# Computational Geometry and Online Algorithms

**Ali Mohammad Lavasani**

**A Thesis**

**in**

**The Department**

**of**

**Computer Science and Software Engineering**

**Presented in Partial Fulfillment of the Requirements**

**for the Degree of**

**Doctor of Philosophy (Computer Science) at**

**Concordia University**

**Montréal, Québec, Canada**

**December 2024**

**© Ali Mohammad Lavasani, 2025**

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mr. Ali Mohammad Lavasani**

Entitled: **Computational Geometry and Online Algorithms**

and submitted in partial fulfillment of the requirements for the degree of

**Doctor of Philosophy (Computer Science)**

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
*Dr. Claudio Contardo*

_____ External Examiner
*Dr. Stephane Durocher*

_____ Examiner
*Dr. Chadi Assi*

_____ Examiner
*Dr. Lata Narayanan*

_____ Examiner
*Dr. Thomas Fevens*

_____ Supervisor
*Dr. Denis Pankratov*

Approved by _____
Joey Paquet, Chair
Department of Computer Science and Software Engineering

_____ 2024

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

# Abstract


**Computational Geometry and Online Algorithms**

**Ali Mohammad Lavasani, Ph.D.**

**Concordia University, 2025**

This thesis explores several problems in computational geometry and online algorithms, focusing on efficient algorithms for geometric optimization and real-time decision-making. We begin by addressing the offline computational geometry problem of the *Maximum Weighted Convex Polytope* (MWCP), followed by two online algorithmic problems: *Online Non-Crossing Matching* and *Online Interval Scheduling*.

In the MWCP problem, the goal is to find a convex polytope within a set of $n$ weighted (positive and negative) points in $\mathbb{R}^d$ that maximizes the total weight of points inside or on the boundary. We present a new simple algorithm for the two-dimensional case, achieving the same time complexity of $O(n^3)$ as previous methods. We also prove that MWCP is $\mathcal{NP}$-hard in dimensions three or higher, even when weights are restricted to $+1$ and $-1$, and that in dimensions four or higher, the problem is $\mathcal{NP}$-hard to approximate within a factor of $n^{\frac{1}{2}-\epsilon}$.

We next focus on the *Online Non-Crossing Matching* problem, where points arrive sequentially in the plane and must be irrevocably matched to previously arrived points such that the resulting matched pairs form non-crossing line segments. We introduce the weighted version of this problem, aiming to maximize the total weight of matched pairs. We show that deterministic algorithms can have arbitrarily bad competitive ratios due to adversarial weight assignments. To address this, we consider weights within the range $[1, U]$ and provide an algorithm with a competitive ratio of $\Omega\left(2^{-2\sqrt{\log U}}\right)$, along with an upper bound of $O\left(2^{-\sqrt{\log U}}\right)$ for any deterministic algorithm. In the setting that allows revoking, we develop an algorithm that achieves a competitive ratio of approximately $0.28$, and prove that no deterministic algorithm can exceed a competitive ratio of $2/3$,

even in the unweighted case. Additionally, we propose a randomized algorithm with a competitive ratio of $1/3$, and show that no randomized algorithm can surpass a competitive ratio of $8/9$ in the unweighted case.

We also study the advice complexity of this problem. We establish a lower bound of $(n/3)-1$ on the advice complexity and provide an algorithm that uses approximately $2n$ bits of advice, matching $\log C_n = 2n - O(\log n)$, where $C_n$ is the $n^{\text{th}}$ Catalan number. Furthermore, we derive a lower bound on the advice complexity required to achieve a competitive ratio $\alpha \in (16/17, 1)$. In the bichromatic version of this problem, where a set of $n$ blue points are given offline and red points arrive online to be matched only to blue points, we correct previous errors in the literature and establish a lower bound of $\log C_n$ on the advice complexity. We also present an algorithm that uses $\log C_n$ bits of advice when points arrive in convex position.

In the final part, we consider the *Online Interval Selection* problem when the interval graph of the input is a simple path, allowing revoking of previous decisions. We show that under the random-order model, a deterministic memoryless algorithm achieves a competitive ratio of $2\left(1 - \frac{1}{\sqrt{e}}\right) \approx 0.78$. We also established an upper bound of $3/4$ for any deterministic revoking algorithm on a simple chain in the adversarial model, and a lower bound of $n/4$ for the advice complexity of Online Interval Selection, marking the first lower bound for this problem.

Throughout this work, we develop novel algorithmic techniques, establish tight complexity bounds, and provide new insights into advice complexity for key problems in computational geometry and online algorithms. These contributions aim to advance the theoretical foundations of these fields and have potential applications in real-time systems and geometric data analysis.

# Acknowledgments

I am deeply grateful to my supervisor, Denis Pankratov, for his invaluable guidance and support throughout my PhD journey. His deep understanding of the field and insightful feedback shaped my research and had a lasting impact on how I approach learning and problem-solving.

I would also like to thank Yaqiao Li. Many of the results in this thesis are the product of our collaboration. Working alongside him has been one of the most rewarding aspects of my PhD experience, and I have learned so much from his expertise and friendship.

I am thankful to my friends, especially Masih Aminbeidokhti, Armita Mohammadi, Mehran Shakerinava, and Motahareh Sohrabi, for being there through thick and thin. Their support meant the world to me and kept me going.

Lastly, I am forever grateful to my parents. My mother, Effat Mollanazar, for her unwavering dedication to my education, and my father, Mahdi Lavasani, for introducing me to the beauty of mathematics. Their belief in me has been my foundation.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Computational geometry is a branch of computer science that focuses on the study of algorithms for solving problems involving geometric objects. These objects can include points, lines, polygons, and polytopes. The central goal of computational geometry is to design efficient algorithms to handle geometric data, addressing problems such as computing convex hulls, finding intersections, and optimizing geometric configurations. The field draws heavily on mathematical foundations, including combinatorics, topology, and linear algebra, but its primary focus is on the algorithmic complexity and computational feasibility of these problems. As geometric data becomes increasingly prevalent in various domains, the development of robust and efficient algorithms has become essential.

Online algorithms are a class of algorithms designed to make decisions incrementally, processing input piece by piece as it arrives without knowledge of future inputs. Unlike offline algorithms, which have access to the complete input in advance, online algorithms must navigate uncertainty and optimize their performance in real time, adapting to new information as it becomes available. They are essential in situations requiring immediate decision-making, such as data structures, scheduling, caching, and networking.

A key metric for evaluating the performance of online algorithms is the *competitive ratio*. This measure compares the efficiency of an online algorithm to that of an optimal offline algorithm, which operates with full knowledge of future inputs. The competitive ratio captures the worst-case scenario by evaluating how closely the online algorithm approximates the cost or performance of

the optimal offline solution.

To assess the robustness of online algorithms, researchers often model future inputs as being generated by an adversary, who seeks to maximize the difficulty of the problem for the algorithm. This approach ensures that the algorithm is equipped to handle even the worst-case scenarios that may arise in the future.

Online algorithms were first formally introduced in the 1980s, particularly in the study of paging and caching problems, with Sleator and Tarjan [58] pioneering the use of the competitive ratio to compare online and offline algorithms. Since then, the field has expanded to cover a broad range of problems, including those in computational geometry, such as the k-server problem and online matching, contributing to more generalized theoretical frameworks.

It is worth noting that online problems can be viewed as information-theoretic problems. We often do not consider the computational complexity or even feasibility of such algorithms. This gives us robust knowledge about "what cannot be done". Although this assumption is not realistic in the algorithm design, except in very few examples, all known algorithms in this field have efficient polynomial computational complexity.

Advice complexity is a measure used in online algorithms to quantify the amount of additional information (or "advice") required to achieve optimal or near-optimal performance. In this model, an all-knowing oracle provides bits of advice to the algorithm, which can read these bits to make decisions as input elements are revealed. The goal is to minimize the number of advice bits needed while improving the algorithm's competitive ratio. Advice complexity bridges the gap between purely online and offline algorithms by offering insights into how much foresight improves decision-making.

In the following, we outline the structure of the thesis, provide brief definitions of the problems studied, offer concise literature reviews, and summarize the results obtained.

## 1.1 Chapter 3: Maximum Weighted Convex Polytope

The Maximum Weight Convex Polytope (MWCP) problem is defined as follows. Given a set of $n$ points $S$ in $\mathbb{R}^d$ with associated weights $w : S \to \mathbb{R}$, the weight of a polytope $P$ is the sum of

the weights of the points located inside or on the boundary of $P$, i.e., $w(P) = \sum_{x \in S \cap P} w(x)$. The objective is to find a convex polytope that maximizes this weight. It is important to note that the point weights can be negative; otherwise, the problem becomes trivial.

Despite its simple and natural formulation, surprisingly, this problem has not been widely studied in the computational geometry literature. The closest related work we discovered is by Bautista et al. [9], where the problem involves finding a monochromatic convex polytope, containing only red or only blue points, with maximum cardinality among a set of red and blue points. They also noted that with slight modifications, their algorithm could solve MWCP in $O(n^3)$ time.

### Our Contributions

- We present a new, arguably simpler algorithm that solves MWCP in two dimensions, maintaining the same time complexity of $O(n^3)$.
- We prove that MWCP in three or more dimensions is $\mathcal{NP}$-hard, even when the weights are limited to $+1$ and $-1$.
- We also demonstrate that solving MWCP in four or more dimensions is $\mathcal{NP}$-hard to approximate within a factor of $n^{\frac{1}{2}-\epsilon}$, even under the same weight restrictions of $+1$ and $-1$.

## 1.2 Chapters 4 and 5: Online Non-Crossing Matching

A set of $2n$ points arrives sequentially in the Euclidean plane. When a point arrives, the algorithm may match it with one of the previously arrived and unmatched points using a straight line segment. Each point must be paired with exactly one other point, and the resulting set of matched pairs (i.e., segments) must be non-crossing. The points are in general position, and decisions are irrevocable unless explicitly stated otherwise. The objective is to maximize the number of matched points.

In the offline setting, this problem is considered classical and can be solved in polynomial time. The study of online matching problems (without geometric constraints) was initiated by Karp et al. [42], where they examined the bipartite version of the problem. In this version, the final graph

forms a bipartite graph $G(U, V, E)$, with the point set $U$ available from the start. Points in set $V$ arrive online one at a time along with the edges connecting it to points in $U$. As with previous setups, the algorithm must decide whether to match the arriving vertex to one of its available (unmatched) neighbors or leave it unmatched.

Karp et al. introduced a randomized algorithm called *ranking*, which achieved a competitive ratio of $1 - \frac{1}{e}$, and proved that no algorithm can perform better. This problem has been extensively studied due to its important real-world applications, particularly in online advertising. Several extensions and variations have been explored with different objectives and constraints [10, 22, 7, 18, 29, 34, 39, 50].

The Online Non-Crossing Matching (ONM) problem was introduced by Bose et al. [15], who showed that any greedy algorithm that matches points whenever possible achieves a competitive ratio of $2/3$, and no deterministic algorithm can improve on this result. Later, Kamali et al. [40] proposed a randomized algorithm that increased the competitive ratio to approximately $0.6695$. Sajadpour [57] further proved that no randomized algorithm can achieve a competitive ratio better than $0.9262$.

Inspired by the online bipartite matching problem, Bose et al. [15] also introduced the bichromatic version of the Online Non-Crossing Matching problem. In this version, $n$ red points are available offline, and $n$ blue points arrive online. Each matching must pair one red and one blue point. They presented an algorithm with a competitive ratio of $\log n - o(\log n)$ and proved that no deterministic algorithm can achieve a better ratio.

Additionally, they explored the advice complexity of this problem. For the monochromatic version, they provided an algorithm with an advice complexity of $2n \log 3 + o(n)$ and established a lower bound of $\Omega(\log n)$. For the bichromatic version, they proposed an algorithm with an advice complexity of $n \log n$ but incorrectly claimed an upper bound of $n \log n$.

### Our Contributions - Advice Complexity

- We establish a lower bound of $n/3 - 1$ on the advice complexity for the monochromatic version.

- We propose an algorithm that solves the monochromatic version using approximately $2n$ bits of advice, corresponding to $\log C_n = 2n - O(\log n)$ where $C_n$ is the $n^{\text{th}}$ Catalan number.
- We correct a previous error in the proof of the $n \log n$ lower bound, providing a lower bound of $\log C_n$ on the advice complexity for the bichromatic version.
- We present an algorithm that uses $\log C_n$ bits of advice for the bichromatic version, assuming that points arrive in a convex position.
- We derive a lower bound of $\frac{\alpha}{2} D(\frac{2(1-\alpha)}{\alpha} || 1/4)n$ on the advice complexity required to achieve a competitive ratio of $\alpha \in (16/17, 1)$ for the monochromatic version, where $D(p||q)$ represents the relative entropy between two Bernoulli random variables with parameters $p$ and $q$.

In addition to advancing the previous results on advice complexity, we introduce the weighted version of the problem, where each point is assigned a positive weight, and the objective is to maximize the total weight of the matched points. We show that any deterministic algorithm can have an arbitrarily bad competitive ratio in this context. We then explore several approaches, including parametrizing the maximum weight (restricting weights to the range $[1, U]$), allowing revoking (where the algorithm can revoke previously matched pairs), and incorporating randomization. Interestingly, each of these approaches result in a constant competitive ratio, assuming $U$ is treated as a constant rather than a variable.

## Our Contributions - Weighted

- For weights restricted to the range $[1, U]$, we provide an algorithm with a competitive ratio of $\Omega\left(2^{-2\sqrt{\log U}}\right)$ and establish an upper bound of $O\left(2^{-\sqrt{\log U}}\right)$ for any deterministic algorithm.
- We develop an algorithm with revoking that achieves a competitive ratio of approximately $0.2862$, and we prove that no deterministic algorithm can exceed a competitive ratio of $2/3$, even in the unweighted version.
- We propose a randomized algorithm with a competitive ratio of $1/3$ and demonstrate that

no randomized algorithm can surpass a competitive ratio of $8/9$, even for the unweighted version.

- Additionally, we show that when points arrive on a line, neither randomization nor revoking alone improves the competitive ratio meaningfully. However, we present a randomized algorithm with revoking that achieves a competitive ratio of $0.5$.

It is worth noting that the randomized algorithm for the weighted version inspired the algorithm with the tight bound for the advice complexity on the monochromatic version.

## 1.3 Chapter 6: Online Interval Selection

In the Online Interval Selection (OIS) problem, a set of intervals $I = \{I_1, I_2, \ldots, I_n\}$ arrives one by one, and the algorithm must decide whether to accept or reject each interval as it arrives, ensuring that the selected intervals do not overlap. In the unweighted version of this problem, discussed in this thesis, the objective is to maximize the number of selected intervals. In the revocable acceptance setting, the algorithm is allowed to revoke previously selected intervals to accept new ones, while maintaining a set of non-overlapping intervals at all times. These intervals form a simple chain if each interval $I_i$ overlaps only with its neighbors, $I_{i-1}$ and $I_{i+1}$ (when they exist).

The Online Interval Selection problem poses a significant challenge in the field of online algorithms and has important applications in areas like job scheduling and network resource allocation. It has been widely studied within the online algorithm community. For an overview of the results and applications in the field of interval selection, we direct the reader to the surveys by Kolen et al. [43] and Kovalyov et al. [44].

In the *real-time model*, where intervals arrive in increasing order of their start times, the algorithm benefits from partial foresight. Faigle and Nawijn [28] proposed a 1-competitive deterministic algorithm, which is optimal when revocable decisions are allowed, enabling previously accepted intervals to be replaced by better ones that arrive later. Without revoking, the competitive ratio deteriorates significantly, as shown by Lipton and Tomkins [46], who established a upper bound of $O(n^{-1})$ for deterministic algorithms.

In the *random-order model*, where intervals arrive in a uniformly random sequence, Borodin and Karavasilis [12] demonstrated that a greedy algorithm can achieve a competitive ratio of $0.4$. Revocable decisions play a key role in this model, allowing deterministic algorithms to leverage randomness in the arrival sequence and enhance performance. Studies on revocable decisions, such as those by Borodin and Karavasilis [12].

### Our Contributions

- We show that a deterministic memoryless algorithm with revoking achieves a competitive ratio of $2(1 - 1/\sqrt{e}) \approx 0.786$ on a simple chain under the random-order input model.
- We prove an upper bound of $3/4$ for any deterministic revoking algorithm on a simple chain in the adversarial model.
- We prove a lower bound of $n/4$ for the advice complexity of Online Interval Selection.

## 1.4 Summary

The following table summarizes the definitions of the problems and highlights the similarities and differences.

| **Problem** | **MWCP** | **ONM** | **OIS** |
|---|---|---|---|
| **Input Items** | points $\in \mathbb{R}^n$ | points $\in \mathbb{R}^2$ | Intervals $\subseteq \mathbb{R}$ |
| **Decision** | Connect points to form a polytope | Connect points to form a matching | Select a set of intervals |
| **Constraint** | Convexity | Non-crossing | Non-crossing |

Table 1.1: Comparison of the Maximum Weighted Convex Polytope (MWCP), the Online Non-Crossing Matching (ONM), the Online Interval Selection (OIS) Problems.

# Chapter 2

# Preliminaries

In this chapter, we provide a brief overview of complexity theory and online algorithms, along with formal definitions that are used throughout the thesis or aid in understanding the chapter introductions. More specific preliminaries and notations relevant to individual chapters are introduced within each chapter as needed.

## 2.1 Basic Mathematical Notations

We show the set of positive real numbers by $\mathbb{R}_{>0} = \{x \in \mathbb{R} | x > 0\}$ and the set of integers $\{1, 2, \ldots, n\}$ by $[n]$. Asymptotic notations formally describe the behavior of functions as the input size $n$ approaches infinity. *Big-O* notation, denoted $O(g(n))$, provides an upper bound on the growth rate of a function. A function $f(n)$ is $O(g(n))$ if there exist positive constants $c$ and $n_0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. Formally,

$$f(n) = O(g(n)) \iff \exists c > 0, n_0 > 0 : \forall n \geq n_0,\ f(n) \leq c \cdot g(n).$$

*Little-o* notation, denoted $o(g(n))$, gives a strict upper bound, indicating that $f(n)$ grows slower than $g(n)$. Formally, $f(n) = o(g(n))$ if for every constant $c > 0$, there exists an $n_0$ such that $0 \leq f(n) < c \cdot g(n)$ for all $n \geq n_0$. The definition is:

$$f(n) = o(g(n)) \iff \forall c > 0, \exists n_0 > 0 : \forall n \geq n_0,\ f(n) < c \cdot g(n).$$

*Big-Omega* notation, denoted $\Omega(g(n))$, provides a lower bound. A function $f(n)$ is $\Omega(g(n))$ if there exist positive constants $c$ and $n_0$ such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$. Formally,

$$f(n) = \Omega(g(n)) \iff \exists c > 0, n_0 > 0 : \forall n \geq n_0,\ f(n) \geq c \cdot g(n).$$

*Little-omega* notation, denoted $\omega(g(n))$, gives a strict lower bound. It indicates that $f(n)$ grows faster than $g(n)$. Formally, $f(n) = \omega(g(n))$ if for every constant $c > 0$, there exists an $n_0$ such that $f(n) > c \cdot g(n)$ for all $n \geq n_0$. The definition is:

$$f(n) = \omega(g(n)) \iff \forall c > 0, \exists n_0 > 0 : \forall n \geq n_0,\ f(n) > c \cdot g(n).$$

*Theta* notation, denoted $\Theta(g(n))$, describes a tight bound on the growth of $f(n)$, meaning $f(n)$ grows at the same rate as $g(n)$. Formally, $f(n) = \Theta(g(n))$ if there exist positive constants $c_1, c_2$, and $n_0$ such that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$. The definition is:

$$f(n) = \Theta(g(n)) \iff \exists c_1 > 0, c_2 > 0, n_0 > 0 : \forall n \geq n_0,\ c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n).$$

## 2.2 Geometry

For two points $p$ and $q$ in the plane, we use the notation $\overline{pq}$ to refer to both the line segment between $p$ and $q$ and the line passing through them, depending on the context. The right (left) side of $\overline{pq}$ refers to the right (left) side of the vector $\overrightarrow{pq}$.

