+1})$, and $w$ is a woman between $w_i$ and $w_{i+1}$ in $m_i$'s preference list, then there is no stable matching containing the pair $(m_i, w)$.*

**Definition 2.4.4** (Elimination of a Rotation)**.** *Let $\rho = ((m_0, w_0), \ldots, (m_{k-1}, w_{k-1}))$ be a rotation exposed in stable matching $M$. The rotation $\rho$ is* eliminated *from $M$ by matching $m_i$ to $w_{(i+1) \bmod k}$, for all $0 \leq i \leq k-1$, leaving all other pairs in $M$ unchanged, i.e., matching $M$ is replaced with matching $M'$, where*

$$M' := M \backslash \rho \cup \{(m_0, w_1), (m_1, w_2), \ldots, (m_{k-1}, w_0)\}.$$

*Note that when we eliminate a rotation from $M$, the resulting matching $M'$ is stable.*[7]

[7]Switching from $M$ to $M'$ makes all the women in $\rho$ happier and all the men in $\rho$ less happy. It is easy to check that this switch cannot create a blocking pair inside the set $\rho$. The only other possibility for a blocking pair is a man in $\rho$ with a woman outside $\rho$. For $(m_i \in \rho, w \notin \rho)$ to become a blocking pair, $m_i$ would have to prefer $w$ to $w_{i+1}$, but by the definition of rotation, $w_{i+1}$ was the first woman on $m$'s list who would prefer to be matched to him, so he cannot prefer $w$ to $w_{i+1}$. See also [GI89, Lemma 2.5.2].

Irving and Leather studied the following process: Fix a stable matching instance $\mathcal{I}$. Starting at the man-optimal matching[8], choose a rotation exposed in the current matching, and eliminate it. They show that for any stable matching $M$, there is a set of rotations, $R(M)$, one can eliminate (starting from the man-optimal matching) that will yield $M$.

However, there is a partial order on the set of rotations – some must be eliminated before others.[9] If $\rho$ must be eliminated before $\rho'$, we write $\rho \prec \rho'$. Let $\mathcal{R}$ be the set of rotations for a stable matching instance, and let $\prec$ be that partial order on the rotations defined by elimination order. We call this poset the *rotation poset.* See Figure 2.2 for an example of a stable matching instance and the corresponding rotation poset.

For our purposes, the important result relating the rotation poset to stable matchings is the following.

**Theorem 2.4.5** ([IL86, Theorem 4.1])**.** *For any stable matching instance, there is a one-to-one correspondence between downsets of the rotation poset and the set of stable matchings. In other words, the number of stable matchings is exactly equal to the number of downsets of* $(\mathcal{R}, \prec)$.

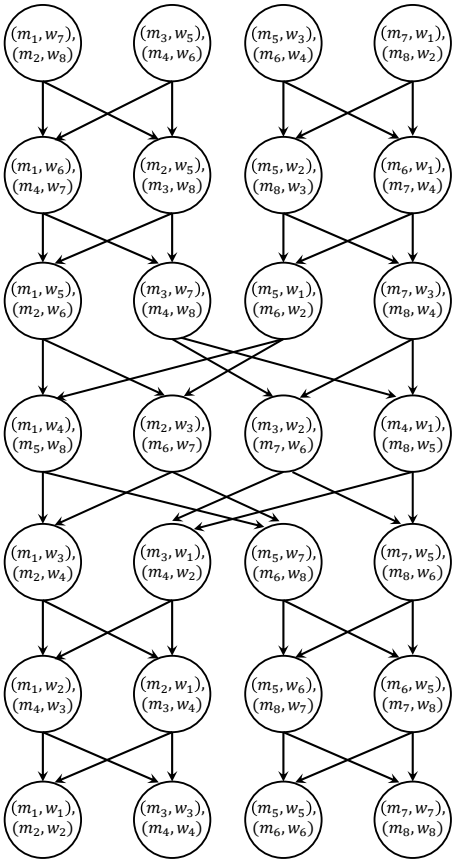
Indeed, the downset corresponding to $M$ is exactly the set $R(M)$ discussed above.

Thus, to prove Theorem 2.2.6, it remains to show that the rotation poset associated to any stable matching instance with a total of $n$ men and women is $n$-mixing. First we construct the chains $C_1, \ldots, C_n$. We will have one chain for each agent (man or woman), where the corresponding chain contains all the rotations that include that agent. Call these sets $C_1, \ldots, C_n$. To prove that $C_1, \ldots, C_n$ is $n$-mixing, we first need to show that each $C_i$ is

---

[8] A fascinating fact about stable matching is that there is a matching, known as the man-optimal matching, in which each man is matched with his favorite attainable partner. Recall that woman $w$ is attainable for man $m$ if there is some stable matching in which they are matched.

[9] For example, a rotation containing a pair $(m, w)$ is not exposed (and thus cannot be eliminated) until that pair is matched, so a rotation with consecutive pairs $(m, w'), (m', w)$ must be eliminated first. The details of exactly when one rotation must be eliminated before another are not of direct use to us (we only require the rather coarse description in Claim 2.4.6), so we do not describe them here. See [IL86] or [GI89] for a full description of the poset.




Men's preferences:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $m_1$ | $w_1$ | $w_2$ | $w_3$ | $w_4$ | $w_5$ | $w_6$ | $w_7$ | $w_8$ |
| $m_2$ | $w_2$ | $w_1$ | $w_4$ | $w_3$ | $w_6$ | $w_5$ | $w_8$ | $w_7$ |
| $m_3$ | $w_3$ | $w_4$ | $w_1$ | $w_2$ | $w_7$ | $w_8$ | $w_5$ | $w_6$ |
| $m_4$ | $w_4$ | $w_3$ | $w_2$ | $w_1$ | $w_8$ | $w_7$ | $w_6$ | $w_5$ |
| $m_5$ | $w_5$ | $w_6$ | $w_7$ | $w_8$ | $w_1$ | $w_2$ | $w_3$ | $w_4$ |
| $m_6$ | $w_6$ | $w_5$ | $w_8$ | $w_7$ | $w_2$ | $w_1$ | $w_4$ | $w_3$ |
| $m_7$ | $w_7$ | $w_8$ | $w_5$ | $w_6$ | $w_3$ | $w_4$ | $w_1$ | $w_2$ |
| $m_8$ | $w_8$ | $w_7$ | $w_6$ | $w_5$ | $w_4$ | $w_3$ | $w_2$ | $w_1$ |

Women's preferences:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $w_1$ | $m_8$ | $m_7$ | $m_6$ | $m_5$ | $m_4$ | $m_3$ | $m_2$ | $m_1$ |
| $w_2$ | $m_7$ | $m_8$ | $m_5$ | $m_6$ | $m_3$ | $m_4$ | $m_1$ | $m_2$ |
| $w_3$ | $m_6$ | $m_5$ | $m_8$ | $m_7$ | $m_2$ | $m_1$ | $m_4$ | $m_3$ |
| $w_4$ | $m_5$ | $m_6$ | $m_7$ | $m_8$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ |
| $w_5$ | $m_4$ | $m_3$ | $m_2$ | $m_1$ | $m_8$ | $m_7$ | $m_6$ | $m_5$ |
| $w_6$ | $m_3$ | $m_4$ | $m_1$ | $m_2$ | $m_7$ | $m_8$ | $m_5$ | $m_6$ |
| $w_7$ | $m_2$ | $m_1$ | $m_4$ | $m_3$ | $m_6$ | $m_5$ | $m_8$ | $m_7$ |
| $w_8$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $m_8$ |

Figure 2.2: A size-8 stable matching instance with its rotation poset. This is part of the family of instances described by Irving and Leather, which produces the $\Omega(2.28^n)$ lower bound.

indeed a chain, i.e., every pair of rotations where a specific agent appears are comparable and second we need to show that $C_1, \ldots, C_n$ satisfy property (ii) of Definition 2.2.4.

**Claim 2.4.6.** *If two rotations share an agent, then they are comparable.*[10]

*Proof.* First, suppose that the shared agent is a man. If it is a woman, we can just switch the designations of "men" and "women" and use the same proof on the "reversed" version of the rotation graph. Let $\rho_1, \rho_2$ be rotations sharing an agent $m$, and let $m$ be matched to $w_1$ in $\rho_1$ and $w_2$ in $\rho_2$, where $m$ prefers $w_1$ to $w_2$.

For the sake of contradiction assume that $\rho_1$ and $\rho_2$ are incomparable. We show that there exists a rotation $\rho$ which causes $m$ to skip over $w_1$. This would contradict Lemma 2.4.3 as it implies that $(m, w_1)$ belongs to no stable matching.

Suppose we start from the man-optimal stable matching and eliminate all rotations dominated by $\rho_2$ and let $M$ be the resulting stable matching. By the correspondence in Theorem 2.4.5, $(m, w_2) \in M$. Since $m$ prefers $w_1$ to $w_2$, there must be a rotation $\rho$ that we eliminated which caused $m$ to be matched to someone worse than $w_1$ for the first time. Since $\rho_2$ and $\rho_1$ are incomparable, $\rho_1 \neq \rho$. Therefore, by Lemma 2.4.2 $(m, w_1) \notin \rho$; so, $\rho$ caused $m$ to skip over $w_1$. This is a contradiction. □

**Claim 2.4.7.** *Every set of $k$ rotations contains at least $2\sqrt{k}$ agents.*

*Proof.* We argue by contrapositive. Suppose we have a set of rotations involving fewer than $2\sqrt{k}$ agents. Every rotation contains a (man, woman) pair, who (by Lemma 2.4.2) have not appeared together before. With fewer than $2\sqrt{k}$ agents, there are strictly less than $k$ (man, woman) pairs which can appear, and thus fewer than $k$ rotations in the set. □

## *2.5 Conclusion*

We have shown there is some constant $c$ such that $f(n) \leq c^n$. We have not made a significant effort to optimize the constants in our argument, favoring ease of exposition over the exact

[10] This observation is not novel; for example, it is implicit in the discussion of [GI89], but we have not seen the statement explicitly written down, so we prove it here.

result. By making a few minor changes to the argument, we obtain $f(n) \leq 2^{17n}$ for sufficiently large $n$.

A more careful argument could probably improve this constant somewhat, but this approach will not get a constant $c$ close to the (approximately) 2.28 we would need to match the best known lower bound. Determining the precise asymptotic behavior of $f(n)$ remains an interesting open problem.

# Chapter 3

# ONLINE MATCHING

## *3.1 Introduction*

In this chapter, we discuss the Min-Cost Perfect Matching with Delays problem (MPMD). In MPMD, a series of requests appear over time in a metric space. Our goal is to pair these requests (online) to minimize the distances between matched pairs plus the time each request waits to be matched. As motivation, consider the scenario of running a gaming server. Each request is a player looking for a partner to play against. The metric space represents the quality of game expected between those two players (which might take into account player ability, connection strength, and other factors). Our minimization, then, has the goal of ensuring players get compatible opponents, but also that they do not get bored waiting to be matched to their opponent.

More formally, fix a metric space $(\mathcal{X}, d)$. A total of $2m$ requests appear at points $x_1, \ldots, x_{2m}$ in the metric space. The requests appear at times $t_1, \ldots, t_{2m}$. At any time $t > t_i, t_j$ our algorithm can decide to match requests $i$ and $j$, at a cost of $(t-t_i)+(t-t_j)+d(x_i, x_j)$. All requests must be matched before the algorithm ends. When $\mathcal{X}$ is a finite metric space, we denote $|\mathcal{X}|$ by $n$.

In the basic version of the problem, the requests (both the times and locations where they appear) are chosen by an adversary who has knowledge of the algorithm being run to create the matching. If the algorithm is randomized, the adversary is oblivious. That is the adversary will know the distributions of any random variables utilized by the algorithm, but cannot see their actual values before choosing the input.

Since we are making matching decisions as requests appear (and cannot undo a pairing once it is made) we use competitive analysis. We wish to minimize the ratio between the

(expected) cost of our algorithm and the cost of the best possible matching (made with knowledge of the exact set of requests).

Since its introduction by Emek, Kutten, and Wattenhofer in 2016 [EKW16], MPMD has been the focus of a flurry of activity in the theory community. As one of our major contributions is to tie together much of the previous work with the "ball growing" metaphor, we discuss the prior work extensively. We divide our study of previous work into three sections: in the first two sections, we study the power of randomization for MPMD algorithms. Section 3.3 will discuss randomized algorithms for MPMD, and section 3.4 will recap deterministic algorithms. There is still a substantial gap between the quality of matches made by randomized and deterministic algorithms for this problem. Then, in section 3.5, we discuss extensions to the basic MPMD problem, including the "bipartite" version of the problem (where requests come in two different types that can only be matched to each other) and the "impatient" version where the penalty for keeping a request unmatched grows as some convex function of the waiting time.

We also outline our new applications of the ball growing approach. In section 3.6 we show that ball growing performs very well for requests drawn randomly. Then, in section 3.7 we show that a ball growing algorithm can match the best known bounds for MPMD. Before moving to the previous work, we fix a consistent set of notation, then we spend section 3.2 developing intuition for MPMD with two example instances which have been helpful for algorithm designers.

### *3.1.1 Notation*

Let $B(x, r) = \{y \in \mathcal{X} : d(x, y) \leq r\}$ be the ball of radius $r$ centered at $x$. We denote the cost incurred by the current algorithm under study by ALG. We will compare to OPT the cost for the optimal (found offline) solution. When it causes no confusion, we will refer to the algorithms themselves as ALG and OPT. We will occasionally need to separate the time cost and distance costs of the algorithms in our analyses. We add a subscript "time" to denote time cost and subscript "dist" to denote distance cost.

For deterministic algorithms, we say ALG is $\alpha$-competitive if for all inputs,

$$\text{ALG} \leq \alpha \cdot \text{OPT}$$

The goal of the algorithm designer is to minimize $\alpha$.

For randomized algorithms, ALG is a random variable, so our goal is to show $\mathbb{E}[\text{ALG}] \leq \alpha \cdot \text{OPT}$ (note that even though ALG is randomized, OPT will not be, as it has full knowledge of the sequence).

Number the requests $1, 2, \ldots, 2m$. We use $r_i$ to denote the $i^{\text{th}}$ request. The request $r_i$ is defined by $x_i$ and $t_i$, respectively the location and time where the request arrives. We say $r_i$ is "pending" at time $t$ if $t > t_i$ and $r_i$ has not yet been matched (i.e. it is available for the algorithm to match it). When it does not cause confusion, we will conflate a request with the point where it is located.

## *3.2 Intuition via Examples*

Two example instances are quite useful in explaining why MPMD is a non-trivial problem. We start by introducing these two examples so that the reader can see immediately why algorithm designers have made certain choices. For both of the examples, we will describe the construction and then briefly explain why the example makes designing MPMD algorithms more difficult.

### *3.2.1 A Bad Example for Greedy Algorithms*

We begin with an example adapted from previous literature. Prior to the invention of MPMD, computer scientists studied offline versions of matching problems. In that literature, Reingold and Tarjan produced an instance to show that a greedy algorithm was not a good choice in their offline setting [RT81].