In the discussions of adversarial inputs and algorithms in Chapter 4 and Chapter 5, we employ a technique called "convex partitioning" of the plane. Starting with the entire plane as a single region, if points $p$ and $q$ lie within a region $R$, we can divide $R$ into two regions using the line $\overline{pq}$. The specific choices of points used to partition the plane into convex regions will be detailed in each scenario. The following fact is being used in multiple proofs of Chapter 4 and Chapter 5.

**Fact 1.** *A set of $2n$ points in the plane always admits a perfect non-crossing matching (a matching of size $n$).*

*Proof.* Consider a perfect matching of minimum total length on these $2n$ points. We claim that this

matching is non-crossing. For contradiction, suppose two segments in this matching are crossing. These crossing segments form diagonals of a quadrilateral, which can be replaced by the two sides of the quadrilateral, resulting in a perfect matching of smaller total length. This contradicts the assumption that the original perfect matching has the minimum total length, proving that the matching is non-crossing. □

## 2.3 Offline Algorithms

Complexity theory, a branch of computer science, classifies computational problems based on their inherent difficulty. It provides a framework for understanding how resources, such as time and space, required to solve a problem scale with the size of the input. This thesis focuses specifically on time complexity (Chapter 3).

A Turing machine is a theoretical model of computation that describes an abstract device capable of manipulating symbols on a tape based on a finite set of predefined rules. Although it is a simple concept, a Turing machine can simulate the logic of any (classical) computer algorithm, thus serving as the foundational model for understanding computability and complexity in theoretical computer science.

In complexity analysis, we often abstract away the details of the computational steps of a Turing machine and instead focus on the number of high-level operations or executed lines of code in an algorithm. This approach allows us to analyze and compare algorithms more easily, as we are primarily interested in the order of magnitude of the running time, not the exact number of steps.

We use time complexity notation for problems as well. The lower bound on the time complexity of a problem indicates that no algorithm exists with a better time complexity than the bound for that problem. If an algorithm solves the problem, it provides an upper bound for the problem's time complexity.

A computational problem with binary output (yes or no) is called a decision problem. $\mathcal{P}$ (polynomial time) is the class of decision problems that can be solved by a deterministic Turing machine in polynomial time, meaning there exists an algorithm that can solve the problem in time $O(n^k)$ for some constant $k$. $\mathcal{NP}$ (nondeterministic polynomial time) is the class of decision problems for

which solutions can be verified by a deterministic Turing machine in polynomial time.

Despite being one of the most fundamental questions in computer science, it remains unknown whether there exists an $\mathcal{NP}$ problem without a polynomial algorithm. However, there is a set of problems known to be as hard as any problem in $\mathcal{NP}$. To define hardness formally, we first define the notion of *reduction*. A problem $A$ is said to be reduced to problem $B$, denoted as $A \leq_p B$, if there exists a polynomial-time computable function $f$ from the set of instances of $A$ to the set of instances of $B$ such that for every instance $x$ of $A$, $x$ is a yes-instance of $A$ if and only if $f(x)$ is a yes-instance of $B$.

A problem $A$ is $\mathcal{NP}$-hard if any problem in $\mathcal{NP}$ can be reduced to $A$. A problem $A$ is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and it is $\mathcal{NP}$-hard. The implication is that if we want to prove a problem $A$ is hard, we pick a problem known to be $\mathcal{NP}$-hard, such as the maximum independent set problem, and reduce it to $A$. While this does not mean $A$ lacks a polynomial-time algorithm, it suggests that finding one would resolve a question (whether $\mathcal{NP} = \mathcal{P}$) that has eluded computer scientists for decades.

**Definition 1.** *Given a graph $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges, the maximum independent set problem seeks to find the largest subset $I \subseteq V$ such that no two vertices in $I$ are adjacent.*

**Fact 2.** *The maximum independent set problem is $\mathcal{NP}$-hard.*

In addition to exact algorithms that solve problems optimally, complexity theory also studies approximation algorithms, which aim to find solutions that are "good enough" within a certain factor of the optimal solution. This is particularly useful for problems that are $\mathcal{NP}$-hard, where finding an exact solution efficiently (in polynomial time) is unlikely.

An approximation algorithm is an algorithm that finds approximate solutions to optimization problems. Instead of guaranteeing the optimal solution, an approximation algorithm guarantees that the solution will be within some factor of the optimal one.

For a maximization problem, let $I$ be an instance of the problem. Let $\textsc{Opt}(I)$ be the value of the optimal solution for input $I$, and $\textsc{Alg}(I)$ be the value of the solution found by an approximation algorithm ALG. We say ALG is an $f(n)$-approximation if for every input $I$ of size $n$, $\textsc{Alg}(I) \geq$

$f(n) \cdot \text{OPT}(I)$. Note that $f(n)$ can be a constant number. For some problems, like the maximum independent set problem, even finding a constant approximation algorithm is $\mathcal{NP}$-hard.

## 2.4 Online Algorithms

### 2.4.1 Online Problems and Algorithms

In online problems, the inputs are presented as a sequence of objects $I = (i_1, i_2, \ldots, i_n)$, revealed to the algorithm one by one. Upon the arrival of each item, the algorithm must make a decision from the available actions based on the sequence received so far, without knowledge of future inputs. These decisions are typically irrevocable, although some variations allow for the revocation of previous decisions, which will be discussed later.

The challenge in designing efficient online algorithms can be framed as an information-theoretic problem. The primary limitation is that the algorithm must function without foreknowledge of future inputs. Importantly, we do not impose any restrictions on the computational power of the online algorithm. This means an online algorithm can have exponential time complexity and even access functions beyond the capability of a Turing machine. Although this might seem impractical, most online algorithms are executed in polynomial time with few exceptions. Furthermore, the upper bound on the performance of these algorithms (for maximization problems) is robust and demonstrates a limit even if $\mathcal{NP} = \mathcal{P}$.

### 2.4.2 Online Algorithmic Models

**Deterministic Online Algorithms.** Let $I = (i_1, \ldots, i_n)$ be an input instance of an online problem. An online deterministic algorithm ALG on this input makes a series of decisions $D = (d_1, \ldots, d_n)$ such that $d_i$ is made only by observing $i_{\leq j}$ $(i_1, \ldots, i_j)$. The value of ALG is a function of the input sequence and decisions.

**Randomized Online Algorithms.** An online randomized algorithm $\text{ALG}_R$ has access to a set of random variables $R$ and hence each of its decisions is randomly selected from the set of all available actions. We consider the *oblivious* adversary that knows the the algorithm, but not outcomes of $R$.

**Online Algorithms with Advice.** In this framework, a powerful oracle has complete foresight of

the entire input and writes bits onto an infinite advice tape in advance. The online algorithm, ALG, processes the input one item at a time and can access any number of advice bits from the tape during its execution. The oracle is reliable and operates to maximize the performance of the online algorithm, following a predefined protocol. Therefore, the oracle and the algorithm work together to achieve optimal results against an adversarial input. The worst-case number of advice bits accessed by the online algorithm on inputs of length $n$ defines the *advice complexity* of the algorithm as a function of $n$. The advice complexity of ALG is represented as $a(\text{ALG}, n)$.

### 2.4.3 Measuring Performance of Online Algorithms

With a slight abuse of notation, we use $\text{ALG}(I)$ for both the output and the objective value of ALG after receiving input $I$ entirely. Similar to offline setting and complexity analysis, we can measure the performance of an online algorithm as a game between the algorithm and an adversary that knows the algorithm and chooses the worst input sequence accordingly. This shows the performance of the algorithm in the worst-case scenario. The *competitive ratio* of a deterministic algorithm in the adversarial input model is defined by:

$$\rho(\text{ALG}) = \liminf_{|I|\to\infty} \frac{\text{ALG}(I)}{\text{OPT}(I)}$$

OPT denotes the best possible algorithm that knows the input sequence in advance i.e. the optimal offline algorithm. This competitive ratio is rather asymptotic. If $\text{ALG}(I) \geq \rho\text{OPT}(I)$ we say ALG has a *strict* competitive ratio of $\rho$.

In the case of a randomized algorithm $\text{ALG}_R$ which uses a set of random variables $R$, the adversary knows the algorithm but it is oblivious to the outcome of random variables in $R$. The competitive ratio for $\text{ALG}_R$ is defined by:

$$\rho(\text{ALG}_R) = \liminf_{|I|\to\infty} \frac{\mathbb{E}_R\{\text{ALG}_R(I)\}}{\text{OPT}(I)}$$

In the random order input model, the set of input $I$ is generated by the adversary but the order of arrival . In this case, we can think of input $I$ as a set $S$ and a permutation $\pi$ indicating the arrival order. Thus in this model, the competitive ratio is defined by:

$$\rho(\text{ALG}_R) = \liminf_{|S|\to\infty} \frac{\mathbb{E}_{R,\pi}\{\text{ALG}_R(S,\pi)\}}{\text{OPT}(S)}.$$

### 2.4.4 Online Algorithms with Advice

In this subsection, we briefly discuss two different models of algorithms with advice: the advice tape model and the parallel algorithms model. For completeness, we present a known argument demonstrating the equivalence (up to lower order terms) of two models. Thus, in the rest of the chapter, we shall use two models interchangeably.

**Advice tape model.** In this model, there is an all-powerful oracle that sees the entire input in advance and populates an infinite advice tape with bits. Then the online algorithm ALG receives the input one item at a time and ALG has the ability to read any number of bits of advice from the tape at any point during the runtime. The oracle is trustworthy and always behaves in the best interests of the online algorithm according to a pre-agreed protocol. Thus, the oracle and the algorithm co-operate to achieve the best possible performance against an adversary. The worst-case number of advice bits read by the online algorithm with advice on inputs of length $n$ is the *advice complexity* of the algorithm (as a function of $n$). The advice complexity of ALG is denoted by $a(\text{ALG}, n)$.

**Parallel online algorithms model.** In this model, a set of deterministic online algorithms (without advice) $\mathcal{A}$ is said to solve a set of inputs $\mathcal{I}$ if for every input sequence $I \in \mathcal{I}$ there is at least one algorithm $\text{ALG} \in \mathcal{A}$ such that ALG solves $I$ optimally (or within the given level of approximation). We define the *width* of $\mathcal{A}$ as $\lceil \log |\mathcal{A}| \rceil$), denoted by $w(\mathcal{A})$.

**Fact 3** (Equivalence of the two models)**.** *The advice tape model and the parallel online algorithms model are equivalent (up to additive $O(\log n)$ terms) in the following sense:*

*(1) If* ALG *is an algorithm in the advice tape model that solves inputs of length $n$ correctly then there is a set $\mathcal{A}$ of algorithms in the parallel online algorithms model that solves the same inputs correctly and such that $w(\mathcal{A}) \leq a(\text{ALG}, n)$.*

*(2) If $\mathcal{A}$ is a set of algorithms that solves a set of inputs of length $n$ correctly in the parallel online algorithms model then there exists an algorithm* ALG *in the tape model with advice that solves the same inputs correctly and such that $a(\text{ALG}, n) \leq w(\mathcal{A}) + O(\log w(\mathcal{A}))$.*

*Proof.*

(1) For $x \in \{0,1\}^{a(\text{ALG},n)}$ we define $\text{ALG}_x$ to be the algorithm ALG with the first $a(\text{ALG}, n)$ bits on the advice tape fixed to be $x$ and restricted to inputs of length $n$. Then setting $\mathcal{A} = \{\text{ALG}_x\}_{x\in\{0,1\}^{a(\text{ALG},n)}}$ establishes the claim.

(2) The oracle and the online algorithm ALG agree on the ordering of $\mathcal{A}$. The oracle uses Elias delta coding scheme to encode the index of the first algorithm from $\mathcal{A}$ that solves the given input sequence correctly. This requires $w(\mathcal{A})+O(\log w(\mathcal{A}))$ bits of advice, the rest of the bits are arbitrary. ALG starts by reading and decoding the index of an algorithm from $\mathcal{A}$ written on the advice tape, and then runs that algorithm.

□

Since $O(\log w(\mathcal{A}))$ additive terms are small in our setting, we shall use the two models interchangeably and we shall sometimes refer to $w(\mathcal{A})$ as the advice complexity. Analogous statements and considerations hold for approximation.

# Chapter 3

# Maximum Weight Convex Polytope

In this chapter[1], we introduce an offline problem, the *Maximum Weight Convex Polytope*, or MWCP for short. Suppose you are given a set of $n$ points $S$ in $\mathbb{R}^d$ with weights $w : S \to \mathbb{R}$, where weights can be positive or negative. The weight of a polytope $P$ is defined as $w(P) = \sum_{v \in S \cap P} w(v)$ (see Figure 3.1). The objective is to find a convex polytope with maximum weight. This is a rather natural and fundamental computational geometry question.

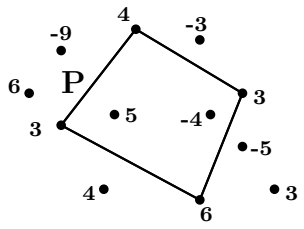



Figure 3.1: An example illustrating the calculation of the weight of a polytope, where $w(P) = 4 + 3 + 6 + 3 + 5 - 4 = 17$.

## 3.1 Literature Review

MWCP with a binary weight function, such as $w : S \to \{+1, -1\}$, belongs to a large class of computational geometry problems on bichromatic point sets with weights $\{+1, -1\}$ corresponding to two colors, typically labelled "red" and "blue". For example, in the maximum box problem, one

[1]The results presented in this chapter are based on joint work by the author in collaboration with Mohammad Ali Abam and Denis Pankratov [2].

is given a set of $r$ red points and a set of $b$ blue points in the plane and the goal is to find an axis-aligned rectangle which maximizes the number of blue points and does not contain any red points. Liu and Nediak [47] gave an exact $O(r \log r + r + b^2 \log b)$ algorithm, and Eckstein et al. [27] construct an efficient branch-and-bound algorithm motivated by a problem in data analysis. Liu and Nediak [47] also show how to solve efficiently a related bichromatic separability with two boxes problem, introduced by Cortés et al. [20].

MWCP is also related to bichromatic discrepancy problems, where one is given two finite sets of points $S^+$ and $S^-$ in $\mathbb{R}^d$, and the goal is to find an axis-aligned box $B$ maximizing the difference between the number of the points of $S^+$ and $S^-$ inside the box, i.e. $||B \cap S^+| - |B \cap S^-||$. Let $n = |S^+ \cup S^-|$ denote the total number of points. Dobkin et al. [23] solved this problem in $\mathbb{R}^2$ in $O(n^2 \log n)$ time. Liu and Nediak [47] presented a 2-factor approximation for this problem in $\mathbb{R}^2$ with $O(n \log^2 n)$ running time.

In another related problem, namely, the numerical discrepancy problem, one is given a set of $n$ points $S \subset [0,1]^2$. The goal is to find a box $B$ that maximizes the numerical discrepancy of $B$ defined as $||B \cap S|/|S| - \mu(B)|$, where $\mu(B)$ denotes the area of $B$. Observe that the numerical discrepancy of $B$ can be thought of as measuring the deviation of the empirical distribution from the uniform distribution. Dobkin et al. [23] solved this problem in $\mathbb{R}^2$ in $O(n^2 \log^2 n)$ time. Liu and Nediak [47] presented a 2-factor approximation for this problem in $\mathbb{R}^2$ with $O(n \log^3 n)$ running time.

In the mentioned problems, the solution shape is constrained to be an axis-aligned box, unlike the general convex polytope considered in MWCP. In another variation studied by González-Aguilar et al. [33], the geometric shape of the solution is restricted to be a rectilinear convex hull of points (note that the rectilinear convex hull is not necessarily a convex subset of $\mathbb{R}^2$). González-Aguilar et al. [33] gave an $O(n^3)$ algorithm for this problem.

We note that the above problems are very similar to our problem at first glance. A deeper investigation shows that the nature of restriction on the solution set is crucial for the above problems and algorithms for them, and so new ideas and techniques are needed for MWCP problem. There is one other problem that is directly relevant to MWCP, and that is the optimal islands problem studied by Bautista et al. [9]. In this problem, one is given a set $S$ of $n$ points colored with 2 colors in

the plane. A subset $\mathcal{I} \subseteq S$ is called an island of $S$, if $\mathcal{I}$ is an intersection of $S$ and a convex set $C$. Bautista et al. [9] gave an $O(n^3)$-time algorithm to find a monochromatic island of maximum cardinality. Their algorithm can also be used to solve the MWCP problem in 2 dimensions.

The class of problems to which MWCP belongs have important practical applications in data analysis and machine learning. In particular, Bautista et al. [9] were motivated by clustering applications. Given a training dataset of points $S \subset \mathbb{R}^d$ that are labelled with two colors "red" and "blue", in a classification problem one is interested in a simple description of a region of space corresponding to the class of "red" points, for example. One possibility is to use convex hulls for such a description (see, for example, Kudo et al. [45]). If dataset is 2-dimensional one arrives naturally at the optimal islands problem. However, datasets are often noisy, so one should not expect to see large monochromatic islands, so perhaps weighted version of the problem, such as MWCP, might be more suitable. A bigger issue is that in classification problems datasets are often high dimensional and one cannot always hope to obtain clusters by projecting to 2 dimensions first. Thus, for clustering applications it is important to be able to solve MWCP efficiently in high dimensions. This is the question we tackle in this chapter. Alas, we show that MWCP is $\mathcal{NP}$-hard in 3 dimensions (Theorem 11), and that it is $\mathcal{NP}$-hard to approximate within $n^{1/2-\epsilon}$ for any $\epsilon > 0$ in 4 dimensions even with binary weights (Theorem 14). We also give a completely new algorithm for 2 dimensions with running time $O(n^3)$ matching Bautista et al.

## 3.2 Preliminaries

Whenever we write "polytope" in this chapter we mean a convex polytope. $S$ denotes the input set of $n$ points in $\mathbb{R}^d$ for $d \geq 1$ and a weight function is denoted by $w : S \to \mathbb{R}$. The weight of a polytope $P$, denoted by $w(P)$, is defined as follows:

$$w(P) = \sum_{v \in S \cap P} w(v)$$

In MWCP problem, the goal is to find a polytope with maximum weight. Note that points $v \in S$ with $w(v) = 0$ do not affect weight of any polytope, and so they can be removed from the input in

a preprocessing step. Henceforth, we assume that for all $v \in S$ we have $w(v) \neq 0$. We use $S^-$ and $S^+$ for the subsets of points of $S$ with negative and positive weights respectively. For a set of points $C \subset \mathbb{R}^d$ we let $conv(C)$ denote the convex hull of $C$. With a slight abuse of notation, we define $w(C) = w(conv(C))$. A subset $C \subseteq S^+$ is *maximal* if for every $v \in C$, $w(C) > w(C \setminus \{v\})$.

A polytope has two standard equivalent descriptions: $\mathcal{V}$-polytope is described as a convex hull of vertices, and $\mathcal{H}$-polytope is described as an intersection of half-spaces. We shall primarily work with $\mathcal{V}$-polytopes due to the nature of MWCP problem. We let $vert(P)$ denote the set of vertices of a polytope $P$. Vertices of a polytope are also its 0-faces and edges of a polytope are its 1-faces. We state a few facts about polytopes here that will be used later in the chapter; for a more thorough introduction to polytope theory, the reader is referred to the excellent lecture notes of Ziegler [59].

**Fact 4** ($\mathcal{V}$-polytope definition)**.** *Let $P \subseteq \mathbb{R}^d$ be a polytope and $v \in \mathbb{R}^d$ be a point. $v \in P$ if and only if there is a convex combination of $vert(P)$ equal to $v$.*

**Fact 5.** *Let $P \subseteq \mathbb{R}^d$ be a polytope and $F$ be a face of $P$. The face $F$ is a polytope, with $vert(F) = F \cap vert(P)$.*

Let $P \subseteq \mathbb{R}^d$ be a polytope and $F$ be a face of $P$. For a hyperplane $h$ such that $F \subseteq h$ we define $h^-$ and $h^+$ to be the open half spaces bounded by $h$ such that $h^- \cap P = \emptyset$.

A polytope $P \in \mathbb{R}^d$ is a *polytope embedding* of a graph $G(V, E)$ if there exist a one-to-one function $f : V \to vert(P)$ such that if $(u, v) \in E$ then $(f(v), f(u))$ is an edge of $P$. Note that $P$ may have some extra edges compared to $G$. If $P$ has exactly $|E|$ edges, then we call this embedding a *polytope realization* of $G$.

## 3.3 Results

In this section we present our results for the MWCP problem beginning with an overview of upper bounds in Section 3.3.1 (where we present a new algorithm for 2 dimensions), followed by lower bounds for 3 and 4 dimensions in Section 3.3.2.

### 3.3.1 Upper Bounds for 1 and 2 Dimensions

We begin with a simple observation: we can assume without loss of generality that vertices of a maximum weight polytope are elements of $S^+$.

**Lemma 6.** *For every set $S$ of points in $\mathbb{R}^d$, there exists a maximum weight polytope $P$ with $vert(P) \subseteq S^+$.*

*Proof.* Let $P$ be a maximum weight polytope and define $C = S^+ \cap P$. The convex hull $conv(C)$ is a subset of $P$ that has all the positive points of $P$. The convex hull $conv(C)$ contains all positive-weight points of $P$, and its vertices are a subset of $S^+$, ensuring that its weight satisfies $w(C) \geq w(P)$.

□

The above lemma implies that to solve MWCP it is sufficient to find a set $C \subseteq S^+$ with maximum weight of its convex hull. In particular, when $d = 1$ the MWCP problem reduces to the maximum subarray problem (consider the array of weights of points in $S$, in increasing order of their $x$-coordinates). The following result is immediate from well known algorithms for the maximum subarray problems.

**Theorem 7** (Folklore)**.** *The* MWCP *problem in 1 dimension can be solved in $O(n)$ time if the input points are sorted, and in $O(n \log n)$ otherwise.*

*Proof.* First, we sort the points from left to right and place their weights in an array $A$, which takes $O(n \log n)$ time. The sum of the points between any two points in the input corresponds to the sum of elements in $A$ between their respective positions. This reduces the problem to finding the maximum subarray sum, which can be solved using Kadane's algorithm in $O(n)$.

We iterate through the elements of $A$ from left to right, maintaining a variable, $curr\_sum$, which holds the maximum sum ending at the current element. If $curr\_sum$ becomes negative at any point, we reset it to the value of the current element. Otherwise, we add the value of the current element to $curr\_sum$. By tracking the maximum value of $curr\_sum$ throughout the iteration, we find the maximum subarray in $A$, and thus, MWCP for the input. □

Bautista et al. [9] gave a dynamic programming algorithm that solves the MWCP problem in 2 dimensions in $O(n^3)$ time. Their algorithm is based on a triangulation of a convex polytope from a topmost *anchor* vertex.

**Theorem 8** (Bautista et al. [9])**.** *The* MWCP *problem is solvable in* $O(n^3)$ *time in* 2 *dimensions* ($d = 2$)*.*

In the rest of this section we present a new algorithm which solves MWCP problem in 2 dimensions, albeit with the same $O(n^3)$ running time. Our algorithm is based on a different decomposition (see Figure 3.2), and is arguably simpler than the algorithm of Bautista et al.

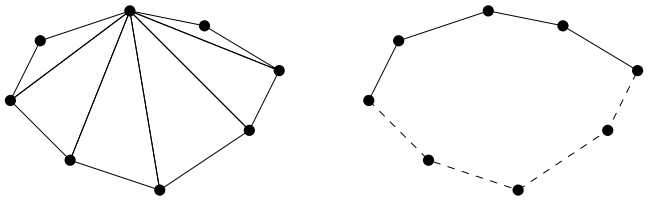


Figure 3.2: Two decompositions of a polytope which form a basis of two dynamic programming approaches. In the approach of Bautista et al. [9] (shown on the left) a polytope is decomposed via a triangulation from an anchor (topmost) vertex. In our approach (shown on the right) a polytope is decomposed into two paths from a leftmost to a rightmost vertex: top concave path (shown solid) and bottom convex path (shown dashed).

Without loss of generality, we can assume that no two points of $S$ have the same $x$-coordinates. Otherwise, in $O(n^2)$ time we can find a line $\ell$ such that $\ell$ is not parallel to any line passing through two point in $S$. Then we can rotate the axes so that the $y$-axis becomes parallel to $\ell$.

Let $p_1, \ldots, p_n$ be the points in $S$ sorted from left to right by their $x$-coordinates. Consider a directed edge from $p_i$ to $p_j$ for every $i < j$. Weight of the edge $p_i \to p_j$, denoted by $w(p_i, p_j)$, is the sum of all the weights of points $p_k$ such that $i < k < j$ and $p_k$ is below the line segment joining $p_i$ and $p_j$. We can use brute-force algorithm to compute $w(p_i, p_j)$ for all $i < j$ in $O(n^3)$ time. Thus, we assume that all these weights have been precomputed and are available to us when we need them. A *path* is a sequence of connected edges. For a path $\mathcal{P}$ we define its weight, denoted by $w(\mathcal{P})$, to be the sum of the weights of its edges and its vertices. For a path $\mathcal{P}$ we define its

sub-weight, denoted by $w^-(\mathcal{P})$, to be the sum of the weights of its edges only.

A polygon $P$ can be represented as a pair of paths consisting of a concave path $\mathcal{C}$ and a convex path $\mathcal{V}$ between its leftmost and its rightmost vertices (see Figure 3.2). Thus the weight of $P$ is equal to $w(\mathcal{C}) - w^-(\mathcal{V})$. We shall present a dynamic programming algorithm to solve the optimization version of the problem, where we are interested in computing the weight of a maximum-weight polygon only. The algorithm can be easily modified to find a maximum-weight polygon itself by the standard technique of remembering which choices resulted in individual entries of the dynamic programming tables.

For every $i < j \leq k$, let $C[i, j, k]$ (respectively $V[i, j, k]$) be the maximum (respectively, minimum) weight (respectively, sub-weight) of a concave (respectively, convex) path from $p_i$ to $p_k$ such that the first edge is $p_i \to p_j$. We denote the maximum weight of a polygon with leftmost vertex $p_i$ and rightmost vertex $p_k$ by $M[i, k]$. If $i = k$ then $M[i, k] = w(p_k)$, and if $i < k$ then $M[i, k]$ can be computed as:

$$M[i,k] = \max_{j:i<j\leq k} C[i,j,k] - \min_{j:i<j\leq k} V[i,j,k].$$

The solution to the overall problem is then given by the $\max_{i\leq k} M[i, k]$.

In the remainder, we explain how the table $C[i, j, k]$ can be computed. The table $V[i, j, k]$ is computed analogously with some trivial modifications (such as excluding contribution of vertices of the path, replacing concavity with convexity, and replacing maximization objective with minimization objective).

In the algorithm, we have to check whether a line segment joining vertices $p$ and $q$ can be extended to a vertex $r$ with $p.x < q.x < r.x$ while maintaining concavity. This can be tested by checking whether the vector $r - p$ is turned clockwise relative to the vector $q - p$ (see Figure 3.3). In turn, this can be achieved by checking the sign of 2-dimensional cross-product, denoted by $\times_2$, and defined as $v_1 \times_2 v_2 = v_1.x \cdot v_2.y - v_1.y \cdot v_2.x$. To summarize we have that the path $p \to q \to r$ is concave if and only if[2] $(r - p) \times_2 (q - p) > 0$.

[2] A bit of care is needed to handle inputs that are not in general position. If three points $p, q, r$ with $p.x < q.x < r.x$ are collinear then $(r - p) \times_2 (q - p) = 0$, and the path $p, q, r$ should be considered concave. However, this makes $q$ not a vertex of the resulting polytope, as it appears in the middle of an edge. In our description, we tacitly assumed that points are in general position to simplify the presentation. It is easy to extend our algorithm to handle points not in general

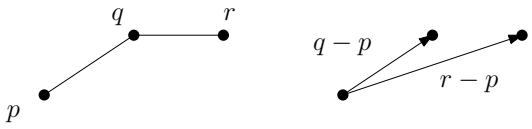



Figure 3.3: The path $p \to q \to r$ is concave if and only if vector $r - p$ is turned clockwise relative to vector $q - p$.

Base cases for the table $C[i, j, k]$ are the following:

$$C[i, k, k] = w(p_i, p_k) + w(p_i) + w(p_k) \qquad \text{if } i < k$$
$$C[i, j, k] = -\infty \qquad \text{if } i < j < k$$
$$\text{and } (p_k - p_i) \times_2 (p_j - p_i) < 0$$

It is clear that the other entries $C[i, j, k]$ with $i < j < k$ can be computed according to the following formula:

$$C[i, j, k] = \max_{j'}\{w(p_i, p_j) + w(p_i) + C[j, j', k] : \tag{1}$$
$$j < j' \leq k \text{ and } (p_{j'} - p_i) \times_2 (p_j - p_i) > 0\}.$$

A naive computation of the above table takes $O(n^4)$ time, since the table has $O(n^3)$ entries and each entry can be computed in $O(n)$ time. Next, we show a trick of how the time complexity can be reduced to $O(n^3)$. The idea is for a fixed $j$ and $k$ to fill in entries $C[i, j, k]$ for all $i$ in $O(n)$ time.

We precompute in $O(n^2 \log n)$ total time for all $j$ two lists: $L_j = (l_1, \ldots, l_{j-1})$ and $R_j = (r_1, \ldots, r_{n-j})$. $L_j$ ($R_j$) consists of points $\{p_1, \ldots, p_{j-1}\}$ (respectively, $\{p_{j+1}, \ldots, p_n\}$) to the left (respectively, to the right) of $p_j$ and sorted in clockwise order with respect to $p_j$ as the origin.

Now, fix a pair of indices $j < k$. In $O(n)$ time it is easy to compute $D[j', k] = \max_{j''}\{C[j, j'', k] : j'' \leq k$ and $p_{j''}$ is either $p_{j'}$ or appears after $p_{j'}$ in $R_j\}$. Define the first compatible $j'$ for the given $i, j$, denoted by $fc(i, j)$, as the first $p_{j'}$ appearing in $R_j$ such that $p_i \to p_j \to p_{j'}$ is concave. Then

position.

it is clear that $C[i, j, k]$ can be equivalently restated as follows:

$$C[i, j, k] = w(p_i, p_j) + w(p_i) + D[fc(i, j), k].$$

This is because, every $p_{j''}$ that appears after $fc(i, j)$ in $R_j$ also forms a concave path $p_i \to p_j \to p_{j''}$. Thus, the third term $D[fc(i, j), k]$ in the above equation is exactly the same as the third term in Equation (1).

Lastly, it is left to observe that as one considers points $p_i$ in the order in which they appear in $L_j$, the corresponding sequence of $fc(i, j)$ also forms an increasing sequence in $R_j$. Thus, by maintaining a running pointer into $R_j$ one can compute $fc(i, j)$ in $O(n)$ time for all $p_i \in L_j$. This finishes the description of the algorithm. One readily checks that all precomputing steps take $O(n^3)$, base cases of $C[i, j, k]$ can also be computed in $O(n^3)$ time, and all other entries can be computed in $O(n^3)$ as well, by iterating over all pairs $j < k$ and filling in $C[i, j, k]$ for all $i$ in $O(n)$ time.

### 3.3.2 Lower Bounds for 3 and 4 Dimensions

Recall that a *strict reduction* from an optimization problem $\mathcal{A}$ to an optimization problem $\mathcal{B}$ is a pair of functions $(f, g)$, where $f$ maps instances $x$ of $\mathcal{A}$ to instances $f(x)$ of $\mathcal{B}$ and $g$ maps solutions $y$ of $\mathcal{B}$ to solutions $g(y)$ of $\mathcal{A}$, such that the approximation ratio achieved by solution $y$ on instance $f(x)$ of $\mathcal{B}$ is at least as good as the approximation ratio achieved by solution $g(y)$ on instance $x$ of $\mathcal{A}$. All our lower bound results in this section are based on the following technical lemma.

**Lemma 9.** *Let $\mathcal{G}$ be a graph family. If for every $G \in \mathcal{G}$ a polytope embedding of $G$ into $\mathbb{R}^d$ can be found in polynomial time and bit complexity polynomial in $n$, then there is a strict reduction from the maximum independent set on $\mathcal{G}$ to* MWCP *in $d$ dimensions with weights $\{+1, -1\}$.*

*Proof.* Given input instance $G = (V, E)$ to the maximum independent set on $\mathcal{G}$, we let $P$ be the result of applying the polytope embedding to $G$. Let $S^+ := vert(P)$ and assign $+1$ weight to every vertex in $S^+$. Create set $S^-$ by adding two points with weights of $-1$ at two arbitrary positions of every graph edge. Let $S = S^+ \cup S^-$. For a negative point $v \in S^-$, let $p_1(v), p_2(v) \in S^+$ be positive-weighted vertices such that $v$ was placed on the edge joining $p_1(v)$ with $p_2(v)$ and $n(v)$ be the other negative point on that edge. See Figure 3.4 for an example.

We claim that for a subset $C \subseteq S^+$, there exists a negative point $v \in S^-$ in $conv(C)$ if and only if $p_1(v), p_2(v) \in C$. One direction is clear: if $p_1(v), p_2(v) \in C$ then by Fact 4 $n(v)$ and $v$ are in $conv(C)$. For the other direction, assume that $v \in conv(C)$. Let $e$ be the edge between $p_1(v)$ and $p_2(v)$. By the definition of $P$, there exist a hyperplane $h_e$ such that $S^+ \cap h_e^- = \emptyset$. Therefore $C \cap h_e^- = \emptyset$ and $F := conv(C) \cap h_e$ is a face of $conv(C)$. $F \neq \emptyset$ since $v$ is in $h_e$ and $conv(C)$. Only vertices of $S^+$ in $h_e$ are $\{p_1(v), p_2(v)\}$. By Fact 5 $vert(F) = F \cap vert(conv(C)) \subseteq h_e \cap S^+ = \{p_1(v), p_2(v)\}$. Without loss of generality suppose $vert(F) = \{p_1(v)\}$, this implies $v \notin F$ which is a contradiction. Thus $vert(F) = \{p_1(v), p_2(v)\}$ and $p_1(v), p_2(v) \in C$.

Let $C \subseteq S^+$ be a maximal subset. We claim that $conv(C)$ contains no negative points and all positive points in $conv(C)$ are precisely the vertices of $conv(C)$. First, suppose there exists a negative point $v \in conv(C)$ thus $p_1(v), p_2(v) \in C$ and $n(v) \in conv(C)$. $w(C \backslash \{p_1(v)\}) \geq w(C) + 2 - 1 > w(C)$ since $v, n(v), p_1(v) \notin conv(C \backslash \{v_i\})$. This is a contradiction to maximality of $C$. Second, suppose there exist a positive point $v \in S$ in $conv(C) \setminus vert(conv(C))$. Because $v$ is a vertex of $P$ there exist a hyperplane $h_v$ such that $S^+ \cap h_v^- = \emptyset$. Therefore $C \cap h_e^- = \emptyset$ and $v$ is a vertex of $conv(C)$ which is a contradiction.

Therefore, we can conclude that $w(C) = |C|$ if $C \subseteq S^+$ is maximal. Next, we prove there exists a maximal subset $C \subseteq S^+$ if and only if there exist an independent set $\mathcal{I} \subseteq V$ such that $w(C) = |\mathcal{I}|$.

**If:** Let $\mathcal{I} \subseteq V$ be an independent set and $C \subseteq S^+$ be the set of corresponding vertices of $\mathcal{I}$ in $S^+$. Because there is no edge between vertices in $\mathcal{I}$, there is no graph edge between vertices in $C$. Thus there are no negative points in $conv(C)$. Since all vertices inside $conv(C)$ are positive, $C$ is a maximal subset and $w(C) = |C| = |\mathcal{I}|$.

**Only if:** Let $C \subseteq S^+$ be a maximal subset and let $\mathcal{I} \subseteq V$ be the set of corresponding vertices of $C$ in $G$. Because $C$ is a maximal subset, there is no negative point in $conv(C)$, and there is no graph edge between vertices of $C$. Thus the set of corresponding vertices of $C$ in $G$ is an independent set. $|\mathcal{I}| = w(C)$ since $w(C) = |C|$.

Without loss of generality we can suppose every approximation algorithm for MWCP outputs a maximal subset of $S^+$. Thus there exist a strict reduction from the maximum independent set problem of graph $G(V, E)$ to MWCP in $\mathbb{R}^d$. □

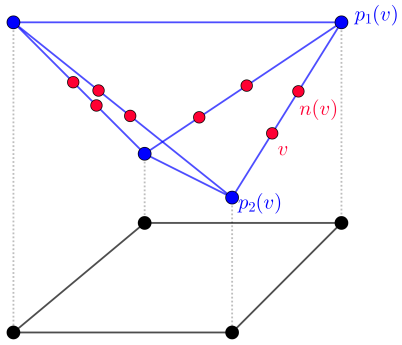



Figure 3.4: An example illustrating the placement of negative-weight points along edges to ensure that a polytope's convex hull encodes independent set constraints.

We obtain the lower bound for 3 dimensions by applying Lemma 9 to the class $\mathcal{G}$ of planar graphs. We note that the maximum independent set problem is $\mathcal{NP}$-hard even for planar graphs [32]. Our lower bound relies on the polynomial embedding in 3 dimensions due to Das et al. [21]. A *maximal planar graph* is a planar graph such that an addition of any new edge results in a non-planar graph.

**Lemma 10** (Das et al. [21])**.** *Given a maximal planar graph $G(V, E)$ with $n$ vertices, a polytope realization of $G$ in $\mathbb{R}^3$ can be found in $O(n)$ time and with bit complexity polynomial in $n$.*

Thus the following theorem can be easily deduced from Lemmas 9 and 10.

**Theorem 11.** *Let $S$ be a set of $n$ points in $\mathbb{R}^3$ with weight function $w$, finding* MWCP *of $S$ is $\mathcal{NP}$-hard even if $w : S \to \{-1, +1\}$.*

*Proof.* Let $\mathcal{G}$ be the family of all planar graphs. By adding edges to a planar graph $G$ we can make it maximal. The polytope realization of the new maximal planar graph is also a polytope embedding of $G$. Thus with Lemma 10 we can conclude for every $G \in \mathcal{G}$ a polytope embedding of $G$ in $\mathbb{R}^3$ can be found in polynomial time and with polynomial bit complexity. By Lemma 9, there is a strict reduction from maximum independent set on planar graphs to MWCP with weights $\{+1, -1\}$, hence it is an $\mathcal{NP}$-hard problem. □

Let $S$ be the set of points $(i, i^2, i^3, i^4)$ in $\mathbb{R}^4$ for $i \in [n]$. The convex hull of $S$ is known as the cyclic polytope on $n$ vertices in $\mathbb{R}^4$ and it is a polytope realization of a complete graph with $n$

vertices (for more details, see, for example, [59]). This gives the following lemma.

**Lemma 12.** *Given a complete graph $K_n$ with $n$ vertices, a polytope realization of it in $\mathbb{R}^4$ can be found in $O(n)$ time with a bit complexity polynomial in $n$.*

We can use Lemma 12 to show that MWCP in $4$ dimensions is as hard as finding a maximal independent set on arbitrary graphs. Zuckerman [60], strengthening an earlier result of Håstad [38], showed that it is $\mathcal{NP}$-hard to approximate independent set on arbitrary graphs within an $n^{1-\epsilon}$ factor for any $\epsilon > 0$.