The instance is constructed recursively. The base case (iteration 0) of the construction is a set of 4 requests on a line, the middle two requests are at distance $1 - \epsilon$, while the outer two requests are distance 1 from the closer of the middle two. To make the $i^{\text{th}}$ iteration

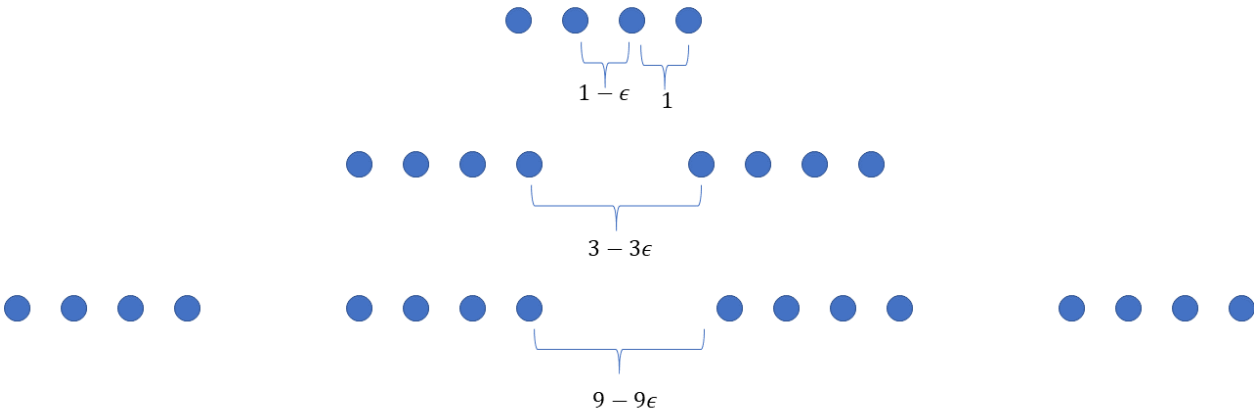



Figure 3.1: Iterations $0, 1,$ and 2 of a construction that causes poor performance for greedy algorithms.

of the construction, we take two copies of the iteration $i-1$ construction, and place them next to each other (on the same line) such that the closest requests from the two copies are distance $3^i(1-\epsilon)$ apart. See Figure 3.1.

The instance is designed to make algorithms that greedily match requests (i.e. by matching the two closest unmatched requests) behave poorly. A greedy algorithm will match as follows. First it matches the middle requests of every copy of an iteration 0 instance. Each iteration 1 instance is now just 4 requests with the middle two slightly closer to each other than any other pairs (and with the level 1 instances well-separated enough to not interfere). This pattern continues, always matching the "middle two" requests in each instance. See Figure 3.2 for a visual representation of a greedy algorithm matching this instance.

It is not hard to see that this is not the optimal choice. Instead of starting by matching the requests at distance $1-\epsilon$, match all pairs at distance 1.

We omit a careful computation of the exact competitive ratio for this algorithm on this instance, and just state the final ratio:

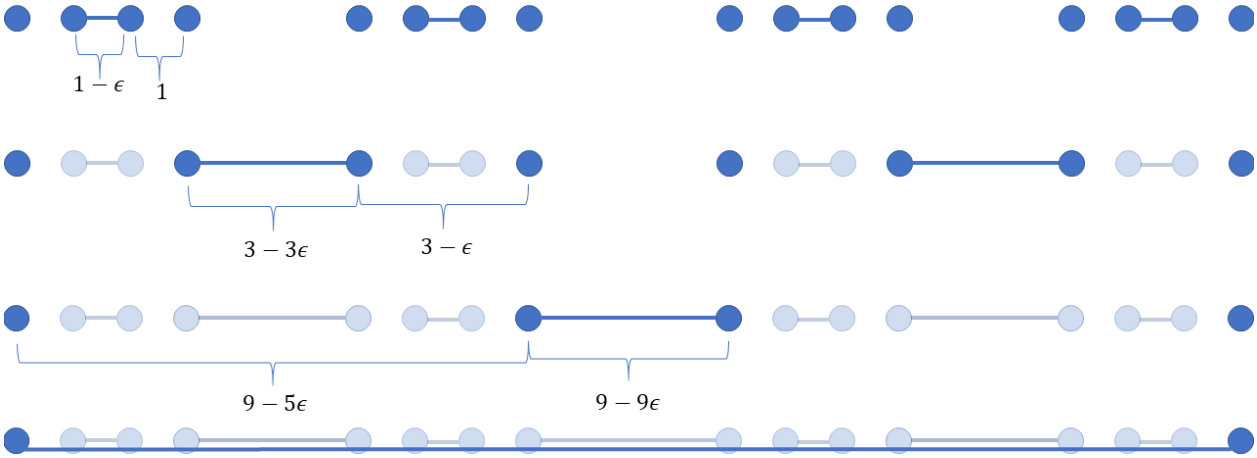



Figure 3.2: The matching choices made by a greedy algorithm on iteration 2 of the construction. The algorithm matches the closest requests first, then matches remaining (unmatched) pairs in order of distance. Semi-transparent requests are already matched.

**Theorem 3.2.1** (Reingold and Tarjan [RT81])**.** *On an instance of Example 1 with $m$ requests, the greedy algorithm incurs a factor $\Omega(m^{\log_2(3/2)}) \approx \Omega(m^{0.584})$ more distance cost than OPT.*

It is worth noting that at this point we have not defined an MPMD instance. So far we have only asked for a perfect matching in a metric space, there is no time component. An adversary could easily utilize this instance by simply causing it to appear (i.e. have all requests arrive simultaneously). In this case, if our algorithm only makes very local/very greedy decisions, we risk being tricked by this instance.

### *3.2.2 A Bad Example for Jumpy Algorithms*

Our next instance shows that sometimes significant patience is required to find good matches; even when requests appear at the same point in the metric space, it still may be wise to wait to match requests. The instance appears (independently) in papers by Azar and Jacob-

Fanani [AJF18] and Liu, Pan, Wang, and Wattenhofer [LPWW18].

Our metric space has only two points, at distance 2 from each other, call the points $x$ and $y$. Requests appear online at both $x$ and $y$ at each of the following times: $0, 1-\epsilon, 1, 2-\epsilon, 2, 3-\epsilon, 3, \ldots$. The optimal matching is the following: pair the first two requests (that appear at $t = 0$) to each other, (at a cost of 2) for all remaining pairs, match the requests that appear at the same locations at times $a - \epsilon$, $a$ for all $a \geq 1$. We incur a matching cost of $(m-1)\epsilon + 2$.

It is fairly natural to imagine matching rules will follow the following principle: "if there are at least two requests at the same point, match them." After all, those requests will incur no distance cost. However, this example makes following such a principle complicated. Imagine an algorithm which will not match the first pair requests at $x$ and $y$ before the next pair of requests appear. By following the principle, both matches incur $1-\epsilon$ time cost, whereas had the new requests been ignored for just $\epsilon$ time, the new pairs could be matched at a (time) cost of $\epsilon$ each. If this behavior is repeated, the jumpy algorithm will incur a cost of $m(1-\epsilon)$. See Figure 3.3. Observe that for $\epsilon = \Theta\left(\frac{1}{m}\right)$, we get a competitive ratio of $\Omega(m)$.

We will not try to formalize the purpose of this example into an actual theorem. We will see two ways to avoid this example in this chapter. The first is to allow for the algorithm to remember some state. The key to this example is repeating the "surprise appearance" of another pair $\epsilon$ time after an appearance. If an algorithm can "remember it was tricked" and alter its behavior it can sidestep this example. The second way to avoid the issue is to throw out the natural principle and allow requests at the same location to go unmatched (until, for example, both have waited for similar amounts of time).

### *3.3 Randomized MPMD algorithms*

We now turn to a discussion of algorithms for MPMD. We start with a discussion of randomized algorithms. We give a brief description of the various algorithms that have been used, as well as the key observations required to understand the algorithms and prove their correctness. For this section, we assume the underlying metric space $(\mathcal{X}, d)$ is finite.

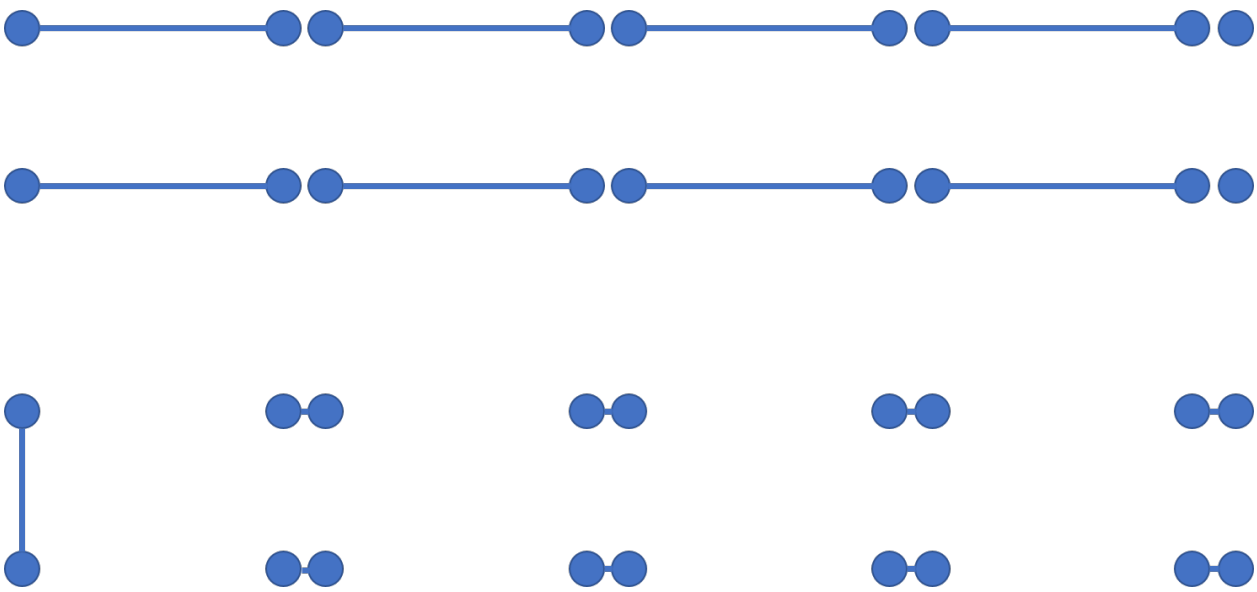


Figure 3.3: On the top, a matching made by a "jumpy" algorithm, below is the optimal matching for this instance. The two points of the metric space are shown vertically, with time progressing horizontally.

The first paper to discuss randomized algorithms for MPMD was Emek et al.'s introductory paper on MPMD [EKW16]. Their algorithm proceeds in two steps, the first of which is to embed into a binary HST.

Hierarchically Separated Trees are a common tool in algorithm design. We do not cover the details here (see, e.g. [WS11] for a sample construction). Multiple papers on MPMD have used HSTs, each needing a slightly different flavor, so we do not state the details each time. As a sample, here is the result we need in our work:

Let $\mathcal{D}$ be a distribution over trees, such that the leaves of every tree are in bijection with $\mathcal{X}$. We say we have embedded $(\mathcal{X}, d)$ into $\mathcal{D}$ if:

1. for all $T$ in the support of $\mathcal{D}$, $d'_T(u,v) \geq d(u,v)$ for all $u, v \in \mathcal{X}$

2. $\mathbb{E}_{T\sim\mathcal{D}}[d'_T(u,v)] \leq \mathcal{O}(\log n)d(u,v)$ for all $u, v \in \mathcal{X}$.

where $d'_T(\cdot,\cdot)$ is the natural distance metric in $T$.

**Theorem 3.3.1.** *There is a polynomial time algorithm to choose a tree according to $\mathcal{D}$.*

The first construction of HSTs with $\mathcal{O}(\log n)$ distortion is [FRT04]. HSTs are a useful tool as trees are a much simpler space to design algorithms. Since the adversary is oblivious to our random choices, the adversary cannot know which pairs $u, v$ ended up far from each other in the tree metric. It knows only that in expectation each pair is stretched by at most $\mathcal{O}(\log n)$.

With the tree in hand, we can run the algorithm. Whenever a request appears in the "real" metric space, we think of a request as appearing in the corresponding leaf of the HST. Now on a binary tree, the algorithm proceeds as follows. If two requests are pending at the same leaf, match them. Now, for every vertex $v$, such that both of its children have an odd number of requests pending in their subtrees: proceed down the tree (from both children), always going to the odd child, until reaching a leaf. We will find two pending requests this way (call the corresponding leaves $x$ and $y$). Now at $v$, set an exponentially distributed timer

with parameter proportional to $1/d'_T(x, y)$. If the timer expires before the request at $x$ or at $y$ is matched, then match them when the timer expires.

The key of the analysis is to reinterpret the execution of the algorithm as an Alternating Poisson Process. We will not recap the details of the proof here, except to mention that the height of the HST appears in the competitive ratio. The height of the tree is $\mathcal{O}(\log \Delta)$, where $\Delta = \frac{\max_{x \neq y \in \mathcal{X}} d(x,y)}{\min_{x \neq y \in \mathcal{X}} d(x,y)}$. We will see that the height of the HST commonly appears in these bounds.

Their final result is the following:

**Theorem 3.3.2** (Emek et al. [EKW16])**.** *There is an* $\mathcal{O}(\log^2(n) + \log(\Delta))$*-competitive algorithm for MPMD.*

### *3.3.1 Decreasing the competitive ratio*

The next major paper on MPMD was by Azar, Chiplunkar, and Kaplan [ACK17]. The paper has three main contributions: an introduction of the bipartite version of MPMD (which we discuss in section 3.5), a lower bound of $\Omega\left(\sqrt{\log n}\right)$ on the competitive ratio for any MPMD algorithm, and finally an algorithm that achieves an $O(\log n)$ competitive ratio for MPMD.

The intuition for the algorithm is very similar to the intuition we will use for our own algorithm on this problem, (and is the current best-known for the problem) so we describe it in detail.

As with [EKW16], the first step is to embed into an HST. Unlike Emek et al., Azar et al. can use height-reduced HSTs (inspired by the work of Bansal, Buchbinder, Madry, and Naor on the $k$-server problem [BBMN15]). These trees have height $\mathcal{O}(\log n)$, which will be one of the keys to improving the algorithm.

The other key is a deterministic algorithm, which achieves an $\mathcal{O}(h)$ competitive ratio on trees of height $h$.

We begin with the intuition behind this algorithm. Since our goal is to have a competitive algorithm, we want to avoid times where our algorithm is incurring cost, but OPT is not.

ALG and OPT will both incur costs as long as they have requests pending, and whenever they match requests to each other. The key to the analysis is to show that if OPT and ALG aren't incurring similar waiting costs, then it must be that OPT incurred some large amount of distance cost to cause the imbalance.

More specifically, fix some vertex $u$ of the tree, and consider the subtree $T_u$ rooted at $u$. If both ALG and OPT have the same number of requests pending inside $T_u$, then we do not need to worry about this tree (ALG is incurring just as much waiting cost here as OPT is). On the other hand if OPT has fewer requests pending, OPT will not be incurring the same amount of waiting time in that subtree. Our hope would be to show that OPT must have matched some request inside $T_u$ to another request far away and thus used significant distance cost to cause $T_u$ to be this imbalanced.

Unfortunately, only knowing that OPT has fewer pending requests than ALG is not sufficient to argue that OPT has incurred significant distance costs – it could be that it matched two pending requests inside $T_u$ to each other before ALG decided to. Luckily, this kind of matching is not concerning for us. Intuitively, an algorithm will have to wait a bit longer than OPT to match requests; OPT has foreknowledge of the coming requests and can match as soon as both requests are available (while ALG will have to wait a bit to ensure no better option is coming along). The algorithms for these problems ensure that the time and distance costs of our algorithms are on the same order of magnitude, so as long as OPT is making the same matching decisions as ALG is, the algorithm will be competitive. Instances where OPT and ALG pair different requests are the real concern of these arguments.

Observe that when OPT or ALG match a pair within $T_u$ the number of requests may be different, but their parities will not change. Indeed, the only way to change the parities of ALG's pending requests in $T_u$ or OPT's *relative to each other* is to have OPT or ALG match a request inside $T_u$ to one outside $T_u$. Observing that whenever the number of requests pending for OPT in $T_u$ is odd requires OPT to be incurring waiting time, parity will still suffice for arguing about waiting time.