**Theorem 13** (Zuckerman [60])**.** *For any $\epsilon > 0$ it is $\mathcal{NP}$-hard to approximate maximum independent set to within $n^{1-\epsilon}$.*

Combining the above ideas we establish the inapproximability of MWCP in $4$ dimensions and higher.

**Theorem 14.** *For any $\epsilon > 0$ it is $\mathcal{NP}$-hard to approximate* MWCP *in $4$ dimensions (or higher) with weights $\{+1, -1\}$ to within $n^{1/2-\epsilon}$.*

*Proof.* Let $\mathcal{G}$ be the family of all finite graphs. By Lemma 12 for every $G \in \mathcal{G}$ a polytope embedding of $G$ in polynomial time and with polynomial bit complexity can be found (recall that the embedding is allowed to have extra edges compared to $G$). By Lemma 9, there is a strict reduction from maximum independent set on general graphs to MWCP with weights $\{+1, -1\}$. Since Theorem 13 is expressed in terms of input size, it is left to observe that the reduction of Lemma 9 produces instances of MWCP with the number of points that is at most quadratic in the number of vertices of the input graph. □

## 3.4 Discussion

We showed that MWCP in $3$ dimensions is $\mathcal{NP}$-hard, and in $4$ dimensions and higher, it becomes $\mathcal{NP}$-hard to approximate. A natural question that arises is whether it is possible to achieve a constant-factor approximation. One reasonable direction is to explore whether polytopes with a constant number of vertices can provide a guaranteed constant approximation. However, as the following result demonstrates, even in $2D$, such solutions cannot guarantee a constant approximation.

**Theorem 15.** *By restricting solutions to polytopes with constant number of vertices one can not achieve a constant factor approximation for* MWCP *even in* $\mathbb{R}^2$ *and even for* $\{+1, -1\}$ *weights.*

*Proof.* Let $P$ be a regular $n$-gon and let the weight of each vertex be $+1$. Put a vertex with weight $-1$ outside of $P$ on the perpendicular bisector of each edge of $P$ at $\epsilon$-distance away from the edge. Choose $\epsilon$ to be a sufficiently small distance such that the line segments joining consecutive negative points cross $P$. This defines the instance of MWCP with $P$ being an optimal solution of weight $n$.

Let $v_1, v_2, \ldots, v_n$ and $u_1, u_2, \ldots u_n$ be vertices of the clockwise order of $S^+$ and $S^-$, respectively, such that $u_i$ has $\epsilon$-distance with the edge between $v_i$ and $v_{i+1}$ ($v_{n+1} := v_1$).

Let $C$ be a convex $k$-gon, we claim $w(C) \leq k$. Observe that what makes this claim non-trivial is that we cannot assume that $vert(C) \subseteq S^+$ as in Lemma 6, since we have an additional restriction of exactly $k$ vertices.

$\overline{C \backslash P}$ (the closure of $C \backslash P$) is a set of vertices, edges and non-convex polygons. Let $C'$ be one of these non-convex polygons. It suffices to show $w(C') \leq |vert(C) \cap vert(C')|$. Without loss of generality suppose $vert(C') \cap S^+ = \{v_1, v_2, ..., v_r\}$.

Let *outer negative points* be the set $\{u_{i_1}, u_{i_2}, \ldots, u_{i_\ell}\} \subseteq \{u_1, u_2, \ldots u_{r-1}\}$ such that for every $1 \leq j \leq \ell$, $u_{i_j} \notin C'$. For each $1 \leq j \leq \ell$ associate $u_{i_j}$ to the edge $e$ of $C'$ that crosses the shortest line between $u_{i_j}$ and $P$. By the choice of $\epsilon$ two vertices of $e$ are in $vert(C') \cap vert(C)$ and no edge is associated to more than one outer negative point. Thus $|vert(C') \cap vert(C)| \geq \ell + 1$. On the other hand there is at most $r$ positive and at least $r - 1 - l$ negative points in $C'$ thus $w(C') \leq l + 1 \leq |vert(C') \cap vert(C)|$. □

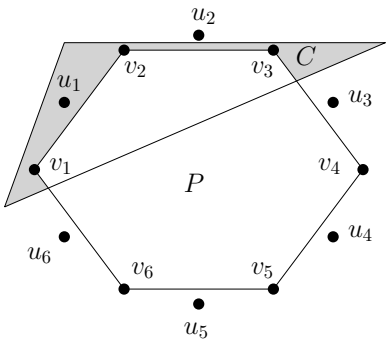



Figure 3.5: Illustration of the proof of Theorem 15. Here, $n = 6, k = 3$, we chose $C$ to result only in a single $C'$, which is shown as a shaded area. We have $l = 1$ with $u_{i_1} = u_2$ and vertex $u_2$ is associated with the topmost edge of $C'$. We have $w(C') = w(v_1) + w(v_2) + w(v_3) + w(u_1) = 3 - 1 = 2 = l + 1$.

# Chapter 4

# Weighted Online Non-Crossing Matching

In this chapter[1], we introduce and study the following problem, which we call Online Weighted Non-Crossing Matching (OWNM). Suppose $2n$ points $p_1, \ldots, p_{2n}$ in Euclidean plane arrive online one-by-one. When $p_i$ arrives, its positive weight $w(p_i) \in \mathbb{R}_{>0}$ is revealed and an algorithm has an option of matching $p_i$ to one of the unmatched previously revealed points, or leaving $p_i$ unmatched. In the vanilla online model, the decisions of the algorithm are irrevocable. There is a non-crossing constraint, which requires that the straight-line segments corresponding to the edges of the matching do not intersect. Assuming that the points are in general position, the goal is to design an algorithm that maximizes the weight of matched points.

## 4.1 Introduction

The interest in geometric settings, particularly the Euclidean plane setting, for the matching problem stems from applications in image processing [19] and circuit board design [36]. In such applications, one is often required to construct a matching between various geometric shapes, such as rectangles or circles, representing vertices, using straight-line segments or, more generally, curves. Geometry enters the picture due to constraints on the edges, such as avoiding intersections among

[1]The results presented in this chapter are based on joint work by the author in collaboration with Joan Boyar, Shahin Kamali, Kim S. Larsen, Yaqiao Li, and Denis Pankratov [17].

the edges, as well as avoiding edge-vertex intersections. These constraints can have a significant impact on the offline complexity of the problem, often resulting in variants of the problem that are $\mathcal{NP}$-hard (see the survey by Kano and Urrutia [41]).

The unweighted version of the Non-Crossing Matching problem (i.e., when $w(p_i) = 1$ for all $i \in \{1, \ldots, 2n\}$) has been studied both in the offline setting ([6, 35]) and the online setting ([15, 57, 40, 53]). See Chapter 5 for more details on the unweighted version. For now, it suffices to observe that an offline algorithm that knows the locations of all the points in advance can match all the points while satisfying the non-crossing constraint (Fact 1). Thus, the value of offline OPT is always $W := \sum_{i=1}^{2n} w(p_i)$. As discussed in Chapter 2, the performance of an online algorithm is measured by its competitive ratio, which, for our problem, represents the fraction of $W$ that the algorithm can guarantee to achieve in the worst case.

It is relatively easy to see that when there are no restrictions on the weights of points, no deterministic online algorithm can guarantee a non-trivial competitive ratio bounded away from $0$ (in particular, this is an immediate corollary of Theorem 18). We study different regimes under which the problem admits algorithms achieving non-trivial competitive ratios. Our results can be summarized as follows:

- In the Restricted OWNM, we assume that the weights of points are restricted to lie in the interval $[L, U]$ for some $L \leq U \in \mathbb{R}_{>0}$ that are known to the algorithm at the beginning of the execution. Note that by scaling, we can assume that $L = 1$; thus, without loss of generality, we assume that all the weights are in the interval $[1, U]$ in Restricted OWNM. We show that the competitive ratio of any deterministic online algorithm is $O\left(2^{-\sqrt{\log U}}\right)$ (Theorem 18). We also present a deterministic online algorithm, Wait-and-Match (WAM), which has competitive ratio $\Omega\left(2^{-2\sqrt{\log U}}\right)$ (Theorem 22).

- We show, perhaps surprisingly, that randomization alone is enough to guarantee a constant competitive ratio for arbitrary weights. We present a simple randomized online algorithm, called Tree-Guided-Matching (TGM), and prove that it has a competitive ratio of $1/3$ (Theorem 27). We supplement this result by showing that no randomized online algorithm can achieve a competitive ratio better than $8/9$, even for the unweighted version of the problem

(Theorem 26).

- We show that allowing revocable acceptances (see the beginning of Section 4.5 for the definition of the model) permits one to obtain competitive ratio $\approx 0.2862$ by a deterministic algorithm even when the weights of points are unrestricted (Theorem 30). We supplement this result by showing that no deterministic algorithm with revoking can achieve a competitive ratio better than $2/3$ (Theorem 28).

- We also study this problem when the points are not in general position but are all located on a line (see Section 4.6). We show that, even in the unweighted case and with all points on a line, neither revoking nor randomization alone helps achieve a non-trivial competitive ratio. However, we present a randomized algorithm with revoking that achieves a competitive ratio of $0.5$ in the unweighted version.

## 4.2 Preliminaries

The input to the matching problems considered in this chapter is an online sequence $I = (p_1, \ldots, p_{2n})$ of points in general position, except in Section 4.6. Each point $p_i$ has a positive real-valued weight $w(p_i) \in \mathbb{R}_{>0}$. We use $W$ to denote the total weight of all the points, i.e., $W = \sum_{i=1}^{2n} w(p_i)$. For the Restricted OWNM, the weights are assumed to lie in the interval $[1, U]$ for some known value of $U$, which is considered to be a hyper-parameter and not part of the input. Upon the arrival of $p_i$, an online algorithm must either leave it unmatched or match it with an unmatched point $p_j$ ($j < i$), in which case the line segment $\overline{p_i p_j}$, must not cross the line segments between previously matched pairs of points. The objective is to maximize the total weight of matched points. For an online algorithm ALG (respectively, offline optimal algorithm OPT), we use ALG$(I)$ (respectively, OPT$(I)$) to denote the total weight of points matched by the algorithm on input $I$.

## 4.3 Deterministic Algorithms for Restricted OWNM

We begin this section with a simplified version of the problem that involves only two different weights, illustrating the intuition behind the main algorithm and the upper bound proof. Next, we introduce a classification of points by their weights, which is used in both the positive and negative results presented later.

### 4.3.1 Warm Up

As a warm-up, we firstly consider the case where points can have only weight 1 or weight $U > 1$. Our later algorithm and analysis are inspired by the two-weight case. Also, as we shall see in this case our algorithm achieves a tight competitive ratio $1/3$ as $U \to \infty$.

We consider the following algorithm that we call "Two-Weight-Matching" (TWM). Assume the points appear in a bounding box, $B$. Throughout its execution, TWM maintains a "convex partitioning" of $B$. Initially, there is only one region formed by the entire $B$. The algorithm matches two points only if they appear in the same convex region. Whenever two points in a convex region $R$ are matched, the line segment between them is extended until it hits the boundary of $R$, which results in partitioning $R$ into two smaller convex regions.

When a point $p$ arrives in a region $R$ in which there are already some unmatched points, the algorithm TWM matches $p$ with another point in the following cases: (1) if $p$ has weight $U$, then $p$ matches with a point $q$ of the largest weight; (2) if $p$ has weight 1, and if there is a point of weight $U$ in $R$, $p$ matches with that point; (3) otherwise, i.e., $p$ has weight 1 and all unmatched points in $R$ have weight 1, then $p$ matches with a point $q$ in $R$ if there is at least one point in each side of $\overline{pq}$.

**Observation**: it is an invariant during the execution of TWM that if a region contains more than one point, all the points have weight 1, and that no region contains more than three unmatched points.

**Theorem 16.** *For the two-weight OWNM, assume $U \geq 3$ and at least one point of weight $U$ is matched, then, the competitive ratio of* TWM *is at least* $1/3$.

*Proof.* At the completion of TWM, let $m_1$ be the number of points of weight 1 that are matched, and $m_U$ be the number of points of weight $U$ that are matched. Thus, TWM $= m_1 + U m_U$. Furthermore,

let $r_1$ be the number of regions with at least one unmatched point of weight 1, and $r_U$ be the number of regions with a single unmatched point of weight $U$. By the observation we made before, these are all the possible regions containing unmatched points, and these two types of regions are *disjoint*.

Since it takes two matched points to divide a region into two, $r_1 + r_U \leq \frac{m_1+m_U}{2} + 1$, so $r_1 \leq \frac{m_1+m_U}{2} + 1 - r_U$.

Assume that there is a unmatched point of weight $U$ in region $R_1$ and let $R_2$ be the other region that was created at the same time as $R_1$ by matching two points $p$ and $q$. If both $p$ and $q$ have weight 1, then both $R_1$ and $R_2$ contain at least one point of weight 1 and, therefore, $R_1$ cannot have an unmatched point of weight $U$, which is a contradiction. So, either $p$ or $q$ has weight $U$, hence, $r_U \leq 2m_U$.

In the worst case, OPT matches all points, so

$$
\begin{aligned}
\text{OPT} &\leq m_1 + Um_U + 3r_1 + Ur_U \\
&\leq m_1 + Um_U + 3(\frac{m_1+m_U}{2} + 1 - r_U) + Ur_U, \qquad \text{by the above} \\
&= m_1 + Um_U + 3(\frac{m_1+m_U}{2} + 1) + (U-3)r_U \\
&\leq m_1 + Um_U + 3(\frac{m_1+m_U}{2} + 1) + (U-3)2m_U, \qquad \text{since } U \geq 3 \\
&\leq \frac{5}{2}m_1 + 3Um_u + 3\frac{m_U}{2} + 3 - 6m_U \\
&\leq 3m_1 + 3Um_u = 3\text{TWM}. \qquad \text{since } m_U \geq 1
\end{aligned}
$$

□

We now show that no algorithm is better than $1/3$-competitive, for arbitrarily large $U$.

**Theorem 17.** *For the two-weight OWNM, the competitive ratio of any deterministic online algorithm is at most* $1/3 + 2/(3U+3)$.

*Proof.* All points given by the adversary in this proof are given on a circle and all chords are non-crossing.

Note that since all points are given on the circle, no points given in one region can be matched with any points from other regions. For a given algorithm, ALG, the adversary gives $2k$ points of weight 1. ALG matches $s$ pairs, creating $s$ chords. If $s < \frac{k}{3}$, the sequence ends, and we have a ratio

of $\frac{2s}{2k} < \frac{1}{3}$. Otherwise, we consider $s$ of the $s+1$ regions ALG has divided the circle into separately. We ignore one region, so that we can associate exactly one chord with each of the remaining $s$ regions. This is safe since the ignored region does not have any chord associated with it, so ALG has not matched anything in that region.

Let $S_4$ denote the regions with at least four unmatched points, and let $S_0$, $S_1$, $S_2$, and $S_3$ denote the regions with 0, 1, 2, or 3 points, respectively. We treat a number of cases below. When we have treated a case, we list the profit of both ALG and OPT in parenthesis. For a given region, we count the two points that are matched by the chord associated with the region. Thus, both ALG and OPT always get a profit of at least two in any region (except the one we ignore). OPT can of course process the points offline, matching all points, except possibly one, if the number of points is odd (such a missing point can only give rise to an additive constant, so it is not relevant for the asymptotic competitive ratio). However, for the proof, we count and compare the weighted points for both ALG and OPT in each region separately.

No further points are given to the $S_4$-regions (ALG: 2, OPT: at least 6).

For the $S_0$-regions, the adversary gives one point of weight $U$ that ALG cannot match (ALG: 2, OPT: $U+2$).

For the $S_1$-regions, the adversary gives one point of weight $U$. If ALG does not match the now two points, no further points are given to that region (ALG: 2, OPT: $U+3$). Otherwise, two points of weight $U$ are given on either side of the line representing the latest match, and ALG cannot match these (ALG: $U+3$, OPT: $3U+3$).

For the $S_2$-regions, the adversary gives one more point of weight 1. If ALG does not match this point to any of the two others, we move to the $S_3$ case. Otherwise, ALG has created an empty region and the adversary gives a point of weight $U$ in that region that ALG cannot match (ALG: 4, OPT: $U+5$).

Finally, for $S_3$-regions, the adversary gives one more point of weight 1. If ALG does not match that point, no further points are given, and we are in the $S_4$ case. Otherwise, if ALG creates a region with no points, the adversary gives a point of weight $U$ in that region that ALG cannot match (ALG: 4, OPT: $U+6$). Otherwise, it has created two regions with one unmatched point each. We treat those two independently as the $S_1$ case.

Having treated the cases, we observe that for large $U$, the largest ratio between ALG and OPT is $\frac{U+3}{3U+3} = \frac{1}{3} + \frac{2}{3U+3}$. □

### 4.3.2 Point Classification.

In both lower and upper-bound arguments, we use a point classification, based on parameters, $k \in \mathbb{N}$ and $U \in \mathbb{R}$, which we explain here. Let $k = \lceil \sqrt{\log U} \rceil$, and define values of $a_0, a_1, \ldots, a_k$ so that

$$a_0 = 1, a_k = U, \;\; r = a_1/a_0 = a_2/a_1 = \ldots = a_k/a_{k-1},$$

which implies that $r = U^{1/k} = 2^{\frac{\log U}{k}} \leq 2^{\sqrt{\log U}}$ and $a_i = r^i$. For a given value $w \in [1, U]$, define $\lfloor\!\lfloor w \rfloor\!\rfloor$ as the largest $a_i$ such that $a_i \leq w$. In what follows, a point with weight $w$ is said to have type $i$ if $\lfloor\!\lfloor w \rfloor\!\rfloor = a_i$. Thus, there are $k+1$ distinct types, with type $k$ containing only the value $U$. The type of a line segment between two matched points $p$ and $q$ is defined by the type of the end-point with larger weight, that is, $\overline{pq}$ has type $i$ if one of its endpoints has type $i$ and the other endpoint has type at most $i$.

### 4.3.3 Negative Result

**Theorem 18.** *For a sufficiently large value of $U$, the asymptotic competitive ratio of any deterministic online algorithm for the Restricted* OWNM *problem is $O\left(2^{-\sqrt{\log U}}\right)$.*

*Proof.* Let ALG be any online deterministic algorithm. We use an adversarial argument. The adversary sends all points on a circle $C$, so any match the algorithm makes creates a chord in the circle. We keep a convex partitioning of the plane by the segments created by ALG. Note that since points are all in a convex position, ALG cannot match two points in different regions, otherwise it would create an intersection with existing chords. The adversary's strategy is to maintain a mapping from unmatched points to matched points to ensure the ratio between the total weight of matched points and unmatched points is $O\left(2^{-\sqrt{\log U}}\right)$. Note that this implies the ratio between the total weight of matched points and all points is also $O\left(2^{-\sqrt{\log U}}\right)$.

The adversary starts the input with an arbitrarily large number, $m$ (this is required to guarantee that our bound is asymptotic). It puts points of weight 1 in arbitrary positions on $C$ until either the

algorithm matches $m$ pairs of points or it reaches $m2^k$ points on the circle. In the latter case, the competitive ratio is at most $O(2^{-k}) = O(2^{-\sqrt{\log U}})$.

Therefore, we may assume that ALG eventually matches $m$ pairs of points, creating non-intersecting chords, and $m + 1$ regions. Now, make each matched pair responsible for a distinct region created, though with the first matched pair being responsible for two regions, initially the first two regions. Suppose a new chord $\overline{pq}$ divides region $R$ into two. Let $\{p_R, q_R\}$ be the responsible pair for $R$, $R_1$ be the side of $R$ that has $\overline{p_R q_R}$ on its boundary and $R_2$ be the other side. Leave $\{p_R, q_R\}$ responsible for $R_1$ and make $\{p, q\}$ responsible for $R_2$. This ensures that each matched pair is responsible for at least one region.

For each region $R$, the adversary makes $R$ the active region, runs the following procedure and continues with the next region until it covers all the regions. Let $\{p_R, q_R\}$ be the responsible pair of points for $R$. Consider the following two cases, depending on the number of unmatched points in $R$:

**Case 1.** If the number of unmatched points in $R$ is greater or equal to $2^k - 1$, the adversary does not send any point in $R$ and continues to the next region. In this case, we map the unmatched points in $R$ to the matched pair $\{p_R, q_R\}$. Note that $2^k - 1$ points of weight 1 are mapped to a segment of total weight 2. The ratio between the weight of matched points to the unmatched points will be $\leq 2/(2^k - 1) \in O\left(2^{-\sqrt{\log U}}\right)$.

**Case 2.** If the number of unmatched points in $R$ is less than $2^k - 1$, the adversary plans to send a sequence of points, $P = (p_1, p_2, \ldots, p_k)$, with weights $a_1, a_2, \ldots, a_k$ (respectively), one point from each weight, in the ascending order of their weights, in the following manner, (see Fig. 4.1). The point $p_1$ of weight $a_1$ appears in an arbitrary position in $R$ (on the circle). Upon the arrival of a point $p_i$ with weight $a_i$ ($i \in \{1, \ldots, k\}$), either ALG matches it with a point of weight 1 or leaves it unmatched. In the latter case, the adversary does not send more points in $R$ and continues with the next region.

In the former case, when ALG matches a point $p_i$ of weight $a_i$ with a point $q$ of weight 1, make the side of $\overline{p_i q}$ that contains at most half of the unmatched points, the active region. The adversary continues putting the remaining points of $P$ in the active region. Thus the unmatched points on the opposite side of $\overline{p_i q}$ stay unmatched, since $\overline{p_i q}$ is between the new point and those unmatched

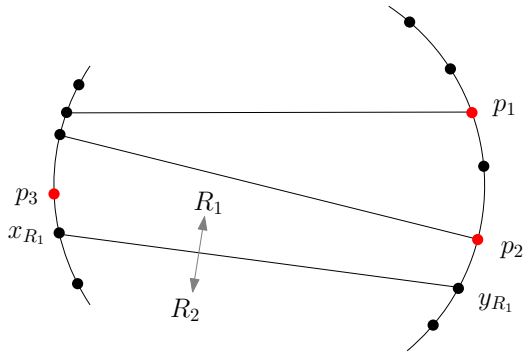



Figure 4.1: An illustration of the adversary's strategy for $k = 3$. The two arcs form the active region. Black points have weight 1. Suppose in the first phase ALG matched $x_{R_1}$ and $y_{R_1}$, which became responsible for $R_1$ region. Note that the number of unmatched points (of weight 1) in $R_1$ is 6, which is less than $2^k - 1 = 7$. Thus, in the second phase, the adversary plans to send points $p_1, p_2, p_3$ of weights $a_1, a_2, a_3$ in $R_1$. Suppose ALG matches the point of weight $a_1$; then the adversary sends $p_2, p_3$ below the line segment between the matched pair (there are fewer unmatched points there). Similarly, after the point $p_2$ of weight $a_2$ is matched, the adversary sends $p_3$ to the side of the resulting segment with no unmatched points. This ensures that some point of weight $a_i$ (here $a_3$) stays unmatched and is mapped to the matched pairs.

points.

Therefore, after matching $p_i$ and $q$, the number of unmatched points of weight 1 that can match with future points in $P$ decreases by a factor of at least 2. Let $p_j$ be the first point in $P$ that the algorithm leaves unmatched. Given that the adversary can send up to $k$ points, and there are initially less than $2^k - 1$ unmatched points in $R$, there exists such $p_j$ of weight $a_j$. At this point, the adversary ends the procedure for $R$ and continues with the next region.

The total weight of points in matched pairs in $R$ before the arrival of $p_j$ is:

$$M = \underbrace{2}_{\text{for}(p_R, q_R)} + \underbrace{j-1}_{\text{endpoints of weight 1}} + \underbrace{a_1 + a_2 + \ldots + a_{j-1}}_{\text{endpoints with weight } a_i} \leq 2\frac{a_j - 1}{r - 1}.$$

Given that the unmatched point $p_j$ is of weight $a_j$, the ratio between the weight of matched points and unmatched points is at most $M/a_j \in O(1/r) = O\left(2^{-\sqrt{\log U}}\right)$.

Given that each matched pair is responsible for at least one region, the above procedure creates a mapping of matched points to unmatched points with a weight ratio of $O(2^{-\sqrt{\log U}})$ in all cases, as desired. This finishes the proof. □

### 4.3.4 Positive Result: The Wait-and-Match Algorithm

Now we propose the main algorithm called "Wait-and-Match" (WAM) (see Algorithm 1) for the general case where weights range between 1 and $U$. We keep a convex partitioning of the plane by matched pairs and only match points that are in the same region. We use the same point classification as defined in Section 4.3.2.

Suppose a new point $p$ appears, and let $R$ denote the convex region of $p$. In deciding which point to match $p$ to (if any), the algorithm considers all unmatched points in $R$ in the non-increasing order of their weights. Let $q$ be the next point being considered, and let $i$ be the maximum of the type of $p$ and the type of $q$. The algorithm matches $p$ with $q$ if there are at least $2^{k-i} - 1$ unmatched points on each side of $\overline{pq}$. If all points in $R$ are examined, and no suitable $q$ exists, $p$ is left unmatched.

**Algorithm 1** $WaitAndMatch$

```
procedure WaitAndMatch
    while receive a new point p_j do
        Find R ∋ p_j.
        P = R ∩ {p_1, ..., p_n}
        Sort points in P in descending order by their weights.
        for p in P do
            Let i be the type of p and p_j.
            if If there are at least 2^{k-i} − 1 point in each side of p̄p̄_j then
                Match p and p_j.
                Divide R by p̄p̄_j.
                Break
            end if
        end for
    end while
end procedure
```

**Example:** Suppose $k = 2$. Then $a_0 = 1, a_1 = \sqrt{U}$, and $a_2 = U$. Let $p$ be a point with weight 1. Upon the arrival of a point $p$, the algorithm matches $p$ with any point $q$ of weight $U$ when there are at least $2^{2-2} - 1 = 0$ points on each side of $\overline{pq}$. That is, if there is an unmatched point of weight $U$ in the region, the algorithm would match $p$ to it unconditionally. Similarly, if there are no unmatched points of weight $U$ in the region, the algorithm tries to match $p$ with any point $q$ of weight $[a_1 = \sqrt{U}, a_2 = U)$ provided there is at least $2^{2-1} - 1 = 1$ point on each side of $\overline{pq}$.

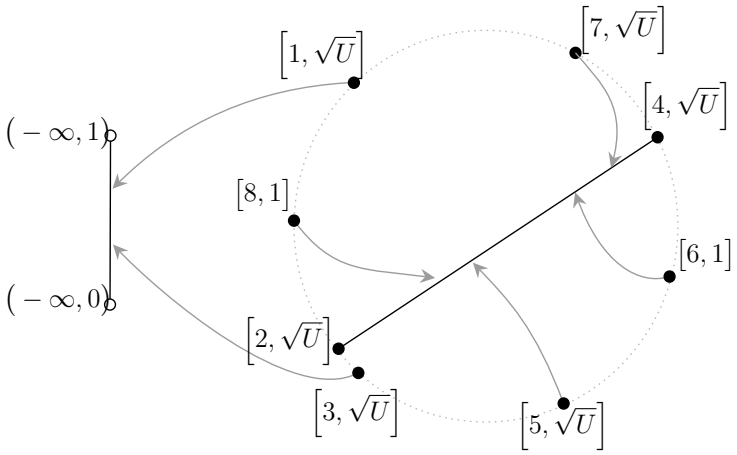



Figure 4.2: An illustration of the mapping used to analyze WAM. In this example, we have $k = 2$ and 8 points with weights in $\{1, \sqrt{U}, U\}$. Here, $[t, w]$ indicates the $t^{\text{th}}$ point in the input sequence having weight $w$. Note that points 1 and 3 are mapped to the segment corresponding to the imaginary points $(-\infty, 0)$ and $(-\infty, 1)$ of weight $U$.

Finally, if previous scenarios do not occur, the algorithm tries to match $p$ with any point $q$ of weight $[a_0 = 1, a_1 = \sqrt{U})$ provided there are at least $2^{2-0} - 1 = 3$ unmatched points on each side of $\overline{pq}$. This will happen if there were at least 7 unmatched points in the region.

To analyze the algorithm, we match each unmatched point into a matched pair. For the sake of analysis, we introduce two "imaginary" points $(-\infty, 0)$ and $(-\infty, 1)$ of weight $U$ and treat them as if they were matched before the input sequence is revealed. Suppose a new point, $p$, arrives in a region $R$ that is not matched. In this case, we map $p$ to the most recent segment that forms a boundary of the region $R$. See Figure 4.2 for an illustration of this mapping.

**Lemma 19.** *Every point of type $i$ is mapped to a segment of type $j \geq i$.*

*Proof.* For the sake of contradiction, suppose a point $p$ with type $i$ arrives in the region $R$ and gets mapped to $\overline{p_R q_R}$ of type $\leq i - 1$.

Without loss of generality, assume $p_R$ arrived after $q_R$. By the definition of the algorithm, at the time $p_R$ appeared, there were at least $2^{k-i+1} - 1$ unmatched points in $R$ (otherwise, $p_R$ would not have been matched with $q_R$). These unmatched points are still unmatched when $p$ appeared (otherwise, $R$ should have been partitioned, and $p$ should have been mapped to some other segment). Thus, when $p$ appeared, the algorithm could match it with the point that bisects these unmatched points, and there would be at least $(2^{k-i+1} - 2)/2 = 2^{k-i} - 1$ points on each side of the resulting

line segment. This contradicts the fact that the algorithm left $p$ unmatched. □

**Lemma 20.** *Let $s$ be any line segment between two matched points. For any $i$, at most $2^{k-i+2}-2$ unmatched points of type $i$ are mapped to $s$.*

*Proof.* For the sake of contradiction, assume at least $2^{k-i+2}-1$ points of type $i$ are mapped to $s$. Then, there must be at least $\lceil (2^{k-i+2}-1)/2 \rceil = 2^{k-i+1}$ points of type $i$ in a convex region $R$ formed by extending $s$. At the time the last of these points, say $p$, arrives, it could be matched to the point $q$ that bisects the other points; there will be at least $(2^{k-i+1}-2)/2 = 2^{k-i}-1$ points on each side of $\overline{pq}$. Since $\overline{pq}$ is of type $i$, the algorithm must have matched $p$ with $q$, which contradicts the fact that $p$ and $q$ are unmatched and mapped to $s$. □

**Lemma 21.** *Assuming $U$ is sufficiently large, the total weight of unmatched points mapped to a segment of type $j$ is at most $a_{j+1}2^{k-j+3}$.*

*Proof.* Note that a point of type $i$ has weight at most $a_{i+1} = r^{i+1}$. Hence, by Lemma 19 and Lemma 20, the total weight of unmatched points mapped to a segment of type $j$ is at most

$$\sum_{i=0}^{j} r^{i+1} 2^{k-i+2} = 2^{k+2} r \sum_{i=0}^{j} \left(\frac{r}{2}\right)^i = 2^{k+2} r \frac{(r/2)^{j+1}-1}{r/2-1} \le a_{j+1} 2^{k-j+3}. \qquad \square$$

Here, we assumed that $r \ge 4$, which holds for a sufficiently large $U$.

**Theorem 22.** *The competitive ratio of the deterministic online algorithm* WAM *for the Restricted* OWNM *problem is $\Omega\left(2^{-2\sqrt{\log U}}\right)$.*

*Proof.* For every matched pair $\overline{pq}$ by WAM consider the set of points formed by $p$, $q$, and the unmatched points mapped to them. By Lemma 21, if $\overline{pq}$ has type $j$, the ratio of the weight of the matched pair over all the points in this set is at least $\frac{a_j}{2a_j + a_{j+1}2^{k-j+3}} \ge \frac{1}{r2^{k+4}}$.

Since the algorithm WAM guarantees that every unmatched point is mapped to some matched pair, the competitive ratio of WAM is at least $2^{-(2k+4)}$, where we used $k = \lceil \sqrt{\log U} \rceil$ and $r = U^{1/k} = 2^{(\log U)/k} \le 2^k$. □

## 4.4 Randomized Algorithms

### 4.4.1 Negative Result

To bound the competitive ratio of randomized algorithms, we will use Yao's minimax principle for online algorithms defined as follows.

**Theorem 23** (Yao's Minimax Principle)**.** *Let $\mathcal{A}$ be the set of all deterministic algorithms and $\mathcal{I}$ be a random input sequence for an online maximization problem. For a deterministic algorithm $\text{ALG} \in \mathcal{A}$, let $\rho(\text{ALG}, \mathcal{I})$ represent the expected competitive ratio of ALG on $\mathcal{I}$. The competitive ratio of any randomized algorithm $\text{ALG}_R$ in the adversarial model, denoted by $\rho(\text{ALG}_R)$, is at most the expected competitive ratio of the best deterministic algorithm on $\mathcal{I}$:*

$$\rho(\text{ALG}_R) \leq \max_{\text{ALG} \in \mathcal{A}} \rho(\text{ALG}, \mathcal{I}).$$

We now aim to provide a random input sequence and establish an upper bound for all deterministic algorithms. By Yao's minimax principle, this will yield an upper bound on the competitive ratio for all randomized algorithms. We generate a randomized unweighted input similar to the one used for the advice model in Chapter 5. We consider a circle and generate points on the circumference of this circle. The segments of the circle bounded by two consecutive points are called *arcs*. For a point $p$, let the left and the right arcs of $p$ be the clockwise and counter-clockwise arcs that are bounded by $p$ respectively.

Put $p_1$ and $p_2$ on two arbitrary antipodals of the circle, creating two arcs. Make $p_2$ the current *active* point. At each step, we choose one of the arcs of the current active point randomly and then we put the next active point on that arc. To deceive the algorithm, sometimes we generate a *fake* point on one of the arcs of the active and then put the next active point on the other arc.

Consider two sequences $L_1, \ldots, L_{2n}$ and $F_1, \ldots, F_{2n}$ of Bernoulli i.i.d. random variables with parameter $1/2$. Iterate the following procedure to make $2n$ points. Let $p_i$ be the current active point, $L_i$ determines the position of $p_{i+1}$. If $L_i$ is 1, put $p_{i+1}$ in the middle of the left arc of $p_i$, and if $L_i$ is 0, put it in the middle of the right arc of $p_i$.

Given $p_i$ is an active point, if $F_{i+1}$ is 1, the point $p_{i+1}$ becomes fake point. Make $p_{i+2}$ the next

active point and put it in the middle of the other arc of $p_i$ (e.g. if $p_{i+1}$ is on the left arc of $p_i$, put $p_{i+1}$ on the right arc of $p_i$). If $F_{i+1}$ is 0, make $p_{i+1}$ the new active point. Continue the procedure with the new active point. First, we present the analysis from [17], followed by the improved bound using the same input, applying Wald's inequality.

**Theorem 24.** *No randomized online algorithm can achieve a competitive ratio better than* $16/17$ *in expectation.*

*Proof.* We aim to bound the competitive ratio of any deterministic algorithm on the described input sequence. Fix a deterministic algorithm ALG. Segments of matched points by ALG divide the circle into convex regions. If an unmatched point is in a region that no new points arrive in, it cannot be matched anymore and we call it an *isolated* point. Given that ALG matches $p_i$ upon its arrival, let $X_i$ be the indicator random variable that $p_i$ is an active point, and matching it causes at least one point to become isolated. Let $A_i$ be the indicator random variable that $p_i$ becomes an active point. For $i \geq 3$, $p_i$ is a fake point if and only if $p_{i-1}$ was an active point and $F_i$ is 1. Thus we can write $A_i$ as $1 - A_{i-1}F_i$.

Suppose $p_i$ is an active point that arrives in a convex region $R$, that ALG matches upon its arrival, splitting $R$ into $R_L$ and $R_R$, which contain the left and right arcs of $p_i$, respectively. If $R_L$ and $R_R$ are both empty, meaning they do not contain any unmatched point, $p_{i+1}$ becomes isolated if it is a fake point. If $R_L$ and $R_R$ are both non-empty and the point $p_{i+1}$ becomes an active point, then the unmatched points of the opposite side of $p_{i+1}$ become isolated. Now suppose $R_L$ is empty and $R_R$ is not empty and $p_{i+1}$ arrives on the left arc of $p_i$. If $p_{i+1}$ is a fake point, it becomes isolated, and if it is the new active point, unmatched points in $R_R$ become isolated. Similarly, if $R_R$ is empty and $R_L$ is not empty and $p_{i+1}$ arrives in the right arc of $p_i$, the segment $\overline{p_i p_j}$ creates isolated points. If $i = 2n$, there is no $p_{i+1}$, and matching $p_i$ makes points isolated if $R_L$ or $R_R$ are not empty. Since we are interested in the asymptotic competitive ratio we can ignore this case. Therefore, given ALG

matches $p_i$ we can write $X_i$ as follows.

$$X_i = \begin{cases} A_i F_{i+1} & \text{if } R_L \text{ and } R_R \text{ are empty} \\ A_i L_i & \text{if } R_L \text{ is empty and } R_R \text{ is not} \\ A_i(1 - L_i) & \text{if } R_R \text{ is empty and } R_L \text{ is not} \\ A_i(1 - F_{i+1}) & \text{if } R_L \text{ and } R_R \text{ are not empty} \end{cases}$$

Let the random variable $M$ be the size of the matching made by ALG, and for each $1 \leq i \leq M$, let $T_i$ be the step number in which ALG makes the $i^{\text{th}}$ match. Thus, the algorithm is guaranteed to have at least $\sum_{i=1}^{M} X_{T_i}$ unmatched points at the end of the execution. In order to bound the expectation of $M$, it may be beneficial to view it in the context of the following game. Suppose that ALG has a budget of $2n$ points. The game proceeds in rounds. In round $j$ the algorithm pays 2 points from the budget to make a guess (this corresponds to a pair of points getting matched) of a Bernoulli random variable outcome (which corresponds to ALG's match either resulting in an isolated point or not). If the guess is correct (this corresponds to $X_{T_j} = 0$, no isolated points are guaranteed to be created), then the algorithm does not pay any more points for this round. If the guess is incorrect (this corresponds to $X_{T_j} = 1$), then the algorithm pays one more point from the budget. ALG tries to maximize the total number of rounds before the budget is exhausted. Thus, in round $j$, the algorithm uses $X_{T_j} + 2$ points from the budget. Overall, $M$ is the largest integer such that $\sum_{j=1}^{M}(X_{T_j} + 2) \leq 2n$. If $X_{T_j}$ were i.i.d., we could use the renewal theorem to bound $\mathbb{E}(M)$. The issue is that $X_{T_j}$ are not i.i.d., because $X_i$ depends on $A_i$ and $F_{i+1}$; thus there are correlations between $X_i$ and $X_{i+1}$. The idea is to lower bound the expression $\sum_{j=1}^{M}(X_{T_j} + 2)$ by the sum of some i.i.d. random variables $Z_i$, compute the corresponding value of $M'$ for the $Z_i$, and then relate it back to the value of $M$.