With this intuition in hand, let's summarize their algorithm. Recall we have embedded

our metric space into an HST, so we care only about a tree metric, and the requests are appearing at the leaves. We use $w(e)$ to denote the weight of edge $e$ in the tree metric. ALG will maintain a forest $F$, a subset of the edges of the HST. $F$ starts empty. At every instant, consider each vertex $u$. If the number of requests pending inside $T_u$ is odd, "pay" at a unit rate toward $e_u$, the edge above $u$. When ALG has "paid" $w(e_u)$ toward $e_u$, add $e_u$ to $F$. Whenever every edge between two requests is in $F$, we match those two requests, and remove all used edges from $F$.

At a high-level, the correctness argument involves charging the purchases of $e_u$ to the cost of OPT. Note that if OPT has matched a request in $T_u$ to one outside $T_u$, then OPT has incurred a large amount of distance cost relative to one purchase of $e_u$. On the other hand, if OPT does not match from inside $T_u$ to outside $T_u$, then we consider two consecutive times where ALG bought the edge $e_u$. Over those two "phases" OPT had the same parity as ALG inside $T_u$ in exactly one of them (between the two purchases, ALG used the edge to match a request in $T_u$ to one outside $T_u$ so ALG's parity changed. Since OPT did not match from inside $T_u$ to outside $T_u$, OPT's parity did not). Thus, the design of the algorithm guarantees that during one of those two phases, OPT and ALG had the same parity, and so ALG was odd a constant fraction of the time ALG was (at least $w(e_u)$ time units, during the time that ALG was odd for $2w(e_u)$ units).

To handle all metrics, Azar et al. randomly embed into an HST. Notice that ALG now operates in the "wrong" metric. It is minimizing the distances in the tree metric, which can be wrong (in expectation) by an $\mathcal{O}(\log n)$ factor. But since the distance cost of ALG is so close to the total cost of OPT, and the trees are of small height, we still arrive at the desired result.

**Theorem 3.3.3** (Azar et al. [ACK17])**.**

$$\mathbb{E}[ALG] \leq \mathcal{O}(\log n)\, OPT$$

*Avoiding the Bad Examples*

Before moving on to deterministic algorithms, let's examine how these algorithms avoid the bad examples of section 3.2. Embedding into tree metrics sidesteps the first example. The key to the example is to create four points $a, b, c, d$ such that $d(a,b) \approx d(b,c) \approx d(c,d) \ll d(a,d)$, but since distances are dependent upon least-common-ancestors, it is not possible to achieve all of these requirements at once. The second example is handled with "memory." For the second example to achieve large competitive ratio, it must be able to repeat the "surprise appearance" arbitrarily many times. However, when ALG matches two requests at the same location, it does not remove any of the edges of $F$, nor does it forget about the waiting time it has already spent toward purchasing any of the edges. Once the trick has been repeated enough to buy a few edges, ALG connects two requests at different points. From then on, ALG will match the requests that appear $\epsilon$ time apart. That is, by purchasing edges, ALG can "remember" that it has left requests waiting for a long time already, and thus should not be afraid to match at a more substantial distance.

As a final remark for this section, we note that this algorithm nearly matches the best known lower bounds for the problem. The best known lower bound for MPMD is due to Ashlagi et al. [AAC+17].

**Theorem 3.3.4** (Ashlagi et al. [AAC+17])**.** *There is no (randomized) algorithm for MPMD which is* $o\left(\frac{\log n}{\log \log n}\right)$*-competitive.*

There is thus only a very small gap between the best-known upper and lower bounds when randomization is allowed.

## *3.4 Deterministic MPMD Algorithms*

While there is still a small gap between the upper and lower bounds for randomized MPMD (and MBPMD, see section 3.5), the problems are somewhat well-understood at this point. The situation is very different if randomization is not allowed. Deterministic algorithms offer some benefits. The randomized algorithms we have described behave very well in

expectation, but they leave open the possibility of a very unlucky choice of HST leading to a bad competitive ratio. In scenarios where an algorithm designer is more risk-averse, one might be willing to accept a higher competitive ratio in exchange for derandomization (thus getting a worst-case guarantee, instead of the average case expectation analyzes).[1] To date, there have been three main approaches to deterministic MPMD algorithms in the literature. A simple greedy algorithm, a smarter greedy-ish algorithm, and a very different LP-duality-based algorithm. We recap the intuition behind each in this section.

### *3.4.1 Local Algorithms*

We begin with the greedy-inspired approaches. Bienkowski, Kraska, and Schmidt have the first paper in this sequence [BKS17a]. The ideas behind both the algorithm and its analysis are straightforward. Whenever a request appears, start growing a ball around the request with the radius increasing at half the speed time is passing.[2] When two balls overlap (i.e. when the sum of their radii is at least the distance between the two points) **and** the radii of the two balls are within a factor 2 of each other, the two requests are matched. Growing balls around requests should be natural – the goal is to balance the costs incurred due to space and time. By slowly allowing worse and worse distance matches, we increase the chances that our time and space costs will be balanced.[3]

The second piece of the matching rule is less obvious, but it can be made intuitive by considering the example of subsection 3.2.2. In that example, one sees that a rule that always immediately matches nearby requests (or even requests directly on top of each other) can be exploited by an adversary to create a large competitive ratio. The adversary can force the

[1]Note that since MPMD is inherently an online problem, usual tricks to get high probability bounds, like taking the best of independent runs of the algorithm, do not apply to MPMD.

[2]Bienkowski et al. do not use the ball-growing metaphor to describe their algorithm (they talk about budgets). Azar and Jacob-Fanani use this metaphor in [AJF18] for their algorithm, and suggest applying the metaphor back onto [BKS17b]. Since we also use ball growing as a metaphor for our similar algorithms, we use this language whenever possible to emphasize the common thread in the literature.

[3]Of course the adversary can force these numbers to still be quite imbalanced, even on the ball-growing algorithm, but this approach limits the ways that the adversary can hurt us.

algorithm to match some requests immediately, while leaving others pending for extended periods, where the optimal matching still incurs little time penalty.[4]

The analysis mostly follows from a few simple observations about the algorithm: for example, if the algorithm decides to match request $r_1$ to $r_2$ when $r_3$ is pending, then $r_1$ and $r_3$ must be far in distance or have arrived at very different times (or else $r_1$ and $r_3$ would have been matched instead of $r_2$). The algebra required to bound the competitive ratio is more involved than it is enlightening, so we will not recap it here. Their final result is the following:

**Theorem 3.4.1** (Bienkowski, Kraska, Schmidt [BKS17a])**.** *There is a* ***deterministic*** *algorithm for MPMD that achieves an* $\mathcal{O}(m^{\log_2(5.5)}) \approx \mathcal{O}\left(m^{2.46}\right)$*-competitive ratio.*

Note that the ratio is now measured in terms of $m$ (the number of requests) not $n$ (the size of the metric space). Indeed, these deterministic algorithms all work on continuous metric spaces, even spaces that are unknown to the algorithm (assuming the algorithm has access to distances between requests).

Bienkowski et al.'s ball-growing approach is improved by Azar and Jacob-Fanani in [AJF18]. They also grow balls, but they do not do so in $(\mathcal{X}, d)$, the metric space where requests appear. Instead they grow in the metric space$(\mathcal{X} \times \mathbb{R}, D)$ where $D(r_1, r_2) = d(x_1, x_2) + |t_1 - t_2|$, i.e. the metric space where the distance between **requests** is the penalty OPT would face for matching those two requests. Moreover they grow only hemispheres; the balls only grow backward in the time axis (they do not grow forward in time, hence becoming only hemispheres).

The tools of the analysis are fundamentally similar to the ones used by [BKS17b], so we will not recap the arguments. Instead, we highlight performance of the hemisphere-growing on our two canonical examples. From the example from subsection 3.2.2, we see why Azar and Jacob-Fanani chose to grow only hemispheres instead of full spheres. If full spheres

[4]Since the competitive ratio proven in Bienkowski et al.'s paper is $\omega(m)$, an example showing that the balance condition is necessary to avoid an $\Omega(m)$ ratio is not perfectly compelling for the importance of the rule in this context, but as this same example is useful in other contexts, we simply use that one.

were grown, then the requests that appear at time $a - \epsilon$ would be matched to the request that appeared at time $a - 1$, creating the bad solution. On the other hand, when requests are only growing hemispheres: it takes only $\epsilon$ time for the appearances at time $a$ to have their hemispheres envelope the previous request at its location. On the other hand, requests at time $a - \epsilon$ will take $1 - \epsilon$ time before their hemispheres hit another request. Thus the algorithms actually find the optimal matching by looking only behind in time instead of looking forward.

On the other hand, both ball growing algorithms fail miserably when given the example from subsection 3.2.1. Both match greedily (as shown in Figure 3.2). This guarantees a competitive ratio of at best $\mathcal{O}(m^{\log_2(3/2)}) \approx \mathcal{O}(m^{0.46})$. Modulo technical details, this is a sharpness example for the competitive ratio of the algorithm.

**Theorem 3.4.2** (Azar and Jacob-Fanani [AJF18])**.** *There is a deterministic algorithm for MPMD that achieves an $\mathcal{O}(m^{0.46})$-competitive ratio.*

### *3.4.2 A different approach*

We were able to phrase all the algorithms in the last section as variations on "ball growing." Not all algorithms for MPMD fit nicely into this framework. Bienkowski, Kraska, Liu, and Schmidt consider a very different kind of algorithm for MPMD [BKLS18]. They use a primal-dual LP-based approach. They consider the linear program in Figure 3.4. Each $x_e$ is supposed to approximate an indicator that we will put $e = (r_i, r_j)$ into our matching. Let opt-cost$(e)$ be the cost that OPT would incur to match $e = (r_i, r_j)$, i.e. $d(x_i, x_j) + |t_i - t_j|$. For a set $S$ let sur$(S)$ be the number of unmatched requests in a maximum matching of $S$ (for MPMD, this is just $|S|$ mod 2), and let $\delta(S)$ be all possible (unordered) pairs of requests, with one request in $S$ and one outside $S$. We use $R$ to denote the set of all requests, and $E$ to be the set of potential matches. Consider the LP and its dual, shown in Figure 3.4.

The algorithm is a greedy algorithm on the dual LP. It maintains a partition of all the requests it has seen (both pending and matched) into "active sets." When a new request

$$\begin{aligned}
&\text{minimize} \sum_e \text{opt-cost}(e) \cdot x_e && && \text{maximize} \sum_{S \subseteq R} \text{sur}(S) y_S && \\
&\text{subject to} \sum_{e \in \delta(S)} x_e \geq \text{sur}(S) && \forall S \subseteq R && \text{subject to} \sum_{S : e \in \delta(S)} y_S \leq \text{opt-cost}(e) && \forall e \in E \\
&\qquad x_e \geq 0 && \forall e \in E && \qquad y_S \geq 0 && \forall S \subseteq R
\end{aligned}$$

Figure 3.4: The linear program of Bienkowski et al., along with its dual.

appears, a new active set containing only that variable is created. While a set $S$ is active and $\text{sur}(S) > 0$, $y_S$ grows at unit rate. When a constraint involving two active sets $S, T$ becomes tight, $S$ and $T$ are made inactive, $S \cup T$ becomes active, and if there are any possible matches in $S \cup T$ they are made. Verifying some technical details, the algorithm maintains a feasible dual point at every point in time. The final analysis uses duality to argue about the quality of the final result.

**Theorem 3.4.3** (Bienkowski, Kraska, Liu, Schmidt [BKLS18])**.** *The competitive ratio of the greedy dual algorithm is* $\mathcal{O}(m)$*.*

The gap between the primal-dual approach and the local approaches is not a result of the analysis. Indeed, Bienkowski et al. produce an MPMD instance on which their algorithm performs a factor $m$ worse than optimal (i.e. showing the competitive ratio is $\Theta(m)$). Their tightness construction has a similar flavor to the example in subsection 3.2.2.

### *3.5 Extensions of MPMD*

The version of MPMD we have discussed so far is not the only version of MPMD which has been studied. Since the introduction of the original problem, researchers have modified it to model problems that do not quite fit into the standard MPMD model. We briefly discuss the two main threads already in the literature.

### *3.5.1 Bipartite*

The most-studied extension of MPMD is the "bipartite" version of the problem (MBPMD). The bipartite version has a very similar problem statement, but with the addition that every request that appears has a polarity (either positive or negative), and matches can only be made between requests of opposite polarities. MBPMD is a better analogue for certain real-world matching problems. The most obvious such problem is that faced by ride-sharing apps (like Lyft and Uber). To model that problem, the positive requests represent people willing to give a ride and negative requests represent those looking for rides. As with the non-bipartite version, a perfect matching is always possible once all requests have appeared, i.e. there will be an equal number of positive and negative requests.

While the problems appear quite similar on the surface, there is no known general reduction from one to the other. However, researchers have had consistent success in adapting algorithms designed for MPMD to apply in the bipartite case. Indeed, for both randomized and deterministic algorithms, the best-known upper bounds are the same and come from similar algorithms, while the best-known lower bounds are similar constructions.

We begin with randomized algorithms. The best-known algorithm is due to Ashlagi et al. [AAC+17]. It follows very closely the ideas Azar et al. in [AAC+17], for the non-bipartite version, so we only highlight the differences. As before, the algorithm begins by reducing to the case of a tree. Rather than maintaining a single forest of purchased edges, for the bipartite version, Ashlagi et al. maintain two forests – one which represents positive requests waiting, the other represents negative requests waiting. Edges are purchased for the positive (negative) forest if the number of pending positive (negative) requests in a subtree outnumber the negative (positive) requests. If there is a pair of requests (one positive and one negative) such that the positive forest has every edge from where the positive request is pending to the least-common-ancestor of the two requests and the negative forest has every edge from the negative pending request to the least-common-ancestor, then the two requests are matched.

**Theorem 3.5.1** (Ashlagi et al. [AAC+17]). *There is an $O(\log n)$-competitive algorithm for MBPMD.*

The proof of correctness is essentially the same except that the second forest induces increases the competitive ratio by a constant factor.

The lower bound construction also can be adapted to the bipartite case, but at a more significant cost.

**Theorem 3.5.2** (Ashlagi et al. [AAC+17]). *There is no (randomized) algorithm for MBPMD with a $o\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ competitive ratio.*

Note that there is still a $O(\sqrt{\log n/\log\log n})$ gap between the best-known upper and lower bounds for (randomized) MBPMD.

On the deterministic side, again a minor modification causes the best-known algorithm to lose only a constant factor in the competitive ratio. Azar and Jacob-Fanini's algorithm can be modified (by just adding a rule that the algorithm matches only if the requests to be matched are of opposite polarity).

**Theorem 3.5.3** (Azar and Jacob-Fanini [AJF18]). *There is a deterministic MBPMD algorithm that achieves an $\mathcal{O}(m^{0.46})$-competitive ratio.*

Incidentally, Bienkowski et al.'s primal-dual algorithm can also be modified to function for the bipartite version, with the same $O(m)$ competitive ratio.

### *3.5.2 Alternate costs*

Having linearly increasing costs for distance and time is a natural choice for an initial model, but this may not always accurately reflect reality. A natural alteration to the problem is to impose super-linear costs for waiting time. A person may not notice the difference between a 20 second wait and 30 second wait on our gaming server, but is likely to grow increasingly frustrated as the waiting time goes beyond 5 minutes. Liu, Pan, Wang, and Wattenhofer

therefore consider what happens if the time penalty grows as a convex function [LPWW18]. We follow [LPWW18] and call this problem convex-MPMD.

Intuitively, making the penalty an increasing function of the wait time seems to make the problem much harder – the existing algorithms for MPMD all strategically leave a request pending to balance the potential distance and time costs. With a convex time penalty the ability to leave a request is severely limited. To simplify the new tradeoff, they only consider the problem on uniform metric spaces.

In this new, simplified setting they show *for many simple convex functions* the competitive ratio is $\Theta(n)$.