Now we define an auxiliary random variable sequence $Y_1, \ldots, Y_M$ as follows:

$$Y_i = \begin{cases} (1 - F_{T_i})F_{T_i+1} & \text{if } R_L \text{ and } R_R \text{ are empty} \\ (1 - F_{T_i})L_{T_i} & \text{if } R_L \text{ is empty and } R_R \text{ is not} \\ (1 - F_{T_i})(1 - L_{T_i}) & \text{if } R_R \text{ is empty and } R_L \text{ is not} \\ (1 - F_{T_i})(1 - F_{T_i+1}) & \text{if } R_L \text{ and } R_R \text{ are not empty} \end{cases}$$

By replacing $A_i$ with $1 - A_{i-1}F_i$, we can see $Y_i \leq X_{T_i}$. Note that $Y_2, Y_4, \ldots, Y_{2\lfloor \frac{M}{2} \rfloor}$ are i.i.d. Bernoulli random variables with parameter $1/4$. Thus for every $m \leq M$, we can bound $\sum_{j=1}^{m}(X_{T_j}+2)$ as follows:

$$\sum_{j=1}^{m}(X_{T_j}+2) \geq \sum_{j=1}^{\lfloor m/2 \rfloor}(X_{T_{2j-1}} + X_{T_{2j}} + 4) \geq \sum_{j=1}^{\lfloor m/2 \rfloor}(Y_{2j-1} + Y_{2j} + 4) \geq \sum_{j=1}^{\lfloor m/2 \rfloor}(Y_{2j} + 4)$$

Let us define yet another auxiliary random variable sequence $Z_1, Z_2, \ldots$ as follows. For $1 \leq i \leq \lfloor \frac{M}{2} \rfloor$, let $Z_i = 4 + Y_{2i}$ and for $i > \lfloor \frac{M}{2} \rfloor$ let $Z_i = 4 + Y_i'$ such that $Y_i'$s are i.i.d. Bernoulli random variables with parameter $1/4$. This makes the $Z_i$ i.i.d. random variables that take on values of either $4$ or $5$ with probability $1/4$ and $3/4$, respectively.

Let the random variable $M'$ be the maximum $m$ such that $\sum_{i=1}^{m} Z_i < 2n$. Note that $\sum_{i=1}^{\lfloor M/2 \rfloor} Z_i = \sum_{i=1}^{\lfloor M/2 \rfloor}(Y_{T_j} + 4) \leq \sum_{i=1}^{M}(X_{T_j} + 2) \leq 2n$. Therefore $M' \geq \lfloor M/2 \rfloor$. Since the $Z_i$'s are i.i.d. and $\mathbb{E}(Z_i) = 17/4$, by the renewal theorem $\mathbb{E}(M') = 8n/17$ and therefore $\mathbb{E}(M)$ is at most $16n/17$. By Yao's minimax principle, this shows an upper bound of $16/17$ on the competitive ratio of randomized algorithms in the adversarial model.

□

Next, we demonstrate how the analysis can be refined to improve the result from $16/17$ to $8/9$ using Wald's equation. First, we need to define the notion of stopping time.

**Definition 2.** *Let $X = \{X_n : n \geq 1\}$ be a stochastic process. A stopping time with respect to $X$ is a random time such that for each $n \geq 0$, the event $\{T = n\}$ is completely determined by $\{X_1, X_2, \ldots, X_n\}$.*

**Lemma 25** (Wald's Equation)**.** *Let $T$ be a stopping time for $\{X_1, X_2, \ldots\}$ with finite expectation. Assume that for $i \leq T$ the values $X_i$ are bounded from above, or are bounded from below. If $\mathbb{E}(X_i \mid i \leq T\} = \mu$ then , $\mathbb{E}(\sum_{i=1}^{T}) = \mu\mathbb{E}(T)$.*

**Theorem 26.** *No randomized online algorithm can achieve a competitive ratio better than $8/9$ in expectation.*

*Proof.* We use the same notation as in the previous proof. Note that:

$$\sum_{i=1}^{M}(Y_i+2) \leq \sum_{i=1}^{M}(X_i+2) \leq 2n.$$

It is easy to see that $M$ is a stopping time for $(X_i+2)$'s, $M \leq 2n < \infty$, and $X_i+2 \leq 3$. Thus conditions of Wald's equation 25 are satisfied and we can use it:

$$\mathbb{E}(\sum_{i=1}^{M}(X_i+2)) = \mathbb{E}(M)\mathbb{E}(X_i+2) = \frac{9}{4}\mathbb{E}(M) \leq 2n \implies \mathbb{E}(M) \leq \frac{8}{9}n.$$

As in the previous proof, by Yao's minimax principle, this shows an upper bound of $8/9$ on the competitive ratio of randomized algorithms in the adversarial model. □

### 4.4.2 Positive Result: Tree-Guided-Matching Algorithm

We propose a randomized algorithm called "Tree-Guided-Matching" (TGM) (see Algorithm 2) that has the following uniform guarantee, regardless of the weights of the points: each point appears in a matching with probability at least $1/3$.

The algorithm TGM uses a binary tree to guide its matching decisions. The binary tree is created online, with each node of the tree corresponding to an online point. Intuitively, the binary tree, as it grows, gives an online refining of the partition of the plane into convex regions, such that for each region there is some online point *responsible* for it. Initially, set $p_1$ as the root of the tree and $p_2$ the child of $p_1$. Let $R_1$ and $R_2$ denote the two regions corresponding to the half-spaces created by $\overline{p_1p_2}$. See Figure 4.3 for an example of this process. Let $p_2$ be responsible for both $R_1$ and $R_2$. In general, when $p_i$ arrives into a region $R$ for which $p_j$ is responsible (of course, $j < i$), make $p_i$ a child of $p_j$ in the binary tree. The line $\overline{p_ip_j}$ divides the region $R$ into two sub-regions $R'$ and $R''$, let $p_i$ be responsible for both of them, and at this point the responsibility of $p_j$ on $R$ is lost as region $R$ has been refined to $R'$ and $R''$. Note that this implies every node of the tree has at most two children. Next, we describe how TGM chooses to match points. At the beginning, TGM matches $p_2$ with $p_1$ with probability $1/3$. After that, upon the arrival of $p_i$, let $p_j$ be its parent in the tree. If $p_j$ is unmatched and $p_i$ is its first child, match $p_i$ to $p_j$ with probability $1/2$. If $p_j$ is unmatched and

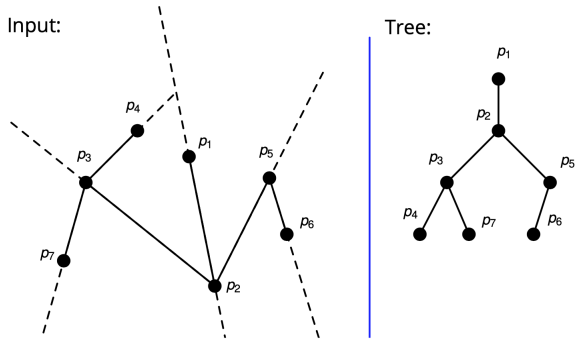



Figure 4.3: On the left is the input and how TGM divides and partitions the plane, and on the right is the tree it creates from the input. Based on this tree, it matches nodes with their parents randomly, such that every node upon its arrival gets matched with a probability of $1/3$.

$p_i$ is its second child, match $p_i$ to $p_j$ deterministically. Note that TGM only tries to match an online point with its parent in the tree.

**Algorithm 2** $TreeGuidedMatching$

```
procedure TreeGuidedMatching
    Receive p_1 and p_2.
    Match p_1 and p_2 with probability 1/3.
    Divide the plane by p_1p_2 into R_1 and R_2.
    Make p_2 the responsible point for R_1 and R_2.
    while receive a new point p_i do
        Find R ∋ p_i.
        Let p_j be responsible of R.
        if p_j is unmatched then
            if p_j has no child then
                Match p_i with p_j with probability 1/2.
            else
                Match p_i with p_j.
            end if
        end if
        Make p_i the new child of p_j.
        Divide R by p_ip_j into R_1 and R_2.
        Make p_i the responsible point for R_1 and R_2.
    end while
end procedure
```

**Theorem 27.** *Every point, regardless of its weight, is chosen into a matching by the randomized algorithm* TGM *with probability at least* $1/3$*. Hence,* TGM *achieves a strict competitive ratio at least* $1/3$*.*

*Proof.* Note that since TGM only matches a child to its parent in the binary tree, the matching is non-crossing. Indeed, by our construction of the tree, every child is a point *inside*[2] a convex region for which its parent is responsible, and its parent lies on the boundary of that region. Hence, the line segment formed by them does not cross any existing line segment.

Next, we show the claimed performance of TGM. By the definition of TGM, $p_1$ is matched (by $p_2$) with probability $1/3$. We will show that every $p_i$, $i \geq 2$, *upon its arrival* gets matched to its parent with probability exactly $1/3$, which implies the claim. To see this, proceed inductively. The base case is true for $p_2$. Let $p$ be the currently arrived point and $q$ be its parent. We consider two cases.

- If $p$ is the first child of $q$, then by the induction hypothesis $q$ at this moment is unmatched with probability $2/3$, hence according to TGM, $p$ is matched (to $q$) with probability $(2/3) \cdot (1/2) = 1/3$.
- If $p$ is the second child of $q$, then $q$ at this moment is unmatched with probability $1 - 1/3 - 1/3 = 1/3$. By TGM, $p$ is matched (to $q$) with probability $(1/3) \cdot 1 = 1/3$. □

## 4.5 Revocable Acceptances

In this section, we consider the revocable setting. When a new point $p$ arrives, an algorithm has an option of removing one of the existing edges from the matching prior to deciding on how to match $p$. The decision to remove an existing edge is irrevocable. The benefit of making this decision is that the end-points of the removed edge, along with possible points on the other side of the edge (though our positive result does not use this possibility), become available candidates to be matched with $p$, provided the non-crossing constraint is respected.

### 4.5.1 Negative Result

[15] showed that a deterministic greedy algorithm without revoking can achieve $2/3$ competitive ratio in the unweighted version. In this section, we prove that in the unweighted version, no deterministic algorithm with revoking can beat the ratio $2/3$.

[2] Recall that online points are in general position.

**Theorem 28.** *No deterministic algorithm with revoking can achieve a competitive ratio better than* $2/3$ *even in the unweighted version.*

*Proof.* Fix a deterministic algorithm ALG, an arbitrary large $n$, and a circle in the plane. The adversary adds at least $2n$ points, all of weight 1, on the circle, one by one, and let ALG match them into pairs. We maintain the invariants that there is always one active region of the circle, and that for each matched pair, there is always at least one unmatched point.

Initially, the entire circle is the active region. A phase consists of the adversary presenting points on the circle, in the active region, until ALG either matches a pair or revokes a matching, or until $2n$ points have been given. The adversary stops if there are $2n$ points and the last point is unmatched. Otherwise, if a match has just occurred, there are two cases.

In Case 1, the current point, $p$, is simply matched to a point, $q$, on the circle. The chord $\overline{pq}$ divides the active region into two sub-regions, $R_1$ and $R_2$. If neither region has any points, add a point, $p'$, to $R_1$. Without loss of generality, assume that $R_1$ contains at least as many unmatched points as $R_2$. If $p'$ is matched, ALG has revoked a matching; and we get the extra point from Case 2. Otherwise, $R_2$ becomes the active region, some unmatched point in $R_1$ is associated with the matched pair, and the phase ends.

In Case 2, ALG revokes a matching and either matches the current point, $p$, or leaves $p$ unmatched. Removing the one match, removes a chord of the circle, joining two regions into a new convex region. This region is the active region if $p$ is not matched. In either case, the number of matched points is not increased. However, the number of unmatched points is increased by at least 1, since at least one of the points, $q$, from the revoked match is now unmatched and $p$ is only matched to one point. If there is a new match for $p$, the sub-region created by the match that does not contain $q$ becomes the active region, and $q$ is the unmatched point associated with the new matched pair. The current phase ends.

Inductively, the invariants hold after each phase, and the unmatched point associated with each matched pair ensures that no more than $2/3$ of the points are matched. Although the number of points may be odd, this gives an asymptotic lower bound of $2/3$ on the competitive ratio. □

Note that ignoring the revoking option, the above proof is a simpler alternative to bound the

competitive ratio of the deterministic algorithm which was given by Bose et al. [15].

### 4.5.2 Positive Result: Big-Improvement-Match

We present a deterministic algorithm with revoking, called "Big-Improvement-Match" (BIM) (see Algorithm 3). This algorithm has a strict competitive ratio of $\approx 0.2862$ even when weights of points are unrestricted. This shows that while revoking does not improve the competitive ratio in the unweighted version, it provides us with an algorithm with a constant competitive ratio, which is unattainable for a deterministic algorithm without revoking.

BIM maintains a convex partitioning of the Euclidean space. Each region in the partition is assigned an edge from the current matching to be responsible for that region. Each edge can be responsible for up to two regions. BIM starts out by matching the first two points, $p_1$ and $p_2$ regardless of their weights, dividing the plane into two half-planes by $\overline{p_1p_2}$. BIM then assigns $\overline{p_1p_2}$ to be responsible for the two half-plane regions. Next, consider a new point $p_i$ (for $i \geq 3$) that arrives in an existing region $R$. Suppose that $\overline{p_jp_{j'}}$ is the responsible edge for $R$. If there is at least one unmatched point in $R$, BIM matches $p_i$ with an unmatched point $p_k$ in $R$ with the highest weight. Then $\overline{p_jp_{j'}}$ is no longer responsible for $R$, and the region $R$ is divided into two new regions by $\overline{p_ip_k}$. The responsibility for both new regions is assigned to $\overline{p_ip_k}$. If $p_i$ is the only point in $R$, then BIM decides to revoke the matching $(p_j, p_{j'})$ or not as follows. Without loss of generality, assume $w(p_j) \leq w(p_{j'})$. If $w(p_i) < rw(p_{j'})$, then BIM leaves $p_i$ unmatched. Otherwise, BIM removes the matching $(p_j, p_{j'})$ and matches $p_i$ with $p_{j'}$. We note that $r$ is a parameter that is going to be chosen later so as to optimize the competitive ratio. If $R$ is the only region that $\overline{p_jp_{j'}}$ was responsible for when $p_i$ arrived, then $R$ is divided into two regions by $\overline{p_ip_{j'}}$, and $\overline{p_ip_{j'}}$ becomes responsible for the two new regions. (The regions on the other side of $\overline{p_jp_{j'}}$ from $p_i$ keep their boundaries, even though $(p_j, p_{j'})$ is no longer in the matching.) Otherwise $\overline{p_jp_{j'}}$ was responsible for $R'$ in addition to $R$ when $p_i$ arrived. In this case, after removal of the match $(p_j, p_{j'})$, regions $R$ and $R'$ are merged to give region $R'' = R \cup R'$, and $R''$ is divided by $\overline{p_ip_{j'}}$ into two regions, and BIM makes $\overline{p_ip_{j'}}$ responsible for both new regions.

**Proposition 29.** *The following observations concerning* BIM *hold:*

**Algorithm 3** $BigImprovementMatch$

**procedure** $BigImprovementMatch$(r)
  Receive $p_1$ and $p_2$.
  Match $p_1$ and $p_2$.
  Divide the plane by $\overline{p_1p_2}$ into $R_1$ and $R_2$.
  Make $\overline{p_1p_2}$ the responsible point for $R_1$ and $R_2$.
  **while** receive a new point $p_i$ **do**
    Find $R \ni p_i$.
    **if** $P := R \cap \{p_1, \ldots p_{i-1}\} \neq \emptyset$ **then**
      Match $p_i$ with $p_j$ in $P$ with the maximum weight.
      Divide $R$ by $\overline{p_ip_j}$ into $R_1$ and $R_2$.
      Make $\overline{p_ip_j}$ responsible for $R_1$ and $R_2$.
    **else**
      Let $\overline{p_jp_{j'}}$ be responsible for $R$ such that $w(p_j) \geq w(p_{j'})$.
      **if** $w(p_i) \geq r.w(p_j)$ **then**
        Let $R'$ be union of regions that $\overline{p_jp_{j'}}$ is responsible for.
        Revoke $\overline{p_jp_{j'}}$.
        Match $p_i$ with $p_j$.
        Divide $R'$ by $\overline{p_jp_{j'}}$ and make it responsible for the new subregions.
      **end if**
    **end if**
  **end while**
**end procedure**

*(1) All responsible edges are defined by two currently matched points.*

*(2) Each edge is responsible for at most two regions.*

*(3) All regions are convex.*

*(4) When a matched edge $(p_j, p_{j'})$ is replaced due to the arrival of a point $p_i$ in region $R$, then edge $(p_i, p_{j'})$ is contained in $R$.*

*Proof.* (1) follows since an edge only becomes responsible when its endpoints become matched. When another edge becomes responsible for a region, the original edge is no longer responsible. (2) follows since the only two regions an edge is made responsible for are the two regions created when the endpoints of the edge were matched. When two points in one of the regions an edge is responsible for are matched, the edge is no longer responsible for that region, but will still be responsible for one region if it had been responsible for two up until that point. (3) follows inductively, since separating two convex regions by a line segment creates two convex regions. In addition, when

BIM removes an edge, that edge was the last matching created in either of the two regions it was responsible for. (4) follows by (3). □

**Theorem 30.** BIM *with* $r \in (1, \sqrt{2}]$ *has strict competitive ratio at least* $\min\left(\frac{r^2-1}{r^3}, \frac{1}{1+2r}\right)$ *for the* OWNM *with arbitrary weights.*

*Proof.* We consider for each region an edge is responsible for, the total weight of unmatched points in that region. These points come in two flavours: those that were matched at some point during the execution, but due to revoking became unmatched, and those that were never matched during the entire execution of the algorithm.

Consider any subsequence of all created edges, $\langle e_1, \ldots, e_k \rangle$, where $e_1$ was created when a second unmatched point arrived in some region, and the possible remaining edges were created via revokings, i.e., $e_i$ caused $e_{i-1}$ to be revoked for $2 \leq i \leq k$, and $e_k$ is in BIM's final matching. Let $e_j = (p_{i_j}, p_{i_{j+1}})$ and $w(p_{i_j}) \leq w(p_{i_{j+1}})$, so $e_{j+1} = (p_{i_{j+1}}, p_{i_{j+2}})$. Thus, for $3 \leq j \leq k+1$, $p_{i_j}$ arrived after $p_{i_{j-1}}$. Every pair ever matched by BIM is included in some such sequence of edges. The points, $p_{i_1}, \ldots, p_{i_{k-1}}$ could be unmatched points in a region for which $e_k$ is responsible.

Let $\alpha = w(p_{i_k})$, so for every $2 \leq j \leq k$, $w(p_{i_j})$ is at most $\alpha r^{-(k-j)}$ and $w(p_{i_1}) \leq w(p_{i_2}) \leq \alpha r^{-(k-2)}$. Let $\beta = w(p_{i_{k+1}})$. The total weight of points in this sequence is

$$\begin{aligned}\sum_{j=1}^{k+1} w(p_{i_j}) &= w(p_{i_1}) + \sum_{j=2}^{k} w(p_{i_j}) + w(p_{i_{k+1}}) \leq \alpha r^{-(k-2)} + \alpha\left(\frac{r}{r-1}\right)(1 - r^{-(k-1)}) + \beta \\ &= \alpha\left(r^{-(k-1)}\frac{r(r-2)}{r-1} + \frac{r}{r-1}\right) + \beta.\end{aligned}$$

Now, we consider other points that were never matched, but were at some time in a region for which one of the $e_j$ was responsible. After $e_1$ is created and before $e_2$, a first point $q_1$ could arrive in one of the regions for which $e_1$ is responsible. Note that $q_1$ is not matched if $w(q_1) < rw(p_{i_2})$. (Note that a second point arriving in that region will then be matched to $q_1$, dividing the region, and the sub-regions will not be considered part of the region for which $e_k$ eventually becomes responsible.) Now, suppose that another point, $q_2$, arrives between when $e_j$ and $e_{j+1}$ are created for some $2 \leq j < k$, remaining unmatched in one of the regions for which $e_k$ is responsible. Then, neither $q_2$ nor $p_{i_{j+2}}$ is in the same region as $p_{i_{j-1}}$ or one of them would have been matched to $p_{i_{j-1}}$ (or $p_{i_{j-1}}$ was already

matched and the region divided). By Proposition 29.2, $e_i$ is responsible for at most two regions, so $p_{i_{j+2}}$ arrives in the same region as $q_2$, while unmatched. This is a contradiction, since BIM would match them. Thus, other than $q_1$, the only never-matched point, $q_2$, in a region for which $e_k$ is responsible, arrives after $e_k$ and $w(q_2) < rw(p_{i_{k+1}})$.

Then, for $k \geq 2$, the total weight of unmatched points for which $e_k$ is responsible is at most $\left(r^{-(k-3)} + r^{-(k-1)}\frac{r(r-2)}{r-1} + \frac{r}{r-1}\right)\alpha + (1+r)\beta$. If $r \leq \sqrt{2}$, then $r^{-(k-3)} + r^{-(k-1)}\frac{r(r-2)}{r-1}$ is at most zero and we can bound the total weight when $k \geq 2$ by: $(\frac{r}{r-1})\alpha + (1+r)\beta$. Thus, the ratio between the weight of matched points in sequence $p_{i_1}, p_{i_2}, \ldots, p_{i_{k+1}}$ and the total weight of all points associated with this sequence for $k \geq 2$ is at least $\frac{\alpha+\beta}{(\frac{r}{r-1})\alpha+(1+r)\beta}$. Since $\frac{r}{r-1} > 1 + r$, for $1 < r \leq \sqrt{2}$, this ratio is minimized when $\beta$ is minimized, which happens at $\beta = r\alpha$. Thus, the competitive ratio for $k \geq 2$ is at least $(1+r)/(\frac{r}{r-1} + r(1+r)) = \frac{r^2-1}{r^3}$.

Now, consider the case of $k = 1$, and let $\alpha$ and $\beta$ have the same meaning as above. Then the sequence $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ consists of a single edge. Thus, the weight of the matched points is $\alpha+\beta$, and there could be two unmatched points $q_1$ and $q_2$ at the end of the execution of the algorithm charged to this edge. We have $w(q_i) < r\beta$, so the ratio between the weight of matched points and the total weight of all points associated with the sequence in case of $k = 1$ is at least $\frac{\alpha+\beta}{\alpha+(1+2r)\beta}$. Observe that this ratio is minimized when $\beta$ goes to infinity and becomes $1/(1+2r)$.

Taking the worse ratio between the above two scenarios proves the statement of the theorem. □

**Corollary 31.** *With the choice of parameter for* BIM*, $r^*$, defined as the positive solution to the equation $\frac{1}{1+2r} = \frac{r^2-1}{r^3}$, approximately $1.2470$, we get a competitive ratio of $\frac{1}{1+2r^*}$, at least $0.2862$.*

*Proof.* The value $r^*$ is obtained by setting the two terms in the minimum in Theorem 30 equal to each other and solving for $r$, giving the lower bound on the competitive ratio.

To show that this result is tight, consider the following input: $p_1$ of weight $\alpha$ arrives at the north pole of the unit sphere, followed by $p_2$ of weight $\beta \geq \alpha$ at the south pole of the unit sphere, followed by $p_3$ of weight $r^*\beta - \epsilon$ at the west pole of the unit sphere, and followed by $p_4$ of weight $r^*\beta - \epsilon$ at the east pole of the unit sphere. The algorithm would end up matching $p_1$ with $p_2$, leaving $p_3$ and $p_4$ unmatched. Thus, in such an instance, the competitive ratio of the algorithm is $(\alpha+\beta)/(\alpha+\beta+2r^*\beta-2\epsilon)$. Taking $\beta$ to $\infty$ and $\epsilon$ to 0 shows that the algorithm does not guarantee

a competitive ratio better than $1/(1+2r^*)$ in the strict sense. □

## 4.6 Collinear Points

In this section, we investigate the non-crossing matching problem when all points lie on a line. Contrary to the general case, we will demonstrate that revoking and randomization alone do not yield any non-trivial competitive ratio, even when the points are unweighted. However, we will show that combining randomization with revoking can result in a constant competitive ratio in the unweighted case. First, we prove that the competitive ratio of any randomized algorithm (without revoking) can be arbitrarily bad.

**Theorem 32.** *No randomized algorithm can achieve a competitive ratio better than $2/n$ when points arrive in arbitrary positions, even if the points are unweighted and all lie on a line.*

*Proof.* We construct a random input, bound the competitive ratio of any deterministic algorithm on that input, and prove the theorem using Yao's minimax principle. To begin, place $p_1$ and $p_2$ in arbitrary positions on the line, then place $p_3$ between them. For each subsequent point $p_i$, randomly choose either the left or right segment relative to the last point $p_{i-1}$, and place $p_i$ in the middle of that segment. This process continues until a total of $2n$ points are positioned on the line.

If a deterministic algorithm matches $p_1$ and $p_2$, it cannot match any other points, resulting in a competitive ratio of $\frac{1}{n}$. Furthermore, whenever the algorithm matches a new arriving point, there is a $\frac{1}{2}$ probability that all future points will arrive on the same side, preventing further matches. Thus, the probability that the algorithm can match at least $k$ pairs is at most $\frac{1}{2^k}$. Consequently, the expected size of the matching created by the algorithm is bounded by $\sum_{i=0}^{\infty} \frac{1}{2^i} \leq 2$.

However, by Fact 1, a perfect matching is always possible, leading to a competitive ratio of $\frac{2}{n}$. □

Next, we show that allowing revoking alone does not result in a non-trivial competitive ratio.

**Theorem 33.** *No deterministic algorithm with revoking can achieve a competitive ratio better than $1/n$ when points arrive in arbitrary positions, even if the points are unweighted and all lie on a line.*

*Proof.* The adversary's strategy is designed to ensure that the algorithm can maintain at most one matched pair at any time. Initially, the adversary places points in arbitrary positions until the algorithm forms a match between two points. Once this match is created, the adversary starts placing new points between the matched pair, forcing the algorithm to revoke the match in order to form any new pairs.

Once the algorithm revokes the match, the adversary goes back to placing points in arbitrary positions. However, as soon as the algorithm creates another match, the adversary again places points between the matched pair, preventing any further progress. This cycle continues: the adversary alternates between placing points freely when no match exists and placing points between matched pairs once a match is formed.

This ensures that the algorithm can only ever maintain one matched pair at any given time. After $2n$ points have arrived, the algorithm will have made only one matched pair, leading to a competitive ratio of $1/n$. □

Next, we present a randomized algorithm with revoking, called "Random-Revoking-Matching" (RRM), which achieves a competitive ratio of $0.5$ (see Algorithm 4). RRM maintains a partition of the line into intervals and matches points only within those intervals. The intervals may be open, half-open, or closed. Initially, the entire line is an open interval. Throughout the execution of the algorithm, the following conditions hold for partitioning the line:

- Open and half-open intervals contain no matched pairs.
- Each half-open interval contains exactly one unmatched point, located on its closed boundary.
- Closed intervals contain exactly two points, both located on the boundaries and matched together.

These conditions hold at the beginning, with the line being one open interval, and they remain valid after each step of the algorithm. The decisions of the algorithm ensure that these conditions are preserved throughout its execution.

The first point that arrives in an open interval remains unmatched, and the partitioning does not change. When a second point arrives in an open interval, it is matched to the previously available

point. This action splits the open interval into a closed interval containing the matched points and two new open intervals on either side of the matched segment.

When a point arrives in a closed interval, RRM revokes the matched segment, randomly matches the new point to one of the now available points, and divides the interval into a closed interval containing the newly matched points and a half-open interval with the previously matched point on its closed boundary.

Finally, when a point arrives in a half-open interval, it is matched to the unmatched point on its closed boundary, splitting the interval into a closed interval between the matched points and an open empty interval (see Figure 4.6).

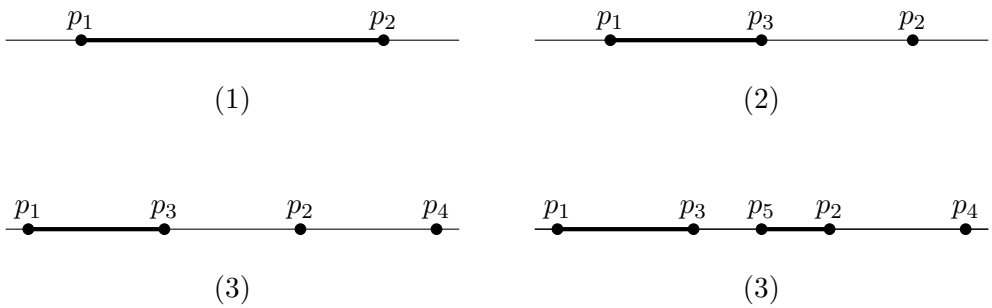



Figure 4.4: (1) The first two points arrive, partitioning the line into $(-\infty, p_1)$, $[p_1, p_2]$, and $(p_2, \infty)$. (2) Point $p_3$ arrives in $[p_1, p_2]$. RRM revokes $\overline{p_1 p_2}$, randomly matches $p_3$ to $p_1$, and partitions $[p_1, p_2]$ into $[p_1, p_3]$ and $(p_3, p_2]$. (3) Point $p_4$ arrives in $(p_2, \infty)$ and remains unmatched as the interval it arrived in is empty. (4) Point $p_5$ arrives in $(p_3, p_2]$ and is matched to $p_2$, splitting the interval into $(p_3, p_5)$ and $[p_5, p_2]$.

**Theorem 34.** *The competitive ratio of the randomized algorithm* RRM*, with access to revoking, when points are unweighted and arrive on a line, is* $0.5$.

*Proof.* Let $f(n)$, $g(n)$, and $h(n)$ represent the expected number of matched pairs when $n$ points arrive in a closed, half-open, and open interval, respectively. For $n = 0$, there are no matched pairs. By the definition of RRM, when a point arrives in an open interval, it remains unmatched, and when a point arrives in a closed interval, a matched pair is revoked and a new one is created, so the number of matched pairs does not change. However, when a point arrives in a half-open interval, RRM creates a new matching. Therefore, we have the initial conditions: $f(0) = f(1) = g(0) = h(0) = h(1) = 0$ and $g(1) = 1$.

**Algorithm 4** $RandomRevokingMatching$

**procedure** $RandomRevokingMatching$
  **while** receive a new point $p_i$ **do**
    Let $I$ be the interval that $p_i$ arrives in.
    **if** $I$ is an open interval $(x, y)$ **then**
      **if** $I$ contains another point $p_j$ **then**
        Match $p_i$ with $p_j$.
        **if** $p_i$ is on the left side of $p_j$ **then**
          Divide $I$ into $(x, p_i)$, $[p_i, p_j]$, and $(p_j, y)$.
        **else**
          Divide $I$ into $(x, p_j)$, $[p_j, p_i]$, and $(p_i, y)$.
        **end if**
      **end if**
    **else if** $I$ is a closed interval $[p_j, p_k]$ **then**
      Revoke $\overline{p_j p_k}$.
      Set $r$ randomly into 0 or 1.
      **if** $r = 0$ **then**
        Match $p_i$ with $p_j$
        Divide $I$ into $[p_j, p_i]$ and $(p_i, p_k]$
      **else**
        Match $p_i$ with $p_k$
        Divide $I$ into $[p_j, p_i)$, and $[p_i, p_k]$
      **end if**
    **else if** $I$ is half-open $[p_j, x)$ **then**
      Match $p_i$ with $p_j$.
      Divide $I$ into $[p_j, p_i]$ and $[p_i, x)$
    **else if** $I$ is half-open $(x, p_j]$ **then**
      Match $p_i$ with $p_j$.
      Divide $I$ into $(x, p_i]$ and $[p_i, p_j]$
    **end if**
  **end while**
**end procedure**

If two or more points arrive in an open interval, the first two points will be matched, dividing the interval into two open intervals and one closed interval. When $n$ points are expected to arrive in the original open interval, the new intervals will receive $k_1$, $k_2$, and $n - k_1 - k_2 - 2$ points, where $k_1, k_2 \geq 0$ and $k_1 + k_2 \leq n - 2$. Therefore, for $n \geq 2$, we have:

$$h(n) \geq 1 + \min_{\substack{k_1,k_2 \\ k_1,k_2 \geq 0 \\ k_1+k_2 \leq n-2}} \{h(k_1) + f(k_2) + h(n - k_1 - k_2 - 2)\}.$$

The first point that arrives in a closed interval revokes the existing matched segment and is

randomly matched to one of the points, dividing the interval into one half-open and one closed interval. When $n \geq 2$ points are expected to arrive, the new intervals will receive $k$ and $n-k-1$ points, where $0 \leq k \leq n-1$. Therefore, for $n \geq 2$, we have:

$$f(n) \geq \min_{0 \leq k \leq n-1} \left\{ \frac{1}{2} \left[ f(k) + g(n-k-1) \right] + \frac{1}{2} \left[ f(n-k-1) + g(k) \right] \right\}.$$

Finally, when the first point arrives in a half-open interval, it gets matched to the only available point, dividing the interval into a closed and an open interval. When $n \geq 2$ points are expected to arrive, the new intervals will receive $k$ and $n-k-1$ points, where $0 \leq k \leq n-1$. Therefore, for $n \geq 2$, we have:

$$g(n) \geq 1 + \min_{0 \leq k \leq n-1} \left\{ h(k) + f(n-k-1) \right\}.$$

Next, we will show the following inequalities using induction for $n \geq 0$:

$$f(n) \geq \frac{n-1}{4}, \qquad g(n) \geq \frac{n+1}{4}, \qquad h(n) \geq \frac{n-1}{4}.$$

It is straightforward to verify these inequalities for $n = 0$ and $n = 1$. For the induction step, for any $k, k_1, k_2$ such that $0 \leq k \leq n-1$, $k_1, k_2 \geq 0$, and $k_1 + k_2 \leq n-2$, we have:

$$h(n) \geq 1 + h(k_1) + f(k_2) + h(n-k_1-k_2-2)$$

$$\geq 1 + \frac{k_1 - 1}{4} + \frac{k_2 - 1}{4} + \frac{n-k_1-k_2-3}{4} = \frac{n-1}{4}.$$

For $f(n)$, we get:

$$f(n) \geq \frac{1}{2} \left( f(k) + g(n-k-1) \right) + \frac{1}{2} \left( f(n-k-1) + g(k) \right)$$

$$\geq \frac{1}{2} \left( \frac{k-1}{4} + \frac{n-k}{4} + \frac{n-k-2}{4} + \frac{k+1}{4} \right) = \frac{n-1}{4}.$$

For $g(n)$, we have:

$$g(n) \geq 1 + h(k) + f(n-k-1) \geq 1 + \frac{k-1}{4} + \frac{n-k-2}{4} = \frac{n+1}{4}.$$

Thus, the expected number of matched pairs by RRM when $2n$ points arrive on a line is at least:

$$h(2n) = \frac{2n-1}{4},$$

where the optimum is always $n$, which implies that RRM achieves a competitive ratio of $0.5$. $\square$

# Chapter 5

# Online Non-Crossing Matching with Advice

In this chapter[1], we study the advice complexity of the Online Non-Crossing Matching problem. We consider bichromatic and monochromatic versions.

## 5.1 Introduction

Suppose that $2n$ points in general position are revealed one by one in the Euclidean plane. When and only when a point is revealed you have a choice of matching it with a previously revealed but yet unmatched point by a straight-line segment. Each point can be matched at most one other point and the connecting segments should not intersect each other: the connecting segments should form a non-crossing matching. Each decision on how to match a point is irrevocable and the goal is to maximize the number of connected points.

It is easy to see that a perfect matching always exists (see Fact 1). However, this optimal solution requires the knowledge of the entire input sequence, and in our version of the problem the points arrive online and decisions are irrevocable. The question we are concerned with in this chapter is how much additional information an online algorithm needs so that it can achieve optimal or near-optimal performance. We study two variants of the above problem: Online Monochromatic

[1]The results presented in this chapter are based on joint work by the author in collaboration with Denis Pankratov [53].

Non-Crossing Matching (or MNM, for short) where each point can be potentially matched with any other point, and Online Bichromatic Non-Crossing Matching (or BNM, for short) where $n$ of the points are colored blue and $n$ of the points are colored red, all the blue points arrive first, and points can be matched only if they have different colors. BNM can be thought of as a bipartite variant of the MNM in the geometric setting.

Matching problems in general abstract graphs have a long history dating back to König's theorem from 1931 and Hall's marriage theorem from 1935. The offline setting of matching algorithms, where the entire input is known in advance, is fairly well understood: efficient polynomial time algorithms are known for the general and bipartite versions of the problem [24, 37, 54, 48, 49]. The online versions of the problem, where input is revealed one item at a time with the restriction of irrevocable decisions, have received a lot of attention in recent years due to applications in online advertising and algorithmic game theory (see the excellent survey by Mehta [51] and references therein). As discussed in Chapter 2, the performance of an online algorithm ALG is measured by its *competitive ratio*. The online maximum matching problem has been studied in a variety of settings, including adversarial [42, 10, 22], stochastic [7, 18, 29, 34, 39, 50], and advice [26]. The advice setting is also the focus of the present chapter, so we shall review it next.

In the online advice model, the algorithm that receives input items one by one is cooperating with an all-powerful oracle, which has access to the entire input in advance. The oracle can communicate information to the algorithm by writing on an infinite tape, which it populates with an infinite binary string before the algorithm starts its execution. At any point during its runtime, the online algorithm can decide to read one or more bits from the tape. The worst-case number of bits read by the algorithm on an input of length $n$ is its advice complexity, as a function of input length $n$. The advice can be viewed as a generalization of randomness. A randomized algorithm is given random bits (independent of the input instance), whereas an advice algorithm is given advice bits prior to processing the online input. Note that unlike the standard Turing machine model with advice, in the online setting advice string contents are allowed to depend on the entire input and not just its length. Thus, advice length can be thought of as a measure of how much extra information about the input the online algorithm needs in order to achieve a certain competitive ratio. For more information about the online advice complexity see the excellent survey by Boyar et al. [16]. This setting is

important not only from the theoretical point of view but also from a practical one, as advice-based algorithms can often lead to efficient offline algorithms when advice is efficiently computable [14]. In addition, recently a new research direction has gained a lot of momentum, where the feasibility of using advice obtained by machine learning techniques in conjunction with online algorithms is being assessed (see [1, 3] for such results related to bipartite matching and [5] references therein for a general theoretical framework).

In a variety of stochastic settings, algorithms for the general abstract online bipartite matching face the competitive ratio barrier of $1 - 1/e$. This is a consequence of a simple observation that when one throws $n$ balls into $n$ bins at random there will be roughly $n/e$ collisions, and this situation is often embedded in the online bipartite matching. In fact, the tight worst-case competitive ratio in the adversarial randomized setting is exactly $1 - 1/e$ [42]. This competitive ratio is achieved by an algorithm that uses $\Theta(n \log n)$ random bits. Miyazaki [52] showed that in order to achieve optimality with advice, one needs $\Omega(n \log n)$ bits of advice. However, if one is satisfied with getting near optimality, Dürr et al. [26] showed that one can get the competitive ratio $(1 - \epsilon)n$ using only $\Theta_\epsilon(n)$ bits of advice, where $\Theta_\epsilon$ hides constants depending on $\epsilon$.