**Theorem 3.5.4** (Liu et al. [LPWW18])**.** *There is an algorithm for convex-MPMD that achieves an $O(n)$-competitive ratio on uniform metric spaces.*

**Theorem 3.5.5** (Liu et al. [LPWW18])**.** *Every convex-MPMD algorithm on uniform metric spaces incurs penalty $\Omega(n)$.*

The algorithm that achieves the $O(n)$ ratio is not patient – if $\delta$ is the distance between any two points in $\mathcal{X}$, and some pair of requests have both waited $2\delta$ time it is immediately matches that pair. It is also "jumpy" in the sense of immediately matching requests pending at the same point. We will not cover the details of the algorithm or the lower bound.

### *3.6 Our Initial Work – Stochastic Setting*

We now turn to our own work on MPMD. In our survey of the prior literature, we observed that ball growing is a common (and useful) metaphor for describing deterministic algorithms for MPMD, but has not been used in randomized algorithms. We are able to design a ball-growing based algorithm for MPMD that matches the best known randomized algorithms, connecting ideas from the two threads of MPMD literature. Before getting to that algorithm, we increase our intuition for ball growing by showing a simple version in a stochastic setting.

In the standard MPMD setting, an adversary chooses the location and time of appearance for each request. Such a model is useful if one is concerned about malicious inputs or does

not know much about the requests that will be given to the algorithm. However, if one desires to model a system where the inputs are predictable (e.g., from historical data) and not likely to be manipulated, it may be more accurate to view the requests as generated randomly.

Our[5] initial work on MPMD focuses on this problem. We assume that the interarrival times (i.e. the time between when requests appear) are drawn i.i.d. from an exponential distribution. When a new request appears, its location is drawn according to a (fixed, but unknown) distribution over the points in the metric space.

In this setting, it no longer makes sense to think about competitiveness as the worst possible ratio between our performance and the optimal. Any instance an adversary could invent has some (very small) probability of actually begin drawn[6]. The probabilistic assumption does not give any benefit for the worst-case. Instead we'll consider the expected value (over only the drawing of the instance – our algorithm is deterministic) of our algorithm's performance against the optimal matching. In a slight abuse of terminology, we will still refer to this number as the competitive ratio.

We show that a simple algorithm has expectation within an $\mathcal{O}(1)$-factor of optimal. Our algorithm simply grows a ball around every request (with the radius increasing at unit rate). When two balls intersect, match the two requests.

### *3.6.1 The "Ball Growing" Algorithm*

The key idea behind all of our algorithms is "ball growing." Each request maintains a number, which we think of as the radius of a ball surrounding the point in the metric space. In the basic version of the algorithm, the radius is exactly how long that point has been waiting to be matched, i.e. for each point the radius starts at 0 when the point appears and increases continuously at a unit rate. As soon as two balls intersect, we match the

[5]this is joint work with Anna Karlin, Shayan Oveis Gharan, and Alireza Rezaei

[6]or more accurately a slightly perturbed version, since our model does not allow for exactly simultaneous arrivals.

corresponding points. Note that this matching rule implies that the balls for unmatched points are always disjoint.

Formally, we use the following model. Let $(\mathcal{X}, d)$ be an underlying metric space. In this section, we make no assumptions on the underlying metric space. For example, our spaces are allowed to be continuous.

Requests arrive randomly in the space as follows:

- **Interarrival times:** When each request arrives, we start a new (independent) exponential clock with parameter $\lambda$. When the clock fires, a new request appears.

- **Location of arrival:** The location of each new request is independently drawn from the density function $f(\cdot)$.

### *3.6.2 Analysis*

Recall that our algorithm creates a ball of radius 0, centered at every new request that appears. The radius of each ball increases at a constant rate so that $t$ time units after it arrives, the radius of the ball is $t$. When the balls for two requests overlap, we match those two requests.

Let $B(x, r) = \{y : d(x, y) \leq r\}$ be the (closed) ball of radius $r$ centered at $x$. Let $B_{\text{open}}(x, r) = \{y : d(x, y) \leq r\}$ be the corresponding open ball. We use ALG to denote the total cost of our algorithm, $\text{ALG}_{\text{time}}$ to refer to the time cost, and $\text{ALG}_{\text{dist}}$ to refer to the distance cost.

The key to the analysis is the following definition

**Definition 3.6.1** ($r_x$)**.** *Define $r_x$ as the smallest $r$ for which the following inequality is true:*

$$\frac{1}{\lambda p_{B(x,r)}} \leq r, \qquad \textit{where} \qquad p_{B(x,r)} = \int_{z \in B(x,r)} f(z)dz$$

Since the left hand side decreases as right hand side increases, they must cross, and thus $r_x$ is well-defined.

Notice therefore that

$$\lambda p_{B(x,r_x)} \geq \frac{1}{r_x}$$

and

$$\lambda p_{B_{\text{open}}(x,r_x)} \leq \frac{1}{r_x}, \quad \text{where} \quad p_{B_{\text{open}}(x,r_x)} = \int_{z \in B_{\text{open}}(x,r_x)} f(z)dz$$

.

Observe that $1/\lambda p_{B(x,r)}$ is the expected time to an arrival in $B(x,r)$. This observation leads us to our two key lemmas. Suppose a request appears at $x$. With constant probability, another request does not appear in $B_{\text{open}}(x,r_x)$ for at least $r_x$ time, so any algorithm will incur $\Omega(r_x)$ cost. On the other hand, our ball growing algorithm will expect to pay only $O(r_x)$ to match a request at $x$. We formalize these observations in the next two lemmas.

**Lemma 3.6.2.** *$ALG \leq 4\int_{\tau=0}^{T}\int_x \lambda f(x) r_x \, dx \, d\tau$*

*Proof.* Observe that the distance cost for any request matched in ball growing is always at most the time cost of matching that request. We claim that the appearance of a new request inside $B(x,r_x)$ at least $r_x$ time units after the request at $x$ appears guarantees the original request will be matched. Indeed, by then if the request has not been match, its ball certainly contains the point where the new request appears. Thus the request at $x$ will be matched, at the latest, the next time after $t + r_x$ that a new request appears in $B(x,r_x)$ Observe that the time to a new request appearing in $B(x,r_x)$ is an exponential random variable with parameter $\lambda p_{B(x,r_x)} \geq 1/r_x$. We thus can bound the expected cost to $\text{ALG}_{\text{time}}$ as follows:

$$\begin{aligned}
\text{ALG}_{\text{time}} &\leq \int_{\tau=0}^{T}\int_x \lambda f(x) \int_0^\infty (t + r_x)\frac{1}{r_x} e^{-t/r_x} dt \, dx \, d\tau, \\
&= \int_{\tau=0}^{T}\int_x \lambda f(x) \left( \int_0^\infty t\frac{1}{r_x} e^{-t/r_x} \, dt + \int_0^\infty e^{-t/r_x} \, dt \right) dx \, d\tau, \\
&\leq \int_{\tau=0}^{T}\int_x \lambda f(x) 2r_x \, dx \, d\tau.
\end{aligned}$$

where the last inequality uses that the first integral in parentheses is the expectation of the

time for an arrival inside $B(x, r_x)$, and the second integral is $r_x$ times the integral of the pdf of an exponential.

Doubling $\text{ALG}_{\text{time}}$ we have the required bound on ALG. □

**Lemma 3.6.3.** *$OPT \geq e^{-2} \int_{\tau=0}^{T} \int_x \lambda f(x) r_x \, dx \, d\tau$,*

*Proof.* We now lower bound OPT. Suppose a new request appears at $x$ at time $\tau$. We show that with constant probability there is no other request that OPT could match $x$ with which does not incur at least $r_x$ cost to OPT.

Let $\alpha_x(\tau)$ be the probability there is no other arrival within $B_{\text{open}}(x, r_x)$ during the time window $[\tau - r_x, \tau + r_x]$. For any such request, OPT must incur $r_x$ cost in the matching, thus we have:

$$\text{OPT} \geq \int_{\tau=0}^{T} \int_x \lambda f(x) r_x \alpha_x(\tau) \, dx \, d\tau,$$

Finally, observe that

$$\alpha_x(\tau) \geq 1 - \int_{\tau}^{\tau + r_x} \frac{1}{r_x} e^{-t/r_x} \, dt - \int_{\tau - r_x}^{\tau} \frac{1}{r_x} e^{-t/r_x} \, dt = 1 - \int_0^{2r_x} \frac{1}{r_x} e^{-t/r_x} \, dt = e^{-2}.$$

□

Combining Lemma 3.6.2 and Lemma 3.6.3 we immediately have

**Theorem 3.6.4.** *In our stochastic setting, ball growing is an $O(1)$-competitive algorithm.*

**Remark 3.6.5.** *The ball-growing algorithm does not require advance knowledge of the underlying metric space. It also does not require knowledge of the probability distribution from which the requests are drawn.*

### *3.7 Our Initial Work – Adversarial Setting*

We now consider an adversarial setting, where the requests are chosen by an (oblivious) adversary not drawn from a probability distribution. Following much of the prior work on

this problem, we assume that the underlying metric space $(\mathcal{X}, d)$ is finite and known in advance.

Our analysis is very similar to the analysis of Azar et al. [ACK17]. Where possible, we have adopted their notation and the structure of their proof. We adapt ball growing to match the best-known performance for adversarial instances for finite metric spaces. As with [ACK17], embed the metric space into an HST of height $\mathcal{O}(\log n)$.

We need to carefully define what balls mean in our new context, and how they should grow. Every request grows a ball "up" the tree (despite being a discrete metric, we think of balls as growing continuously along edges, but only growing toward the root, not back down other branches of the tree). Let $T_B$ denote the subtree rooted at the vertex that the boundary of $B$ most recently grew past. We say a ball $B'$ is "inside" a ball $B$ if the boundary of $B'$ is inside $T_B$ or if the boundaries of $B'$ and $B$ are on the same edge and $B$ is no higher than $B'$ on that edge. At each time instant $t$ and for each request $r$, if the number of balls inside $B_r$ (including $B_r$ itself) is odd, then let $r$'s ball grow at a unit rate. Otherwise $B_r$ is fixed.

In subsection 3.7.1 we reduce to MPMD on certain tree metrics. We then describe the "Odd Ball Growing Algorithm" on (arbitrary) tree metrics in subsection 3.7.2. Finally in subsection 3.7.3, we analyze ball growing on tree metrics and prove the following theorem:

**Theorem 3.7.1.** *Odd Ball Growing is an expected-$O(\log n)$ competitive algorithm on $n$-point metric spaces in the adversarial setting.*

### *3.7.1 Reduction to Tree Metrics*

Prior work in this setting has first embedded $\mathcal{X}$ into a distribution over hierarchically separated tree (HST), and used the structure of the HST to design the algorithm. We do the same. Emek et al. [EKW16] make use of binary HSTs, while Azar et al. [ACK17] use weighted HSTs to lessen the height of the trees (cf. [BBMN15]). We follow Azar et al. The reader should see [WS11] for more on embedding into HSTs. For our purposes, the following

notion of HST suffices:

Let $\mathcal{D}$ be a distribution over HSTs. We say we have embedded $(\mathcal{X}, d)$ into $\mathcal{D}$ if:

1. for all $T$ in the support of $\mathcal{D}$, $d'_T(u,v) \geq d(u,v)$ for all $u, v \in \mathcal{X}$

2. $\mathbb{E}_{T\sim\mathcal{D}}[d'_T(u,v)] \leq O(\log n)d(u,v)$ for all $u, v \in \mathcal{X}$.

Where $d'_T(u,v)$ is the distance in tree $T$ between the leaves corresponding to $u$ and $v$. Standard techniques (see e.g. [FRT04] or [WS11, Theorem 8.17]) allow us to draw from such a distribution efficiently.

### *3.7.2 Algorithm*

Our algorithm can be run on any rooted tree. By adding dummy leaves with edges of weight 0 to each internal vertex, we may assume without loss of generality that the requests appear only at the leaves of the tree. Arbitrarily root the tree if it does not already have one, and add an imaginary edge of infinite weight above the root.

Consider the following variant of ball-growing for HSTs. As before, each request grows a ball. It starts at a leaf of the HST (corresponding to where the request appeared in the original metric space). The boundary of the ball grows only "up" the tree (i.e. only away from the leaves). Despite the underlying metric space being discrete, we will speak as though the balls partially fill the edges of the tree as they grow. We will refer to the edge above the vertex $u$ as $e_u$. Define $T_u$ to be the subtree rooted at $u$ along with the edge $e_u$. For a ball $B$ whose boundary is on $e_u$, we will write $T_B$ to refer to the subtree $T_u$. We say a ball is "in" a subtree $T'$ if the boundary of its ball is in $T'$. Note that even if a request begins inside some subtree, its may cease to be in that subtree (if it grows beyond the highest edge of that subtree).

We now describe how the balls grow and how to match requests. We say a ball $B_1$ is *in* a ball $B_2$ if (1) the boundaries are on the same edge and $B_2$'s boundary is no higher than

$B_1$. or (2) $B_2$'s boundary is on another edge in $T_{B_1}$. We will say "strictly inside $B$" if we wish to exclude $B$ itself.

At each time instant $t$ and for each request $r$, if the number of balls in $B_r$ (including $B_r$ itself) is odd, then let $r$'s ball grow at a unit rate. Otherwise $B_r$ is fixed.

We now describe how to match requests. We make explicit use of the edges of the tree in our matching rules.

Rule 1 If the boundaries of two (adjacent) balls are on the same edge, and their growth across that edge is within a factor of 2, then match them.

Rule 2 If there are four balls whose boundaries are on the same edge, then match the bottom two balls to each other.

Rule 3 If there are two balls whose boundaries are on the (imagined) edge above the root, match them.

The intuition behind the tree version of ball-growing is generally similar to the intuition behind the basic version. We motivate some of the changes: the use of parity is inspired by [ACK17]. Intuitively, as long as we have an odd number of pending requests in a subtree, then either OPT does too (and it must be incurring waiting time in that subtree) or OPT must have matched a request inside that subtree to a request outside that subtree (and thus it incurred distance cost). Rule 2 makes our charging arguments easier, as it ensures there will only ever be three balls growing on the same edge for more than an instant.

### *3.7.3 Analysis*

In this section, we prove Theorem 3.7.1. In our intermediate technical results, we must compare to an arbitrary solution matching SOL instead of OPT. We are interested in comparing to the optimal algorithm for the original metric space, not the optimal matching for the HST we embedded into. Since the embedding distorts the distance costs, the optimal matching on the HST need not be the same as the one for the original metric space.

The main technical result is the next theorem.

**Theorem 3.7.2.** *On any tree of height h, for any solution SOL, the ball growing algorithm achieves:*

$$ALG \leq \frac{75}{2}(SOL_{dist} + hSOL_{time})$$

Our proof proceeds as follows: we first show that ALG is bounded by the sum of the radii of the balls at the time they were matched. We then define a scheme to assign the growth of the radii to the vertices of the tree (the "$y_u$" and "$z_u$" play that role – see Definition 3.7.5 and Definition 3.7.6) and a similar scheme from the time and distance costs of SOL to the vertices (these are "$x_u$" and "$x'_u$" – see Definition 3.7.12). We then argue vertex by vertex that $y_u + z_u$ are at most a constant factor more than $x_u + x'_u$.

Let $R$ be the set of all requests presented by the adversary, and let $\text{radius}_r$ be the radius of $r$'s ball at the time $r$ was matched (in ALG). We begin by showing the final radii are a good approximation to the cost of the algorithm.

**Lemma 3.7.3.** $ALG_{time} \leq 2 \cdot \sum_{r \in R} radius_r$

*Proof.* We show that at any time at least half of the balls are growing. It suffices to find an injective mapping, $f(\cdot)$ from non-growing balls into growing ones. Consider any not growing ball, $B$. Since it is not growing, there must be an odd number of balls strictly inside $B$ (excluding $B$ itself).

If there is another ball whose boundary is on the same edge as $B$ but below $B$, define $f(B)$ to be the highest such ball. Otherwise, $B$ is the lowest ball on its edge, and all other balls inside of it are strictly inside one of child subtrees. Since there are an odd number total, at least one of the child subtrees must have an odd number of balls. Continue recursing into a subtree with an odd number of balls until there is a ball on the edge above the root of that subtree. The top such ball, (call it $B'$) must have an even number of strictly balls inside it, so it is growing. Set $f(B) := B'$.

Our mapping is indeed an injection, as the image of any $B$ is a ball $B'$ where $B'$ is strictly contained in $B$ but such that there is no ball $B''$ such that $B'$ is contained in $B''$ and $B$ is

not contained in $B''$. So the preimage of any ball can be found by going up the tree until one hits a non-growing ball.

Thus at least half of the balls are growing at any time step, and the claim follows. □

**Lemma 3.7.4.** $ALG_{dist} \leq \sum_{x \in R} radius_x$

*Proof.* To match two requests, their balls must have reached a common edge. The common edge must be above the points' least common ancestor, thus if $u$ and $v$ are matched $\text{radius}_u + \text{radius}_v \geq d'_T(u, v)$. □

Our goal in the next step of the proof is to argue vertex by vertex about the costs of the algorithm in comparison to SOL. In order to make this argument, we assign the radii growth to the vertices of the tree. The following definitions of $y_u$ and $z_u$ will serve this purpose. In the following, a "trip" across an edge is the time that the boundary of the ball was growing on that edge. If a request is matched before it finishes growing across some edge, the trip is only "partial" or "unfinished." Intuitively, $y_u$ will represent the ball growth that corresponds to completed trips across the edge $e_u$ (i.e., the edge in the tree above vertex $u$), while $z_u$ will represent the partial trips.

**Definition 3.7.5** ($y_u$)**.** *Define $y_u$ for each vertex $u$ as follows:*

- *If $u$ is the root, let $y_u$ be 0.*
- *Otherwise, let $y_u$ be the growth of all completed trips across $e_u$ (with no contribution from unfinished trips)*

Note that if $k$ balls complete a trip across $e_u$ then $y_u$ is exactly $k$ times the weight of $e_u$. We define the $z_u$ to handle "partial" trips.

**Definition 3.7.6** ($z_u$)**.** *Define $z_u$ for each vertex $u$ as follows:*

- *If $u$ is the root, let $z_u$ be the total growth across $e_u$*

- *Otherwise, every time a ball is matched on $e_u$ by Rule 2 increase $z_u$ by the growth of the top of the two matched balls on $e_u$.*

**Remark 3.7.7.** *The exact definition of $z_u$ may seem strange – our goal is to separate the arguments about rule 2 from the rest of the argument. After showing Lemma 3.7.11, we will be able to focus only on edges matched according to rules 1 and 3.*

Before we begin the actual analysis, we will make a few simple observations about the algorithm.

**Observation 3.7.8.** *By Rule 1, a ball never "passes" another.*

**Observation 3.7.9.** *Balls must be on the same edge, with no balls between them, to be matched.*

This observation may not be obvious for Rule 1. Suppose $B_1$ and $B_2$ can be matched according to Rule 1. A hypothetical ball between them would have growth much less than a factor 2 away from at least one of $B_1$ and $B_2$. Regardless of how they were growing, it would have been possible to match the middle ball to one of the others a small time earlier, so there cannot be a ball between $B_1$ and $B_2$.

**Observation 3.7.10.** *Once a ball becomes the top one on its edge, it will continue to be the top ball on its edge until it is matched by Rule 1 or it grows beyond the edge.*

With these observations in mind, we can proceed with our proof. We begin by showing the sum $y_u + z_u$ does actually correspond closely with $\sum_{x \in r} \text{radius}_x$:

**Lemma 3.7.11.** $\sum_{u \in T} y_u + z_u \leq \sum_{x \in R} radius_x \leq \frac{5}{2} \sum_{u \in T} (y_u + z_u)$

*Proof.* The first inequality follows immediately from the definitions. For the second inequality, we have not accounted for growth caused by balls matched according to Rule 2, nor for the "bottom" ball when we match according to Rule 1.

By Rule 2, no edge ever has 4 balls growing across it (when the fourth appears it is immediately matched), thus when we match according to Rule 2, there is only one ball

that has grown across $e_u$ that we must account for. Call this ball we must account for $B_3$ (because it is the third on the edge). Consider $B_1$, the top ball on $e_u$ when we match $B_3$. By Observation 3.7.10, $B_1$ will increase either $z_u$ or $y_u$. Charge the growth of $B_3$ on $e_u$ to the increase caused by the $B_1$. Observe that whenever $B_3$ was growing $B_1$ was too (since there is one ball between them). Thus each ball is responsible for at most twice its own growth in this paragraph.

The only other growth we have not accounted for is the growth by the second ball matched according to Rule 1. Every such ball has at most half as much growth across $e_u$ as the ball it is matched to.

Thus each ball is responsible for at most 2.5 times its own growth (its own, its match's and balls matched by Rule 2 while it is the top ball). □

**Definition 3.7.12** $(x_u, x'_u)$**.** *Let $x_u$ be the time cost incurred by SOL by all requests in the subtree rooted at $u$.*
*Let $x'_u$ be the distance cost incurred by SOL for using the edge above $u$.*

Observe that the $x'_u$ values exactly split up $\text{SOL}_{\text{dist}}$, while the $x_u$ overestimate $\text{SOL}_{\text{time}}$ by up to a factor of the height of the tree (since a single request could be the only one pending but cause all ancestors in the tree to have their $x_u$ increase).

**Lemma 3.7.13.** *For all $u$, $y_u \leq 3(x_u + x'_u)$*

*Proof.* If $u$ is the root, $y_u$ is 0, so the claim holds. Assume $u$ is not the root.

By definition, $y_u$ only increases on a complete trip across $e_u$. We divide time into phases. We start the first phase when the algorithm starts. Every time a ball finishes growing across $e_u$, end the current phase. Consider only completed phases (i.e. if the algorithm ends before a phase ends, ignore the final partial phase). Observe that $y_u$ is exactly the number of completed phases times $w(e_u)$, the weight of edge $e_u$ in the tree metric.

We first argue assuming that there are at least 2 phases. We compare the following parities: Let $\alpha_u$ be the parity of the number of balls in $T_u$ in our algorithm and let $\beta_u$ be

the number of unmatched requests for vertices in $T_u$ in SOL. Note the asymmetry of this definition – in SOL we count requests (as SOL may not operate by growing balls) and a request only ceases to be counted if it is matched. For ALG a ball will cease to be counted if it grows past $e_u$ (or if it is matched).

Observe that $\alpha_u$ flips exactly when

($\alpha$1) A ball finishes growing across $e_u$.

($\alpha$2) A new request appears at a leaf in $T_u$.

and $\beta_u$ flips exactly when

($\beta$1) SOL matches a request pending at a leaf in $T_u$ to a request pending outside $T_u$.

($\beta$2) A new request appears at a leaf in $T_u$.

Now consider two consecutive phases. Observe that if $\alpha_u$ and $\beta_u$ are equal, they become unequal if and only if events $\alpha$1 or $\beta$1 happen. If event $\beta$1 occurs, then that match causes $x'_u$ to increase by $w(e_u)$. Our two consecutive phases increase $y_u$ by $2w(e_u)$; charge the full increase to the change in $x'_u$. Otherwise, $\beta$1 does not happen. $\alpha$1 happens exactly between the two phases, so in (exactly) one of the two phases $\alpha_u = \beta_u$. During that phase, whenever the ball was growing, $\alpha_u$ was odd, and therefore $\beta_u$ was also odd and $x_u$ increased by at least $w(e_u)$. Charge both increases of $y_u$ to that change.

Repeat this argument across all pairs of phases. If the total number of phases is odd, charge the last phase to the most recent pair. If there are $p$ phases, we have at least $p/3$ pairs, so we have $y_u \leq 3(x_u + x'_u)$.

We now consider the cases where there is at most one completed phase. If there are no completed phases, then $y_u$ is 0 and the claim follows. If the number of completed phases is exactly one, we can still charge to $x_u$ and $x'_u$. Indeed, let $B$ be the ball that finished growing across $e_u$. Note that $\alpha_u$ and $\beta_u$ are equal at time $t = 0$. If SOL uses $e_u$ to match at any time in the course of the algorithm, we may charge to $x'_u$. Otherwise, $\alpha_u$ and $\beta_u$ are equal until $B$

finishes its trip across $e_u$. Since $B$ grows across $e_u$ when $\alpha_u$ is odd, the time it grew across $u$ is $w(e_u)$ time where $\beta_u$ was also odd, and we have $y_u \leq x_u + x'_u$. □

We now turn to bounding $z_u$. We use a parity argument similar to Lemma 3.7.13, though the definitions are slightly more involved.

**Lemma 3.7.14.** *For all $u$, $z_u \leq 2(x_u + x'_u)$*

*Proof.* If $u$ is the root, observe that a ball growing across $e_u$ has all other balls in the algorithm inside of it. Thus it grows if and only if the total number of pending requests for OPT is odd. The parity of the number of pending requests in SOL is the same, so whenever $z_u$ is increasing, $x_u$ is increasing as well, and we have $z_u \leq x_u$.

Now suppose $u$ is not the root. We again divide time into phases. End a phase whenever balls are matched on $e_u$ according to Rule 1. Now consider each phase. We again define an $\alpha$ and $\beta$. Let $B$ be the top ball which is matched to end this phase, and $B'$ the bottom ball matched to end this phase. Let $\alpha_u$ be the parity of the number of balls in $B$ (including $B$ itself). Let $\beta_u$ be the parity of the number of pending requests in $T_u$ in OPT.

Note that the definition of $\alpha_u$ is different in this proof from the proof of Lemma 3.7.13. We again consider when $\alpha_u$ and $\beta_u$ change relative to each other. With the change in the definition, the only possible way for $\alpha_u$ and $\beta_u$ to change relative to each other is for SOL to match a request in $T_u$ to outside of $T_u$.[7]

We now argue in a single phase. In a given phase, if SOL used $e_u$ to match during this phase, then $x'_u$ increased by $w(e_u)$, which is at least the increase to $z_u$ in this phase. If SOL did not use $e_u$, then we know that $\alpha_u$ and $\beta_u$ did not change relative to each other for the entire phase. If they were always equal, then $x_u$ increased whenever $B$ grew. Otherwise, $x_u$ increased whenever $B'$ grew. In either case, since we only match when $B$ and $B'$ have growth within a factor 2, $x_u$ increased by at least half of the increase to $z_u$ in this phase.

In either case we have have the claim. □

[7]Unlike in the previous proof, we no longer need worry about $B$ growing beyond $e_u$, since we know it will be matched before it leaves the edge

We can now prove the main technical result

*Proof of Theorem 3.7.2.* ALG $\leq 3\sum_{x\in R}\text{radius}_x \leq \frac{15}{2}\sum_{u\in T} y_u + z_u \leq \frac{75}{2}\sum_{u\in T} x_u + x'_u$ where the first inequality is by Lemma 3.7.3 and Lemma 3.7.4 and the second follows from Lemma 3.7.13 and Lemma 3.7.14. Recall that we defined $x'_u$ to be the distance cost incurred by SOL for using the edge above $u$, and $x_u$ to be time that the subtree rooted at $u$ has an odd number of requests. Summing over all $u$, $x'_u$ becomes exactly $\text{SOL}_{\text{dist}}$. If we sum over $x_u$, then at each moment, any request may be causing all of its ancestors $u$ to increase their $x_u$, so the time cost of SOL could be a factor $h$ (the height of the tree) less than $x_u$, and we have: $\text{ALG} \leq \frac{75}{2}\sum_{u\in T} x_u + x'_u \leq \frac{75}{2}(\text{SOL}_{\text{dist}} + h\text{SOL}_{\text{time}})$ □

We now show that our main result follows from Theorem 3.7.2

*Proof of Theorem 3.7.1.* Consider an arbitrary metric space $(\mathcal{X}, d)$ on $n$ points.

We can embed $\mathcal{X}$ in a distribution $\mathcal{D}$ over weighted HSTs. Our algorithm draws a tree according to $\mathcal{D}$ then runs the odd-ball-growing algorithm on that HST. By Theorem 3.7.2, in the tree metric, $\text{ALG} \leq \text{SOL}_{\text{dist}} + O(\log n)\text{SOL}_{\text{time}}$. The conversion to tree metric alters the distance cost, but by point 2 of the HST definition, we have:

$$\mathbb{E}_{\mathcal{D}}[\text{ALG}] \leq O(\log n)\text{SOL}_{\text{dist}} + O(\log n)\text{SOL}_{\text{time}} \leq O(\log n)\text{SOL}$$

as required. □

### *3.7.4 Comparison to Azar et al.*

We have followed the argument of Azar et al. very closely. From the overwhelming similarity in the proofs, one might believe that our algorithm and theirs are fundamentally the same; this is not the case. Their intuitions are very similar, but they operate differently. Consider the example in subsection 3.2.2. Azar et al. will match the requests at the same points, paying $1 - \epsilon$ toward the edges until it pays for all the edges to the least common ancestor of the two points. It will then match two distant requests. From then it will be able to

match the points at time $\epsilon$.[8] Observe that the key to the algorithm getting good behavior is *memory*. Each time the algorithm is "tricked" it buys more of the edges above it. Even if no requests are pending (as will happen frequently), the algorithm's state (via the edges) ensures it cannot be tricked indefinitely.

Our algorithm behaves quite differently. We do not keep global memory. If no requests are pending, our algorithm records no state. Thus we must handle the example in subsection 3.2.2 differently. Observe that our algorithm is not "jumpy," i.e. we can have multiple requests pending at the same leaf without matching them. Indeed, on the example from subsection 3.2.2, we will grow balls for the first requests to radius $1 - \epsilon$ before the second set of requests appear. By the time the third set of requests appear, the first set of balls (at radius $1 - \epsilon$) will still be much larger than the radii of the second set (only $\epsilon$), so we do not match the first pairs by location as Azar et al. do. We will pair the second set with the third set. Allowing the algorithm to run, we will match the first pair of requests to each other, and end up with the same pairs as OPT (though incurring more waiting time).

We include this as only an example of the differences in the algorithms; despite the surface similarities they are truly different.

## *3.8 Open Problems*

We now discuss a few open problems and avenues for how they might be addressed.

### *3.8.1 MPMD*

In the stochastic setting, the most obvious avenues for open questions are to generalize our models. A first step would be to argue what happens if requests stop appearing (our current model assumes an infinite time horizon). One approach here would be to try to use the diameter of the space to bound the costs of whatever requests are pending after the final request has appeared. Other possibilities in the stochastic setting include allowing $\lambda$, the

[8] As it matches the points at time difference $\epsilon$, it also pays $\epsilon$ and will eventually buy the edges again and cycle back and forth. The behavior is still good because it matches far more at time $\epsilon$.

frequency at which requests appear, to change. We have initial results assuming $\lambda$ is evolving according to Brownian motion. In the extreme case, one could imagine a problem where an adversary controlled the times when requests appeared, but not their locations.

In adversarial settings there are a few obvious open questions. The most obvious is to close the gap between the upper and lower bounds on (randomized) algorithm performance in terms of $n$. The gap (between $\mathcal{O}(\log n)$ of [ACK17] and $\Omega\left(\frac{\log n}{\log \log n}\right)$ of [AAC+17]) is extremely small, but it is still super-constant asymptotically. It's worth noting that achieving a $o(\log n)$-competitive algorithm would require a significant change in approach. It is already known that there are metric spaces which require $\Omega(\log n)$ distortion [Bar96].

On the deterministic side, we still have substantial gaps. As far as we know, an algorithm could achieve poly-logarithmic performance, which is very far from the $\mathcal{O}(m^{0.59})$ of the best upper bound. Again, the barrier for improving the algorithm is well-understood. MPMD algorithms in the literature tend to be local. Each request looks in an (often expanding) neighborhood around it to find its match, but the deterministic algorithms do not try to maintain any "global state" – they make only local decisions. But local decision-making can be tricked, as we saw with the example in subsection 3.2.1. Indeed, the competitive ratio of $\mathcal{O}(m^{0.59})$ matches the ratio a greedy algorithm gets on this example exactly.

### *3.8.2 Extensions*

For extensions of MPMD, we can take each of the questions we've already asked, and ask them again for MBPMD. In addition there are some questions unique to the extensions: In particular, we have no initial results for MBPMD in the stochastic setting, and it would be a good first step to just see if our ball growing algorithm works in MBPMD – essentially the only difference between the two settings is that balls need not be disjoint in the bipartite setting (but this exact difficulty did not prevent [AJF18] from adapting their ball growing in the adversarial setting, so we should be very optimistic). It would also be interesting to try to adapt odd ball growing to the bipartite setting.

Finally, we could also examine further extensions of different cost functions. An obvious

extension is to try to adapt the results on convex time functions to tree metrics. If they can be extended then HST embedding tricks are likely to give us good results on general metric spaces. Another extension would be to examine concave time functions.

# Chapter 4

# TOURNAMENT DESIGN: CREATING GOOD MATCHUPS IN SPORTS

A *tournament rule* is a mechanism for taking results of games played between $n$ teams and deciding a (single) winner of the tournament. Recent work has considered manipulability of these tournaments. A major impetus for much of this recent work was an incident in the 2012 Olympic Badminton Tournament where both teams playing in a match were incentivized to lose that match (and attempted to do so).

In 2016, the tournament was redesigned, with the stated goal of eliminating misaligned incentives; we show the redesign failed in this goal. We then describe a minimally-manipulable tournament rule which could be reasonably implemented, while maintaining many of the subtler features of the current tournament that a designer would want. Proving results about the manipulability of our design requires extending definitions to a setting where teams may play each other more than once with different results. Finally, we demonstrate that explicitly considering cases with small $n$ can lead to different, practical results than would be predicted by only considering the case of arbitrarily large $n$.

## *4.1 Introduction*

The impetus for this work (and much of the recent tournament design work) was dramatic evidence that a commonly-used tournament design does not have properly-aligned incentives. At the 2012 Summer Olympics, in two badminton matches both teams tried to lose the match they were playing. The bizarre incident generated significant news coverage, as the teams were actually attempting to increase their chances of winning a gold medal [Bel]. The tournament organizers disqualified all of the competitors for "not using one's best efforts

to win a match" and "conducting oneself in a manner that is clearly abusive or detrimental to the sport" [Kel]. The decision was widely (though not unanimously) criticized by sports commentators, who placed the blame on the tournament design (producing incentives incompatible with attempting to win every match) rather than a supposed moral failing of the players [Bor, Lei]. Regardless of the blame, the way to move forward is clear – it is the job of the tournament designer to align incentives.

As a result of the furor, the design was changed for the 2016 Olympics, with the goal of ensuring the incident would not be repeated. Indeed, on announcing the change, the president of the Badminton World Federation said the redesign "will eliminate any player's thoughts about actively trying to lose a match or matches, irrespective of other match results" and that they had "ensure[d] such a regrettable spectacle is never witnessed in badminton again" [Ber].

This goal was not met. The redesigned tournament still suffers from at least two different scenarios which could incentivize competitors to lose matches, as we show in subsection 4.2.3. The main contributions of this chapter are to demonstrate this fact and to suggest an alternative tournament which (under some weak assumptions) provably will never incentivize a team to lose a match, while still maintaining many of the subtler tournament features that the Olympic designers may have considered when choosing their designs.

### *4.1.1 Technical Background and Related Work*

This chapter is part of a line of work that has sought to design tournaments that maintain reasonable methods of choosing a winner while minimizing the ability of players to manipulate the results. The fundamental object of these papers is a **tournament rule**. A tournament rule is a function that maps the results of $n$ teams playing all possible pairwise matches to a (possibly randomized) champion of the competition.[1]

Previous work has shown tradeoffs between the reasonableness of a tournament and its

[1] In subsection 4.1.3 we will extend these definitions to a more general setting where teams may play each other more than once.

vulnerability to manipulation. We begin with a recap of the most common criteria for a good tournament.

**Definition 4.1.1** (Condorcet-Consistency)**.** *A tournament rule is* ***Condorcet-consistent*** *if a team which wins all of its matches is declared the champion with probability* 1*.*

Let $T$ be a tournament, i.e., the results for all matches played. For a participant $i$, let $r_i(T)$ be the probability[2] that $i$ becomes the champion under the events of $T$.

**Definition 4.1.2.** *A tournament rule is* ***monotone*** *if for every player $i$, and every tournament $T$, if $i$ intentionally losing a match would result in the tournament $T'$ then $r_i(T) \geq r_i(T')$.*

That is, a participant cannot increase their chances of winning the tournament by losing.

**Definition 4.1.3** (SNM)**.** *A rule is said to be $k$* ***strongly non-manipulable with parameter*** *$\alpha$ ($k$-SNM-$\alpha$) if no coalition of $k$ players can increase their combined probability of winning the tournament by $\alpha$ by manipulating the results of the games between them.*

Altman and Kleinberg show it is impossible for a tournament to be strongly non-manipulable (i.e. $k$-SNM-$\alpha$ for all $k$ and $\alpha$) and Condorcet-consistent [AK10]. With this impossibility result, it is reasonable to try to weaken one of the two criteria. One possible weakening of Condorcet-consistency is **non-imposition**, which requires that for each team there is a set of results (across all matches) that will result in it becoming the champion. As long as the tournament does not have exactly 3 competitors, Altman, Procaccia, and Tennenholtz show there exists a tournament rule that is pairwise non-manipulable (i.e., 2-SNM-$\alpha$ for all $\alpha$), non-imposing, and monotone [APT09]. However, this tournament design is not Condorcet-consistent, which makes it unsuitable for use in sports tournaments.

Instead of weakening Condorcet-consistency, one can weaken non-manipulability. Schneider, Schvartzman, and Weinberg show that a tournament can be Condorcet-consistent and

[2]probability over any randomness in the tournament rule.

2-SNM-1/3 (and that 1/3 is the best possible $\alpha$ for a Condorcet-consistent tournament) [SSW17].

A common pattern in real-world tournaments is to involve multiple stages. A very common tournament design (used in Olympic badminton as well as other Olympic sports and the soccer World Cup) begins with a "round robin" stage, where teams are partitioned into groups of 4 and play every other team in their group. The first and second place teams of each pool advance to the next stage. This multi-stage process can lead to manipulation as demonstrated by the 2012 Olympics badminton scandal [Kel12]. Theoretical work has also analyzed multi-stage tournaments. Pauly proves it is impossible, under a number of reasonable constraints, to design a monotone tournament where the first round is pool play with multiple teams advancing [Pau14]. One of the constraints Pauly imposes is that the tournament be completely deterministic, so this result does not directly address the 2016 redesign. Using a different notion of incentive-compatibility, Vong shows that any combination of group-play stages is incentive-compatible under their model if and only if each group allows only one team to advance [Von17].[3] Neither of these results applies directly to the redesigned Olympic badminton tournament.

In addition to the more theoretical work above, there is a more recent line or work discovering vulnerabilities in other real-world tournaments. Csato has found potential manipulations in both the 2018 European World Cup qualification [Csa17a] and the European Men's Handball Championship [Csa17b]. Csato also proposes practical modifications to these tournaments, but the modifications are not sufficient for the badminton case [Csa18].[4]

---

[3]Vong assumes that all results (if played with full-effort) are known to competitors in advance. This assumption is not particularly realistic, and allows for a very restrictive definition of IC (intuitively they require that every time teams are "ranked" [to move from one stage to the next] that the true ranking is a Nash equilibrium). Vong's result does not directly address randomized designs, nor any uncertainty in the result.

[4]Csato's work considers teams manipulating group play to affect whether they advance out of group play. The primary issue in the badminton context is not whether the manipulating team advances, but whether they improve their chances in the next round.

### *4.1.2 Our Contribution*

This paper is meant to be a bridge between the two lines of work above. The theoretical work has produced very interesting impossibility (and occasionally possibility) results, but often focuses on results that hold for all $n$. Meanwhile the applied work shows that real-world tournament designers have not actually applied these results, and continue to use non-monotone tournaments. Our aim is to find a design which is as easy to use as possible for real-world designers, while still provably having the incentive-compatibility properties theoreticians know are necessary.

Our contributions are as follows: first we demonstrate that the changes made in 2016 were insufficient. Indeed, the redesigned tournament is still vulnerable to at least two different scenarios which would incentivize a team to lose. Second we design a tournament which is monotone (and therefore not vulnerable to these kinds of manipulation). We are not the first to design such a tournament, but we argue (in subsection 4.3.3) that our design has practical advantages that make it more likely to be utilized than previous suggestions.

On a technical level, we have the following contributions. We describe a new mathematical model for tournaments that allows teams to play each other more than once with different results, (which we have not found in the literature, despite many important tournaments having this feature) and thus extend definitions to this new setting. All prior proofs of monotonicity have been trivial. A key feature of our design is that teams are not eliminated on their first loss, we thus require a non-trivial proof of monotonicity. While our proof is not complicated, we believe this is a key step to designing tournaments that designers will use in the real-world (as we argue in subsection 4.3.3). Finally, our work demonstrates the importance of explicitly considering small $n$. Prior theoretical work often proves results for general $n$, we will see that our tournament design has a subroutine which is not incentive compatible for arbitrary $n$, but is for $n = 4$, and that this is sufficient for our real-world needs.

### *4.1.3 Model and other Preliminaries*

We consider two models for the results of games:

1. Whenever two teams play (intending to win) the same team always wins.

2. Whenever teams $i$ and $j$ play (intending to win) an independent Bernoulli random variable with parameter $p_{ij}$ is drawn to decide who wins. The parameter $p_{ij}$ may differ between pairs, but does not change if $i$ and $j$ repeat a matchup.

Model 1 is commonly used in the literature, but we will see (e.g. in Theorem 4.3.1) that it is insufficient for tournaments where teams are likely to meet more than once. We therefore introduce Model 2. Our model is still a simplification of the real-world (for example, teams may learn about each other in their first match against each other and make adjustments for their rematch, altering the probability of winning), but our assumptions are fairly minimal. More practically, recall that our goal is to understand whether teams would intentionally lose games, when those teams know that previous teams that did so were disqualified from the tournament – our model does not need to be perfect, just accurate enough that the threat of disqualification will overwhelm any lower-order effects we have not considered.

We will need the following basic tournament designs repeatedly, so we introduce them here: a **round robin** tournament involves every team playing every other team once, and ranking the teams based on the number of wins. This design is often used as a subtournament, with teams divided into groups and playing the round robin design only within their groups, with some teams eliminated and others advancing to the next stage of the tournament. A **single elimination tournament** (or SET), sometimes called a "knockout" tournament, is a tournament on $2^i$ teams. At each stage, the remaining teams are paired and each plays one game; the teams that win advance to the next round and the losers are eliminated. One can view this tournament as a binary tree, where the leaves are the teams, and each internal node is a match to be played (between the teams in its child nodes).

## *4.2 2012 Badminton Incident and Olympics Response*

In this section we discuss: the 2012 incident, the modifications made for the 2016 tournament, and show that the modifications were insufficient to actually make the tournament monotone.

### *4.2.1 2012 Badminton Incident*

The 2012 tournament utilized a common group-play-then-knockout format. The competitors were divided into "groups" of four teams each. Each group plays a round robin tournament, where the top two teams in each group[5] advance to a "knockout" round (i.e. a single elimination tournament). In the 2012 games, the position of teams in the SET was determined by finish in the groups (and known in advance to the competitors). Entering the final matches of group play, prior results led to the scenario in Figure 4.1. The final match of Group $\mathcal{A}$ (between $X$ and $Y$) is the one we will study, which both teams attempted to lose.

Notice that the mapping of teams to their location in the knockout bracket is fixed by the results. The final match in Group $\mathcal{A}$ was between teams which were guaranteed to advance. The only stakes were who the teams would play in the knockout. Moreover, because the matches were played at different times, the identities of $D1$ and $D2$ (the teams advancing from group $\mathcal{D}$) were known as well. Based on known team strengths, the most likely results of games among some teams in the knockout are those shown in Figure 4.1.

Looking at the bracket, (and assuming that C1 and C2 will be fairly comparable) the incentives for $X$ and $Y$ are clear, but bizarre. The goal of each team would be to avoid team $D2$ as long as possible (there is a small chance of an upset in any game; the further from $D2$ in the bracket, the better the chances of an upset happening before having to play them).[6] Thus the teams would prefer to be $A2$, and thus want to lose their final match.

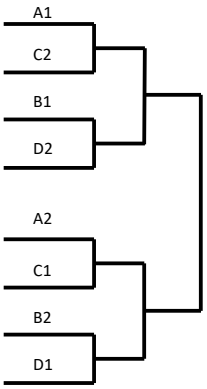
---

[5]That is the two who won the most games, with ties broken by measures of margin of victory.

[6]Under normal circumstances, the seeding was supposed to incentivize winning the group, but an upset in Group $\mathcal{D}$ caused the best team in $\mathcal{D}$ to be seeded as the second place team, and reverse the incentives.

Group $\mathcal{A}$

| | |
|---|---|
| $X$ | 2-0 |
| $Y$ | 2-0 |
| Team 3 | 1-2 |
| Team 4 | 0-3 |



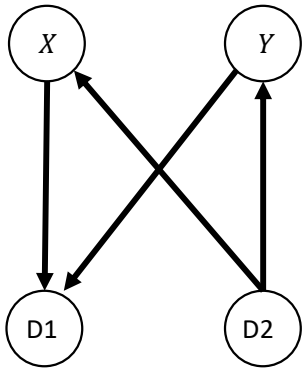




Figure 4.1: Left: The state of Group $\mathcal{A}$ before $X$ and $Y$ played their match. Middle: The assignments of teams that advance from the groups to the positions in the knockout. Right: The likely results of matches between $X, Y$ and the teams advancing from Group $\mathcal{D}$, an edge from $u$ to $v$ means it would be expected that $u$ wins when playing $v$.

### *4.2.2 Olympics Response*

The Badminton World Federation (BWF) responded to the 2012 scandal by altering the tournament. The change adopted for 2016 was to randomize the draw for the knockout stage with the stated goal that the secondary draw would "eliminate any player's thoughts about actively trying to lose a match or matches, irrespective of other match results" [Ber].

Below, we give a formal mathematical statement of the tournament design used in 2016, and show that this game format does not solve the problems in the old game format in that it is not monotone. The official description of the rules for 2016 and 2020 can be found at [dra, Section 5.4.1.2].[7]

The competition is played in two stages— a group play stage followed by a knockout stage.

[7]There is at least one potential ambiguity in the official statement – exactly how far away teams from the same group must be placed. The actual draws produced in 2016 makes the interpretation we give the only possible one.

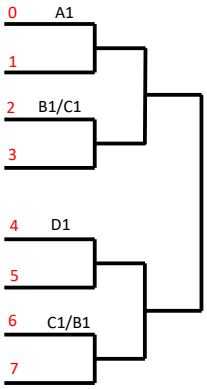



Figure 4.2: The knockout seeding for 2016 and 2012.

During group play, we maintain a scoreboard allowing us to order the teams according to their performance within the groups. To transition to the knockout stage, we pick two top teams from each group's ordering. Call these teams $\{A1, A2, B1, B2, C1, C2, D1, D2\}$, where the letters indicate group and the number indicates their finish.

To define the knockout stage, number the positions of the bracket $0, 1, \ldots, 7$ from top to bottom (see Figure 4.2).

Place $A1$ in position 0 and $B1$ in position 4. Now place $C1$ and $D1$ into positions 2 and 6, deciding which goes where uniformly at random. Finally, to place $A2, B2, C2, D2$ in the remaining spots: choose an assignment uniformly at random from among those which have no teams from the same group playing each other in the first round.

The winner of knockout stage is declared the winner of the competition. This finishes the definition of the new Olympic format.

### *4.2.3 Flaws in the format*

We now show that the redesign failed, and that the 2016 design is still not monotone, and indeed, that a repeat of the 2012 incident is a real possibility. We begin with a description of two known methods of manipulation, and show that each is still possible. The heart of

each of these examples is not novel – both are adapted from examples in Pauly's paper on the original incident [Pau14].

Pauly's results do not apply directly to the redesign (in particular, all designs discussed in the paper are deterministic). We have not seen any commentary in the literature on the redesign, so to the best of our knowledge, this observation is novel.

We use the notation $\phi^{(\mathcal{X})}$ to represent the ordered pair of teams advancing from group $\mathcal{X}$. In the first type of manipulation, a team can alter which team advances with them into the knockout stage of the tournament in order to improve their own chances of winning.

**Example 4.2.1.** *We show that a team ($A$) can manipulate who advances out of their group to increase their chances of avoiding a superior team ($X$) in the knockout. Let $A, B, C, D$ be in group $\mathcal{A}$ and $X$ be in group $\mathcal{D}$. Consider a tournament with the following relationships.*

- *$A$ beats all other teams (including teams in other groups) except $X$*
- *$X$ beats all other teams except $C$*
- *$C$ beats $D$, $X$, and all teams that advance to the knockout except $A$*
- *$B$ beats $C$*
- *$D$ beats $B$*

*Consider Group $\mathcal{A}$ with players $A, B, C, D$. If all teams play at full-strength, $A$ wins the group, and a tie-breaking rule will choose the second-place team. Suppose that rule picks $B$. Team $X$ will win group $\mathcal{D}$, so by the relationships above, $X$ and $A$ will advance to the finals, where $X$ becomes champion.*

*Now suppose $A$ throws its game to $C$ so that after group play. $\phi^{\mathcal{A}} = (A, C)$.*[8] *$A$ and $X$ are now forced to be on opposite sides of the bracket. The second-place teams will be placed*

[8] A tie-breaker will decide the rankings of $A$ and $C$ relative to each other. We assume $A$ can manipulate the results to win the tie-breaker as well. Since the tiebreakers are various ways of measuring margin of victory, $A$ will be able to manage this goal if it wins its first two matches by a sufficient margin.

*so that teams from the same group do not play in the first round. With probability* $2/3$*,* $C$ *will be placed in the bracket such that it will meet* $X$ *before the finals, allowing* $A$ *to become champion. Thus,* $A$ *can improve its probability of becoming champion by* $2/3$ *by intentionally losing a match.*

In the second type of manipulation, a team can intentionally qualify second in the group in order to improve their probability of winning the competition.

**Example 4.2.2.** *We show team* $A$ *can manipulate seeding to increase its chances of becoming champion. Suppose* $A$ *would win all matches in group* $\mathcal{A}$ *if playing at full strength. We use* $\phi^{(\mathcal{X})}$ *to denote the (ordered) pair of teams advancing out of group* $\mathcal{X}$*. Suppose that* $\phi^{(\mathcal{B})} = (C, D), \phi^{(\mathcal{C})} = (E, F)$*, and* $\phi^{(\mathcal{D})} = (G, H)$*. Group* $\mathcal{A}$ *has one game left between* $A$ *and* $B$ *such that if* $A$ *wins,* $\phi^{(\mathcal{A})} = (A, B)$*; otherwise* $\phi^{(\mathcal{A})} = (B, A)$*. Now consider a tournament graph with the following characteristics:*

- $A$ *beats* $\{B, C, E, G\}$ *and loses to* $\{D, F, H\}$*,*
- $B$ *beats* $\{D, F, H\}$*,*
- *For* $i \in \{\mathcal{B}, \mathcal{C}, \mathcal{D}\}$ *and* $j \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}\}$*,* $\phi_1^{(i)}$ *beats* $\phi_2^{(j)}$*,*
- *All other teams lose to the qualifying teams for knockout phase.*

*If* $A$ *decides to win against* $B$ *in the above mentioned game, thereby qualifying as* $\phi^{(\mathcal{A})} = (A, B)$*, then by the seeding rules,* $A$ *would play one of* $\{D, F, H\}$ *in the first round of the knockout causing* $A$ *to be eliminated. On the other hand, if* $A$ *threw the match in question, then it qualifies as* $\phi^{(\mathcal{A})} = (B, A)$*.* $A$ *will play one of* $\{C, E, G\}$ *in the first round of the knockout and win. Furthermore, all of* $\{D, F, H\}$ *will be eliminated after the first round since they will be playing a player that can beat them. This implies that* $A$ *can beat any of the teams remaining and will be the winner of the tournament.*

We have made the situations above quite extreme to make them as simple as possible to understand. However, these examples are quite robust to small changes in the design of the tournament or in the results. For example, a common practical technique to minimize the chances of these incidents is to have the final matches for all groups start at the same time (the World Cup utilizes this technique, for example). The examples our robust to this change – in Example 4.2.1, if $X$ somehow comes in second in its group, $A$ still benefits by bringing $C$ to the next round (though it is less likely that $C$ will meet $X$ in time). In Example 4.2.2, $A$ benefits as long as the majority of the groups come out as expected.

Extreme setups are not required to make a tournament non-monotone. Indeed, consider the exact scenario from 2012, with an upset happening in group $\mathcal{D}$, and the members of group $\mathcal{A}$ (each with 2 wins and no losses) playing only to determine ranking. They will both still want to lose. Note that the bracketing process does not treat groups equivalently – $A1$ and $D1$ are forced to be on opposite sides of the bracket. An $A1$ vs. $D2$ matchup would happen in each round of the bracket with probability 1/3. On the other hand, $A2$ cannot meet $D2$ in the first round, and meets it (or the team that beat it) in the second round with probability only 1/6. The advantage has been slightly decreased (and made harder for non-experts to recognize), but in the *exact* situation from 2012, there is still significant incentive for teams to lose. The non-monotonicity of this tournament is not an esoteric matter only for theoreticians, it is only a matter of time before teams are incentivized to lose again.

## *4.3 Double-Elimination Design*

Having shown the first redesign failed, we now turn to how the BWF should redesign their tournament. In this section we describe our alternative design, and prove that it has the desired features. We will cover the practical benefits of this tournament in detail in subsection 4.3.3, but it is worth briefly describing why we felt the need to go beyond the designs already in the literature. A (randomized) single elimination tournament (that is, making the entire tournament a "knockout" phase) is known to be Condorcet-consistent, monotone, and minimally manipulable (in the 2-SNM sense) [SSW17]. Despite these wonderful properties,

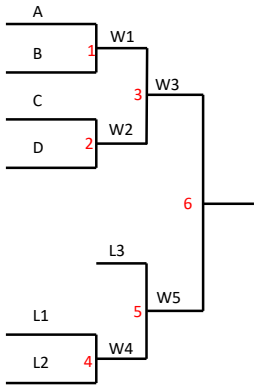



Figure 4.3: The bracket for a 4-team double elimination tournament. Games are numbered, in red. The winner of game $i$ is denoted by W$i$ and the loser by L$i$. Games 1,2, & 3 are the "winners' bracket", 4 & 5 are the "losers' bracket", and 6 is the "final"

the BWF did not choose this design when they remade the tournament (and this decision could not have been from ignorance of the design – a single elimination tournament was used prior to 2012). Instead the designers chose to modify the two-stage system and preserve group play. They must have seen a tremendous benefit to the group play to preserve it. The primary benefit of group play is the guarantee of multiple matches for each team. In a single elimination tournament, half of the teams play only one match (as they lose their first and are immediately eliminated). With group play, every team is guaranteed three matches.[9]

We propose replacing the round robin competition in group play with a different tournament structure (still played in groups of 4). Our design is an altered knockout that adds a second "losers' bracket."

A 4-team Double-Elimination Tournament (DET) has the following format (see Figure 4.3).

In the first round, team A plays team B, and C plays D. In the second-round the winners

[9]Though a team may be de facto eliminated after two matches, it will still play its third match (consider, for example, teams 3 and 4 in Figure 4.1 after each losing to $X$ and $Y$); every team is still "alive" after their first match, though they may need games they do not play in to have a certain result to advance.

of round one play each other, and the losers play each other. After these games, some team has lost two games and is eliminated, another has won two games and advances to the finals. The two remaining teams (which have won one match and lost one match) play each other in the third round, with the winner advancing to the finals and the loser being eliminated. In the finals, the two remaining teams play each other a single time, with the winner being the champion (regardless of who won if they played previously).[10]

Our overall tournament design is the following:[11]

**Olympic Tournament Design**

Phase 1: Divide the teams into groups of 4 (arbitrarily), and have each group perform a 4-team DET. The winner of each DET advances to the next phase.

Phase 2: Place the DET winners (arbitrarily, as long as placement does not depend on the results of Phase 1) into a knockout bracket. The winner of the knockout bracket is the champion.

### *4.3.1 Models*

Recall we have two models for the outcomes of matches:

1. Whenever two teams play (intending to win) the same team always wins.

2. Whenever two teams play (intending to win) an independent random variable is drawn to decide who wins. The distributions may differ between pairs, but is the same distribution every time a given pair plays.

We start by showing why Model 1 is insufficient for understanding this tournament.

---

[10]A more common version of a double elimination tournament has the teams in the final play up to two games, with a team being eliminated when it gets its second loss overall in the tournament. This design is not monotone in general.

[11]In our design, we have tried to follow the current Olympic format as closely as possible, while ensuring monotonicity. A more "obvious" use of our design would be using 4-team DET recursively; this would be monotone, but requires more games than our proposed approach.

**Theorem 4.3.1.** *The outcome of a 4-team-DET is equivalent to the outcome of a 4-team-SET under Model 1.*

*Proof.* Let $A, B, C, D$ be the four teams under Model 1. Let $x, y \in \{A, B, C, D\}$, be the teams that meet in the final round of the DET, where $x$ and $y$ come from the winners' bracket and losers' bracket respectively.

Note that the winners' bracket of the DET (until the final round) has the same match-ups as the SET. That is, the winner of the SET is $x$. We want to show $x$ is also the DET's winner. Observe that it suffices to show $x$ and $y$ already met in the winners' bracket. Indeed, in that case, $y$ must have lost to $x$. Thus, $x$ will win again in the final and become the champion.

Suppose, for contradiction, that $x$ and $y$ have not met before the final round. Then $y$ must have lost to some other team, $z$, in the first round. Since $x$ advances to the final round, $z$ must have entered the losers' bracket, but this leads to a contradiction as $z$ would have eliminated $y$ in game 5.

Thus $x$ is DET's winner as well. □

In the real-world a double-elimination tournament may have a different outcome than a single-elimination tournament (such tournaments are used in practice for exactly this reason), so we need to introduce Model 2.

It will suffice to prove our results in Model 2, as Model 1 is just the special case where each match is won by some team with probability 1.

### *4.3.2 Manipulability Results*

We can now turn to proving our main result. Observe that to show monotonicity, it suffices to argue that for every game, any team would prefer its situation after winning to its situation after losing (since the probability of becoming champion when playing at full-strength is a convex combination of the probability of becoming champion after winning or losing).

We start with the subtournaments. The key intuition for this proof is that a loss will either eliminate a team or force them to win a superset of matches they would have played

had they won.

**Theorem 4.3.2.** *Under Model 2, a 4-team DET is monotone.*

*Proof.* Let $A, B, C, D$ be the four teams in the tournament. By symmetry it suffices to show monotonicity with respect to $A$.

Clearly, there is no benefit to losing if $A$ is already in the loser's bracket or in the finals (as a loss causes the probability of winning the tournament to be 0). Thus it suffices to consider whether $A$ wants to lose in the first game (Game 1 in Figure 4.3) or in the second round of the winners' bracket (Game 3).

First we consider losing in Game 3. Without loss of generality, suppose $A$'s opponent in Game 3 is $C$ and $B$ advanced out of Game 4 (the first game in the losers' bracket). If $A$ loses, then it must beat both $B$ and $C$ to win the tournament. On the other hand if $A$ wins, it must beat only one of $B$ or $C$ to win the tournament. Thus it is no harder for $A$ to win if it advances to the finals.

We now consider the first game of the tournament. For any teams $x, y$ let $p_{xy}$ be the probability that $x$ beats $y$ when they both play at full strength.

If $A$ loses the first game, it will have to beat every other team in turn to become champion, which occurs with probability

$$p_{AB} \cdot p_{AC} \cdot p_{AD}.$$

If $A$ actually wins the first game, its probability of becoming champion is

$$p_{AC}p_{Ax} + (1 - p_{AC})p_{Ay}p_{AC},$$

where $x \in \{B, C, D\}$ is the team that advances out of the loser's bracket if $A$ wins game 3 and $y \in \{B, D\}$ is the team that won game 4.

It suffices to show that

$$p_{AB} \cdot p_{AC} \cdot p_{AD} \leq p_{AC}p_{Ax} + (1 - p_{AC})p_{Ay}p_{AC}.$$

Dividing through by $p_{AC}$ (if $p_{AC}$ is 0, $A$ has no chance of becoming champion, so this division is well-defined), it suffices to show

$$p_{AB} \cdot p_{AD} \leq p_{Ax} + (1 - p_{AC})p_{Ay}.$$

If $x$ is $B$ or $D$ then $p_{Ax}$ alone is at least $p_{AB}p_{AD}$. Otherwise, $x$ is $C$, and we have:

$$p_{Ax} + (1 - p_{AC})p_{Ay} = p_{AC} + (1 - p_{AC})p_{Ay} \geq p_{AC}p_{Ay} + (1 - p_{AC})p_{Ay} = p_{Ay} \geq p_{AB}p_{AD}$$

as required. □

We get our main result almost immediately:

**Theorem 4.3.3.** *The Olympic Tournament Design is monotone.*

*Proof.* Phase 1 is monotone by Theorem 4.3.2. Phase 2 is inherently monotone since losing a match results in elimination. Finally Phase 1 and Phase 2 are completely independent by construction. □

It is worth contrasting Theorem 4.3.2 with a result of Stanton and Williams. They define a double elimination tournament on arbitrarily many teams and show the following theorem.

**Theorem 4.3.4** ([SW13, Section 4.2])**.** *Under Model 1, a 16-team DET is not monotone.*

We note that there is no contradiction between these results – intuitively, a large double elimination tournament is not monotone, because by strategically entering the losers' bracket one may be able to avoid meeting a superior opponent in the winners' bracket and furthermore avoid them in the losers' bracket as well (before hopefully advancing to the finals). A four-team DET has no such possibility: there are simply not enough teams available to avoid facing a team by going into the losers' bracket.

We now briefly measure the quality of the Olympic Design with respect to other measures in the literature. The most common alternative measure of strategy-proofness is 2-SNM-$\alpha$.

Recall that this is a measurement of how significantly a pair of teams can increase their combined probability (i.e. the sum of their individual probabilities) of becoming champion by altering the results of the matches they play against each other. Consistent with previous work on the topic, when considering randomized designs, we fix the pair which will attempt to collude before revealing the randomness.

We call a 4-team DET "randomized" if the identities of $A, B, C, D$ are selected uniformly at random before the start of the tournament (define a "randomized-SET" analogously).

**Theorem 4.3.5.** *Under Model 1, a randomized 4-team DET is 2-SNM-*$1/3$.

*Proof.* Observe that there are 3 possible arrangements with equal probability: $A$ could play any of $B, C$, or $D$ in the first round.

It suffices to show that if two players decide to collude, they cannot improve their combined chance in arrangements where they do not play each other in the first round. Since there are two such arrangements out of three for each pair, this shall prove the theorem.

Consider the arrangements where $A$ is not playing $B$ in the first round. We show that there is no game in which they can alter the result to their combined benefit. First, note that if $A$ and $B$ both end up in the losers bracket at the end of the first round, then collusion does not benefit them. There is some team that can beat $A$ and a different team that can beat $B$. But $A$ and $B$ would have to play these opponents again before they can win the tournament, so the combined probability of winning is zero with (or without) collusion.

With the exclusion of the above case, $A$ and $B$ can meet each other only in Game 3 or in the final. Regardless of the result of Game 3, another team wins the tournament only if they beat $A$ and $B$ once each (Game 3 only decides the order in which those games are played). Thus manipulating the result does not increase their probability of winning. Finally, manipulation in the final round is useless as one of them becomes champion regardless.

It follows from the above argument that $A$ and $B$ cannot improve their combined chances by manipulating results unless they meet in the first round. This matchup occurs with probability at most $1/3$, giving the bound. □

Note that 1/3 is the optimal $\alpha$ for size-2 coalitions among Condorcet-consistent tournaments (under Model 1) [SSW17].

Combining with known results, we get that a randomized version of our Olympic Design is optimal under Model 1.

**Theorem 4.3.6.** *Under Model 1, a randomized Olympic DET Design is* 2*-SNM-*1/3

*Proof.* Phase 1 is 2-SNM-1/3 by Theorem 4.3.5. Phase 2 is 2-SNM-1/3 by [SSW17, Theorem 3.3]. Since teams cannot play each other in both phases, the tournament overall is 2-SNM-1/3. □

Recall that Model 1 is not the best way to understand the how this tournament works in the real-world (based on Theorem 4.3.1, for example), so we do not interpret this theorem to mean there is no better tournament is possible. Indeed, under Model 2, we do not believe the tournament is 2-SNM-1/3 (in Phase 1 both teams being in the losers' bracket is not a guarantee that the teams cannot win the tournament). Regardless, we do not believe this is a practical threat to the tournament. It seems unlikely that Olympic athletes would form coalitions, except between teams from the same country. The results above still holds if the initial draw prevents teams from the same country from being placed in the same group (as is currently required [dra, Rule 4.1]), and this would prevent collusion until at least Phase 2.

### *4.3.3 Practical Advantages*

In addition to the theoretical benefits mentioned above, our proposed design offers several practical benefits over its counterparts.

Firstly, recall from the beginning of section 4.3 that the BWF declined to use a single elimination tournament (despite it being Condorcet-consistent, monotone and 2-SNM-1/3), because most competitors are eliminated after relatively few games. The double-elimination phase in our tournament design ensures that each team plays at least two games before it is eliminated, while maintaining Condorcet-consistency and monotonicity (and 2-SNM-1/3 under Model 1). While not matching the three of standard group play, note that the only

teams that do not play at least three matches are the ones that lose their first two matches. In the current tournament, teams which have lost their first two matches are usually de facto eliminated anyway,[12] so the loss is minimal.

We also compare our design to a modification of 2016 Olympic format where the group play phase is maintained, but we allow only one team to advance per group. Such a modification to Olympic format should make it immune to the flaws we highlighted in subsection 4.1.3 but has drawbacks in practice. The group play phase suffers from the possibility of tied teams (requiring a tie-breaker to choose the team that advances). Indeed, in the 2012 Olympics badminton Women's doubles tournament the two groups that did not have teams disqualified each had a 3-team tie for first. In 2016, one group similarly ended in a 3-way tie. Tie-breakers induce two difficulties. First, a three-way tie for first will create a team which (a) is eliminated with only one loss and (b) defeated the team that advanced (perhaps even the eventual champion). Such a team has a potential argument that it "should have been the champion" (since it performed no worse than the eventual champion in group play, and actually beat them head-to-head). Our design ensures that the team that advances won at least half of the games against each opponent it played, eliminating that sort of argument. Second, the schedule may give an inherent advantage to one of the teams in the tie-breaker. When these ties occur, by the start of the third-match one of the three teams is de facto eliminated (regardless of how it plays in that match, it could not advance). In those instances, one should be concerned that the team will "give up" and not perform as well. In that scenario, the team that plays them last could benefit in the tie-breaker from a large margin-of-victory.

In addition to avoiding the problems above, we note a few benefits of our design. A benefit of the 2012 and 2016 designs is that they can forgive an early loss. A team could lose an early match (perhaps on a fluke or due to start-of-tournament jitters) and still have

[12]By which we mean, they cannot win the tournament regardless of the outcomes of the remaining matches. In the women's 2012 and 2016 tournaments, 12 teams lost their first two matches. Only two were not de facto eliminated, and 8 of the 12 were playing in a match against another de facto eliminated team.

a chance to advance. Our tournament increases forgiveness to very early losses. In regular group play, a team with one loss must rely on the results of matches between other teams to advance (to avoid being eliminated in a potential three-way tie for first). For double-elimination group play, after a loss in the first two matches teams still "control their own destiny" (as long as they continue to win, they will become champion, regardless of other matches).

As a final practical note, the new design requires no more matches than the current one. Indeed, (a subset of) the same schedule can even be used: Games 1 and 2 take the place of the first matches of group play, 3 and 4 in place of the second matches. Game 5 takes the place of the third matches of group play. Game 6 takes the place of the quarterfinals of the knockout.

### *4.4 Conclusion*

We have shown that, despite assertions to the contrary, the 2016 and 2020 Olympic Badminton tournaments are vulnerable to manipulation – indeed were the exact situation from 2012 to arise, the teams would have significant incentive to lose. We provide a potential alternative that (under some assumptions) provably has the best-possible strategy-proofness, while maintaining every team playing at least two games and ensuring that an early fluke loss will not eliminate a team. Moreover, the proof (or at least its intuition) can be easily explained to the athletes: if you lose, you have to win all the games you would have otherwise and an additional one. This ensures not just that they will not benefit from losing, but also that they will not *think* they will benefit from losing.

We leave open whether there are better replacements for group play. In particular, a tournament that forgave a first-loss (regardless of when it occurred) or one which had better SNM properties would be of interest. Additionally, our format also does not allow for draws (like the FIFA World Cup group play does). It would also be interesting to try to extend these constructions to work with other numbers of teams – 3 team and 6 team groups are used in other competitions. A 3-team version (or alternative), would be of particular interest,

since the FIFA world Cup plans to transition to 3-team groups for 2026.

More generally, our work shows the importance of considering the exact scenarios practitioners care about: there turn out to be constructions for $n = 4$ which we cannot extend to the arbitrarily large $n$ that theoreticians usually consider. Considering both regimes should be of interest to the theory community and lead to more usable results.

# Chapter 5

# CONCLUSION

We conclude with of a common thread in the work above and how it might apply to future work. In all cases, the key technical insights used long-standing tools; we discuss these tools again and whether they might be useful to further progress on these topics.

In Chapter 2, the key technical insight came from an analysis of the rotation poset, a fundamental object in the study of stable matchings known since the 1980s. The rotation poset is a powerful tool for understanding stable matchings, and it is doubtful we have fully exploited its description in bounding the number of stable matchings. It is very likely additional insight and a more careful argument could substantially reduce $C$ from its current value of $2^{17}$.

In Chapter 3, the key tool was hierarchically separated trees, a tool developed in the 1990s. In this case, the tool has been used to its fullest extent in this problem – there are metric spaces which induce $\Omega(\log n)$ distortion, so a new tool would be required to break this barrier. There is good reason to believe that breaking this barrier is not possible at all; there is a lower bound on randomized algorithms that nearly matches the performance of the best known algorithms (see section 3.3.1). Closing this gap remains open. Combinatorial tools may well be suitable for generating a new lower bound, but significant care would be required to improve over the current bounds by the necessary $\log \log n$ factor.

In Chapter 4, we needed simply the definitions of the problem and axioms of probability. The obvious open problems in this area is to adapt our group play design to handle different numbers of teams and allowing matches to end in draws. The literature on this problem so far has only used combinatorial tools – they will likely continue to be well-suited to these problems, but we will need new insights to really extend these results to answer the open

questions.

The age of the tools used did not prevent us from making progress: Chapter 2 makes significant progress on a problem open since the 1970s, while Chapter 4 includes (to the best of our knowledge) the first non-trivial proof of monotonicity of a tournament. Meanwhile, in Chapter 3, we are able to connect ideas from two previously-separate lines of literature on closely related problems.

Of course, we do not mean to suggest that more modern tools are not incredibly useful. We simply observe that classic tools can still be employed both to make progress and to make connections in the literature, and are likely to be useful in further progress on these problems.

# BIBLIOGRAPHY

[AAC+17] Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 81. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[ACK17] Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1051–1061, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.

[AJF18] Yossi Azar and Amit Jacob-Fanani. Deterministic min-cost matching with delays. *arXiv preprint arXiv:1806.03708*, 2018.

[AK10] Alon Altman and Robert Kleinberg. Nonmanipulable randomized tournament selections. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[AKL17] Itai Ashlagi, Yash Kanoria, and Jacob D Leshno. Unbalanced random matching markets: The stark effect of competition. *Journal of Political Economy*, 125(1):69–98, 2017.

[APT09] Alon Altman, Ariel D Procaccia, and Moshe Tennenholtz. Nonmanipulable selections from a tournament. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 184–193. IEEE, 1996.

[BBMN15] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph (Seffi) Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5):40:1–40:49, November 2015.

[BCK95] Arthur T Benjamin, Cherlyn Converse, and Henry A Krieger. How do I marry thee? Let me count the ways. *Discrete Applied Mathematics*, 59(3):285–292, 1995.

[Bel] Ken Belson. Olympic ideal takes beating in badminton. The New York Times.

[Ber] Howard Berkes. Badminton takes swing at avoiding repeat of london scandal. National Public Radio.

[BGR08] Nayantara Bhatnagar, Sam Greenberg, and Dana Randall. Sampling stable marriages: why spouse-swapping won't work. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1223–1232. Society for Industrial and Applied Mathematics, 2008.

[BKLS18] Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Paweł Schmidt. A primal-dual online deterministic algorithm for matching with delays. *arXiv preprint arXiv:1804.08097*, 2018.

[BKS17a] Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. A match in time saves nine: Deterministic online matching with delays. In *International Workshop on Approximation and Online Algorithms*, pages 132–146. Springer, 2017.

[BKS17b] Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. A match in time saves nine: Deterministic online matching with delays. *arXiv preprint arXiv:1704.06980*, 2017.

[Bor] Sam Borden. The goal is winning gold, not winning every match. The New York Times.

[CGM12] Prasad Chebolu, Leslie Ann Goldberg, and Russell Martin. The complexity of approximately counting stable matchings. *Theoretical Computer Science*, 437:35–68, 2012.

[Csa17a] László Csató. European qualifiers to the 2018 fifa world cup can be manipulated, 2017.

[Csa17b] László Csató. Overcoming the incentive incompatibility of tournaments with multiple group stages. *arXiv preprint arXiv:1712.04183*, 2017.

[Csa18] László Csató. Incentive compatible designs for tournament qualifiers with round-robin groups and repechage. *arXiv preprint arXiv:1804.04422*, 2018.

[DBŚ13] Ewa Drgas-Burchardt and Zbigniew Świtalski. A number of stable matchings in models of the gale–shapley type. *Discrete Applied Mathematics*, 161(18):2932–2936, 2013.

[DGGJ04] Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004.

[DM10] Brian C Dean and Siddharth Munshi. Faster algorithms for stable allocation problems. *Algorithmica*, 58(1):59–81, 2010.

[dra] Badminton world federation statutes. `https://corporate.bwfbadminton.com/statutes`.

[EKW16] Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 333–344. ACM, 2016.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[GI89] Dan Gusfield and Robert W Irving. *The stable marriage problem: structure and algorithms*. MIT press, 1989.

[GJ12] Leslie Ann Goldberg and Mark Jerrum. Approximating the partition function of the ferromagnetic potts model. *Journal of the ACM (JACM)*, 59(5):25, 2012.

[GS62] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[IL86] Robert W Irving and Paul Leather. The complexity of counting stable marriages. *SIAM Journal on Computing*, 15(3):655–667, 1986.

[IM05] Nicole Immorlica and Mohammad Mahdian. Marriage, honesty, and stability. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 53–62. Society for Industrial and Applied Mathematics, 2005.

[Kel] Paul Kelso. Badminton pairs expelled from london 2012 olympics after 'match-fixing' scandal. The Telegraph.

[Kel12] Paul Kelso. Badminton pairs expelled from london 2012 olympics after matchfixingscandal. *The Telegraph Newspaper, 1 August*, 2012.

[KMP90] Donald E Knuth, Rajeev Motwani, and Boris Pittel. Stable husbands. *Random Structures & Algorithms*, 1(1):1–14, 1990.

[Knu76] Donald Ervin Knuth. *Mariages stables et leurs relations avec d'autres problèmes combinatoires: introduction à l'analyse mathématique des algorithmes*. Montréal: Presses de l'Université de Montréal, 1976.

[Knu97] Donald Ervin Knuth. *Stable marriage and its relation to other combinatorial problems: An introduction to the mathematical analysis of algorithms*, volume 10. American Mathematical Soc., 1997. English translation of *Mariages stables*.

[KOGW18] Anna R Karlin, Shayan Oveis Gharan, and Robbie Weber. A simply exponential upper bound on the maximum number of stable matchings. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 920–925, 2018.

[Lei] John Leicester. Column: Ideals, reality clash at olympic badminton. Associated Press.

[Lev17] Avi Levi. *Novel uses of the Mallows model in coloring and matching*. PhD thesis, University of Washington, 2017.

[LPWW18] Xingwu Liu, Zhida Pan, Yuyi Wang, and Roger Wattenhofer. Impatient online matching. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[Mal57] Colin L Mallows. Non-null ranking models. i. *Biometrika*, 44(1/2):114–130, 1957.

[Man13] David F Manlove. *Algorithmics of matching under preferences*, volume 2. World Scientific, 2013.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[Pau14] Marc Pauly. Can strategizing in round-robin subtournaments be avoided? *Social Choice and Welfare*, 43(1):29–46, 2014.

[Pit89] Boris Pittel. The average number of stable matchings. *SIAM Journal on Discrete Mathematics*, 2(4):530–549, 1989.

[Pit17a] Boris Pittel. On likely solutions of the stable matching problem with unequal numbers of men and women. *arXiv preprint arXiv:1701.08900*, 2017.

[Pit17b] Boris Pittel. On random exchange-stable matchings. *arXiv preprint arXiv:1707.01540*, 2017.

[Pit17c] Boris Pittel. On random stable partitions. *arXiv preprint arXiv:1705.08340*, 2017.

[Ram12] Catherine Rampell. 2 From U.S. win Nobel in economics. *The New York Times*, October 15, 2012.

[Rot15] Alvin E Roth. *Who Gets What–and Why: The New Economics of Matchmaking and Market Design*. Houghton Mifflin Harcourt, 2015.

[RS92] Alvin E Roth and Marilda A Oliveira Sotomayor. *Two-sided matching: A study in game-theoretic modeling and analysis*. Cambridge University Press, 1992.

[RT81] Edward M Reingold and Robert E Tarjan. On a greedy heuristic for complete matching. *SIAM Journal on Computing*, 10(4):676–681, 1981.

[SSW17] Jon Schneider, Ariel Schvartzman, and S Matthew Weinberg. Condorcet-consistent and approximately strategyproof tournament rules. In *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[Sta53] John M Stalnaker. The matching program for intern placement. *Academic Medicine*, 28(11):13–19, 1953.

[Sta11] Georgios K. Stathopoulos. Variants of stable marriage algorithms, complexity and structural properties. Master's thesis, University of Athens, Department of Mathematics, 2011.

[SW13] Isabelle Stanton and Virginia Vassilevska Williams. The structure, efficacy, and manipulation of double-elimination tournaments. *Journal of Quantitative Analysis in Sports*, 9(4):319–335, 2013.

[Thu02] Edward G Thurber. Concerning the maximum number of stable matchings in the stable marriage problem. *Discrete Mathematics*, 248(1-3):195–219, 2002.

[Von17] Allen IK Vong. Strategic manipulation in tournament games. *Games and Economic Behavior*, 102:562–567, 2017.

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227. IEEE, 1977.