Considerations in image processing [19] and circuit board design [36], among other applications, motivate the study of matching problems in geometric graphs. In the geometric setting restricted to 2 dimensions, one is led to consider various shapes in the Euclidean plane, such as circles, regular polygons, convex polygons, points, etc. Such shapes serve as vertices of the graph, on which the matching problem is defined. The adjacency can be defined by geometric considerations as well, such as two shapes intersecting each other, or shapes being reachable from one another via a curve in the plane that avoids all other shapes. The geometric setting motivates introducing new constraints to the problem such as non-crossing edges, which are important for the applications, such as circuit board design. The geometric settings are numerous and not as well understood as the general abstract setting (see the survey by Kano and Urrutia [41]), since introducing new constraints can change the complexity of the problem significantly resulting in some variants of the problem being $\mathcal{NP}$-hard [4]. As mentioned earlier, in this chapter we study the geometric version of matching in 2 dimensions, where geometric objects of interest are points and they can be matched with each other via non-crossing straight line segments. Observe that for our matching problems of interest we have

$\textsc{Opt}(I) = |I|$ as discussed earlier, so the competitive ratio coincides with the fraction of matched points that the algorithm can guarantee in the worst case. The bichromatic version of the problem was first defined by Atallah who gave the $O(n \log^2 n)$ time deterministic offline algorithm for it [6]. Dumitrescu and Steiger [25] generalized the problem and gave efficient approximation algorithms. The online versions of both monochromatic and bichromatic matching in the plane were introduced and studied by Bose et al.[15] and have been further examined by Kamali et al.[40] and in Sajadpour's master's thesis [57]. The main results from the work done on the advice complexity of online MNM and BNM prior to our work can be summarized as follows (recall that the input length is $2n$ rather than $n$ in BNM and MNM):

- an upper bound of $2 \log 3n \approx 3.17n$ and a lower bound of $\Omega(\log n)$ on the advice length to achieve optimality for MNM [15];
- tight bound of $\Theta(n \log n)$ on the advice length to achieve optimality for BNM [15] (note that the proof of the lower bound has a mistake);

Overall, the above set of results paints a picture that the bichromatic version is significantly more difficult than the monochromatic version of the problem, which is contrary to the offline setting, in which the bipartite setting is typically easier than the general setting. The above results indicate that the situation stays the same even in the presence of advice. A significant special case of the BNM and MNM problems is when input points are located at the same distance from the origin, i.e., they are all located on a common circle. First of all, the known lower bounds are based on such inputs, and second of all, designing algorithms for this special case may be easier and often serves as the first step towards the algorithm for the general case. We refer to this special case as "BNM/MNM on a circle", and we say "BNM/MNM on a plane" to emphasize the general case without the common circle restriction.

Our contributions are as follows. We observe that the lower bound of Bose et al. of $\Omega(n \log n)$ on the advice length to achieve optimality for BNM contains a bug. We give a new argument to establish the lower bound of $\log C_n \sim 2n - \frac{3}{2} \log n$, where $C_n$ is the $n^{\text{th}}$ Catalan number. The lower bound of Bose et al., as well as our new lower bound use input points that are located on a circle. We present an algorithm that uses $\log C_n$ bits of advice to solve BNM optimally on a circle. Thus,

we completely resolve the advice complexity of BNM on a circle. We also show a lower bound of $n/3 - 1$ and an upper bound of $\log C_n$ on the advice complexity for MNM on a plane. This gives an exponential improvement over the previously known lower bound and an improvement in the constant of the leading term in the upper bound. In addition, we establish $\frac{\alpha}{2} D(\frac{2(1-\alpha)}{\alpha} || 1/4)n$ lower bound on the advice complexity to achieve competitive ratio $\alpha \in (16/17, 1)$ for MNM on a circle where $D(p||q)$ is the relative entropy between two Bernoulli random variables with parameters $p$ and $q$. The results are summarized in Table 5.1.

| Problem Version | Previous LB | New LB (this work) | Previous UB | New UB (this work) |
|---|---|---|---|---|
| MNM | $\Omega(\log n)$ | $n/3 - 1$ | $2n \log 3 + o(n)$ | $\log C_n$ |
| MNM, $\rho = \alpha$ | - | $\frac{\alpha}{2} D(\frac{2(1-\alpha)}{\alpha} \|\| 1/4)n$ | - | - |
| BNM, Circle | $n \log n$ (Mistake) | $\log C_n$ | $n \log n$ | $\log C_n$ |
| BNM, Plane | $n \log n$ (Mistake) | $\log C_n$ | $n \log n$ | - |

Table 5.1: Summary of previously known results and our new results for the advice complexity of BNM and MNM.

In terms of conceptual contributions, our work shows that the previously held belief that BNM is more difficult than MNM even in the presence of advice is no longer justified. Indeed, it might still turn out that BNM requires asymptotically more bits of advice than MNM to achieve optimality, but this would require new lower bound constructions and arguments where input points are not located on the same circle. Whether the advice complexity of BNM in the plane is $\omega(n)$ or $O(n)$ is left as an important open problem. Our lower bound arguments also provide a conceptual contribution to the advice complexity area, since MNM and BNM have an interesting phenomenon of complicated evolution of constraints during the runtime of the algorithm (because of the non-crossing condition). This makes it difficult (or perhaps even impossible) to use the standard tools from advice complexity, such as a reduction from the string guessing problem [11] or partition trees [8]. Thus, our lower bound arguments are based on the first principles. In the BNM case, we establish connections between BNM on the circle and a particular structured subset of the permutation group that is in one-to-one correspondence with full binary trees. Our further investigations of the BNM problem hint at deep connections between the theory of permutations and the BNM problem, which will perhaps be encountered when one would try to get tight bounds for the plane version of the problem. Our

lower bound for the advice complexity of approximating MNM is based on probabilistic arguments and the nemesis input sequence is defined by a Markov chain that can successfully fool any online algorithm.

The rest of the chapter is organized as follows. Preliminaries are presented in Section 5.2. We discuss the bug in the paper of Bose et al. in Section 5.3. We present our new lower bound argument and the upper bound for BNM on a circle in that section as well. In Section 5.4 we present our results for the monochromatic version including the exponential improvement of the previously known lower bound, a slight improvement on the upper bound for achieving optimality, as well as a new lower bound to approximate MNM.

## 5.2 Preliminaries

### 5.2.1 Online Non-Crossing Matching

The input to MNM is a sequence of $2n$ points $p_1, p_2, \ldots, p_{2n} \in \mathbb{R}^2$ in general position arriving one-by-one. At step $i$, $p_i$ arrives and the algorithm can decide to match it with one of the unmatched points $p_j$ for some $j < i$ with a straight line segment or leave it unmatched. The goal of the algorithm is to match all input points without creating any crossings of line segments.

In BNM, the first $n$ points, denoted by $B = (b_1, \ldots, b_n)$, are blue and can be thought of as given in advance. The next $n$ points, denoted by $R = (r_1, \ldots, r_n)$, are red and arrive one-by-one. The algorithm has to match red points (online) upon their arrival to the blue points (offline) without creating any crossings. Note that any algorithm for BNM can also be used for MNM by treating the first $n$ points $p_1, \ldots, p_n$ as blue and the next $n$ points $p_{n+1}, \ldots, p_{2n}$ as red. Consequently, an upper bound (on competitive ratio and/or advice complexity) for BNM implies the same upper bound for MNM, and a lower bound for MNM implies a lower bound for BNM. Many of our algorithms that achieve good performance on a circle also work on slightly more general inputs, namely, when input points are vertices of their convex hull. When this happens we say that points are in a "convex position".

Suppose that the input to MNM $I = (p_1, \ldots, p_{2n})$ is in a convex position. Order the points in $I$ clockwise starting with $p_1$ in the convex hull of $I$. Let $j_i$ denote the index of point $p_i$ in this order

with $j_1 = 0$. The *parity* of $p_i$ is denoted by $\chi(p_i) := j_i \mod 2$. Therefore $\chi(p_1) = 0$, the parity of clockwise neighbor of $p_1$ in the convex hull is 1 and so forth. See Figure 5.1 for an illustration.

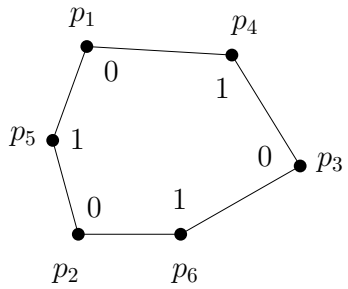



Figure 5.1: An instance of MNM in convex position with 6 points with their convex hull and parities.

### 5.2.2 Catalan Numbers and Related Topics

Catalan numbers [55], denoted by $C_n$, are defined recursively as

$$C_n = \begin{cases} 1 & n = 0 \\ \sum_{i=0}^{n-1} C_i C_{n-i-1} & n \geq 1 \end{cases}.$$

The closed-form expression in terms of binomial coefficients for Catalan numbers is $C_n = \frac{1}{n+1}\binom{2n}{n}$. By Stirling's approximation, we get $\log C_n \sim 2n - \frac{3}{2}\log n$. All logs are to the base 2 unless stated otherwise.

The following three combinatorial objects are used in this chapter extensively:

(1) A binary sequence $B = \{b_1, ..., b_{2n}\}$ is a *Dyck word* if it has the equal number of 0's and 1's and in every prefix of it there are no more 1's than 0's.

(2) A rooted ordered binary tree is a tree with a dedicated root vertex. Each vertex of such a tree has a left subtree and a right subtree (either of which can be empty).

(3) A permutation of integers $\{1, 2, \ldots, n\}$ is a reordering of these integers. We represent a permutation $\sigma$ as a sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ where $\sigma_i$ is the integer in the $i^{\text{th}}$ position according to $\sigma$. We say that $\sigma$ has a 213 pattern if there exist indices $i < j < k$ such that $\sigma_j < \sigma_i < \sigma_k$. If $\sigma$ does not have any 213 pattern, we say $\sigma$ is 213-*avoiding*. For instance, 52431 is 213-avoiding but 35124 is not since the subsequence 314 is a 213 pattern.

Catalan numbers have been discovered in a variety of different contexts, so they have many alternative definitions. In particular, the above three definitions of combinatorial objects can be used to define Catalan numbers. An interested reader is referred to [55] and references therein.

**Fact 35.** *The number of Dyck words of length $2n$, the number of ordered rooted binary trees with $n$ vertices, and the number of $213$-avoiding permutations of $[n]$ are all equal to $C_n$.*

## 5.3 Advice Complexity of Convex BNM

Bose et al. [15] gave an argument that at least $\lceil \log n! \rceil \sim n \log n$ bits of advice are needed to solve BNM with $2n$ points optimally. Their argument is based on a family of input sequences in a convex position. Unfortunately, the proof contains a mistake and it is not possible to fix that mistake to restore the original bound, since there is an algorithm achieving asymptotically much better advice complexity of $\lceil \log C_n \rceil \sim 2n - \frac{3}{2} \log n$ for inputs in convex position. In this section, we begin by presenting this algorithm. The pseudocode is shown in the Algorithm 5 and Algorithm 6 below. Then we show that $\lceil \log C_n \rceil$ bits of advice are necessary to achieve optimality for inputs on a circle, thus, establishing the tightness of $\lceil \log C_n \rceil$ bound. After presenting these arguments, we explain the mistake in the proof of Bose et al.

"Binary Tree Matching" (BTM) algorithm works on an input sequence of BNM as follows. Knowing the input sequence in which all blue points $B$ arrive before all red points $R$, the oracle finds a perfect non-crossing matching $M \subseteq R \times B$ between red and blue points. It is easy to see that such a matching always exists witnessed by a perfect matching of minimum total length, similar to Fact 1. The oracle then creates a tree representation $T$ of $M$ that takes into account the order in which red points arrive so that the online algorithm can later recover $M$ from $T$ on the fly.

The tree is constructed recursively by the convex partitioning of the plane based on pairs in $M$. Take the first red point $r_1$ and let $b_i$ denote the blue point to which $r_1$ is matched in $M$. The segment $\overline{r_1 b_i}$ splits the plane into two half-planes. Since the points are in a convex position and $M$ is a non-crossing matching it follows that each line segment corresponds to matched points in $M \setminus \{(r_1, b_i)\}$ lies entirely either in the left or the side of $\overline{r_1 b_i}$.

Thus, we partition $M = \{(r_1, b_i)\}$ into $M_L$ and $M_R$, where $M_L$ ($M_R$) consists of edges from $M$

that lie entirely in the left (respectively, right) side of $\overline{r_1 b_i}$. A tree $T_L$ ($T_R$) is constructed recursively for $M_L$ (respectively, $M_R$). Then $T$ is constructed by creating a root $t$ with $T_L$ as its left subtree and $T_R$ as its right subtree. Let $\langle T \rangle$ denote the encoding of a tree in binary. Then the oracle writes $\langle T \rangle$ on the tape.

When the online algorithm is executed, it starts by reading $T$ from the tape. When a red point arrives the algorithm uses $T$ to deduce which blue point it should be matched to according to $M$. Let's first observe how it works for the first red point. When $r_1$ arrives the algorithm knows that this point corresponds to the root of $T$ and therefore it knows that the matching $M$ is such that there are $|M_L| = |T_L|$ edges to the left of the line segment by which $r_1$ is matched to a blue point.

Thus, the algorithm can order blue points in clockwise order starting with $r_1$ and deduce that $b_i$ is the $|T_L| + 1$ blue point in this order and match $r_1$ with $b_i$. After these points are matched, the problem splits into two independent problems corresponding to $T_L$ and $T_R$, which are located inside the two regions induced by $\overline{r_1 b_i}$ segment, as described above. Thus, when the next red point $r_2$ arrives, the algorithm can determine whether $r_2$ lies in the left or the right side of $\overline{r_1 b_i}$ and apply the same procedure as before with $T_L$ or $T_R$, respectively. And so on.

We use the following notation some of which depends on the current step in the execution of the algorithm:

- $size(t)$ denotes the number of vertices in the subtree rooted at node $t$; $root(T)$ denotes the root of the (sub)tree $T$; $label(t)$ denotes the label of a node $t$; $left(t)$ ($right(t)$) denotes the left (respectively, right) child of the node $t$;
- $B_r$ is the set of currently available blue points such that matching the red point $r$ does not create any crossings with previously matched points;
- $C_r$ be the convex hull of the points in $B_r \cup \{r\}$;
- $b_r^i$ be the $i^{\text{th}}$ blue point in clockwise order in $C_r$.

In the pseudocode, the procedure $BTMOracle$ describes how the oracle operates. It uses $MatchingToBT$ subroutine to convert $M$ into a binary tree $T$. The procedure $BTMAlgorithm$ describes how the online algorithm operates. As the online algorithm constructs the matching it labels the nodes by the edges of the matching constructed so far.

**Algorithm 5** $BTMOracle$

**procedure** $BTMOracle(B = (b_1, \ldots, b_n), R = (r_1, \ldots, r_n))$
  $M \leftarrow$ non-crossing perfect matching $\subseteq R \times B$
  $T \leftarrow MatchingToBT(B, R, M)$
  write $\langle T \rangle$ on the tape
**end procedure**
**procedure** $MatchingToBT(B = (b_1, \ldots, b_n), R = (r_1, \ldots, r_n), M = \{e_1, \ldots, e_n\})$
  **if** $n = 1$ **then**
    **return** $T$ consisting of a single node
  **end if**
  let $i$ be such that $(r_1, b_i) \in M$
  $B_L, R_L, M_L, B_R, R_R, M_R \leftarrow \emptyset$
  **for** $j = 1$ to $n$ **do**:
    **if** $j \neq i$ **then**
      **if** $b_j$ is in the left half-plane induced by $\overline{r_1 b_i}$ **then**
        $B_L.append(b_j)$
      **else**
        $B_R.append(b_j)$
      **end if**
    **end if**
    **if** $j \neq 1$ **then**
      **if** $r_j$ is in the left half-plane induced by $\overline{r_1 b_i}$ **then**
        $R_L.append(r_j)$
      **else**
        $R_R.append(r_j)$
      **end if**
    **end if**
    **if** $e_j \neq \overline{r_1 b_i}$ **then**
      **if** $e_j$ is in the left half-plane induced by $\overline{r_1 b_i}$ **then**
        $M_L \leftarrow M_L \cup \{e_j\}$
      **else**
        $M_R \leftarrow M_R \cup \{e_j\}$
      **end if**
    **end if**
  **end for**
  let $T_L = MatchingToBT(B_L, R_L, M_L)$
  let $T_R = MatchingToBT(B_R, R_R, M_R)$
  let $T$ be a tree with root $t$
  $left(t) \leftarrow root(T_L)$
  $right(t) \leftarrow root(T_R)$
  **return** $T$
**end procedure**

**Theorem 36.** BTM *(see Algorithms 5 and 6) solves* BNM *with $n$ red and $n$ blue points in convex position optimally with $\lceil \log C_n \rceil$ bits of advice.*

**Algorithm 6** $BTMAlgorithm$

```
procedure BTMAlgorithm
    receive all blue points B = (b_1, ..., b_n)
    read T from the tape
    for every node t ∈ T do
        label(t) ← ∅
    end for
    for i = 1 to n do
        receive r_i
        t ← root(T)
        while label(t) ≠ ∅ do
            if r_i is on the left side of label(t) then
                t ← left(t)
            else
                t ← right(t)
            end if
        end while
        k ← size(left(t)) + 1
        match r to b_r^k
        label(t) ← (r, b_r^k)
    end for
end procedure
```

*Proof.* First, we justify the advice complexity of our algorithm. Let $\mathcal{T}_n$ be the set of all ordered unlabeled rooted binary trees with $n$ nodes. By Fact 35 we know $|\mathcal{T}_n| = C_n$. The oracle and the algorithm agree on an ordering of $\mathcal{T}_n$ and the bits $\langle T \rangle$ written by the oracle on the tape encode the index of $T$ according to this pre-agreed ordering. Thus the oracle can specify $T \in \mathcal{T}_n$ with $\lceil \log C_n \rceil$ bits of advice. Observe that the oracle does not need to use prefix-free code or specify $n$ separately, since the algorithm knows $n$ after receiving the blue points. Thus, the algorithm can deduce $C_n$ and read the first $\lceil \log C_n \rceil$ bits of advice from the tape after receiving the blue points.

From the description of the algorithm preceding the theorem, the correctness of the algorithm should be clear. We provide a brief argument by induction for completeness. Let $M'$ be the matching that is created by BTM in the online phase. By induction on $n$, we prove that $M'$ is the same as the offline matching $M$. The base case $n = 1$ is trivial. Consider next some $n \geq 2$. Let $t$ be the root of $T$. Suppose $r_1$ is matched with $b_i$ in $M$. Let $B_L$ and $B_R$ be the blue points that are one left and right side of $\overline{r_1 b_i}$ respectively and similarly define $R_L$ and $R_R$ for red points.

Let $M_L$ be the matching between $B_L$ and $R_L$ and define $M_R$ accordingly. Since all the points

are in convex position and edges of $M$ are non-crossing straight lines, there is no edge between the left and the right side of $\overline{r_1 b_i}$, thus $M_L = B_L$ and the size of the left subtree of $t$ is $B_L$. In the online matching, $r_1$ will be matched to $b_{B_L+1}$ which is the same blue point as in $M$ since there are $B_L$ blue points on the left side of $\overline{r_1 b_{B_L+1}}$.

Let $T_L$ and $T_R$ be left and right subtrees of $t$ respectively. Subtrees $T_L$ and $T_R$ are created by edges in $M_L$ and $M_R$ respectively. Let $M'_L$ be the result of the algorithm with $B_L, R_L$ and $T_L$ as input. By induction, $M'_L = M_L$ and if we define $M'_R$ similarly, with the same argument $M'_R = M_R$. Note that $M'_L$ and $M'_R$ are obtained by splitting the input sequences into two and running the algorithm twice. In addition, whenever the online algorithm receives a red point on the left (right) side of $\overline{r_1 b_i}$ it looks at $T_L$ ($T_R$). Thus red points on the left (right) side of $\overline{r_1 b_i}$ will be matched with the same blue points as in $M'_L$ ($M'_R$) and we can conclude $M' = M$ which means the matching by the online algorithm is perfect and non-crossing. □

Next, we present a lower bound on the advice complexity of BNM. Our lower bound uses input points located on a common circle, so it implies the same lower bound for BNM on inputs in a convex position and in general position.

**Theorem 37.** *Any online algorithm that solves* BNM *on a circle with $n$ red and $n$ blue points optimally uses at least $\lceil \log C_n \rceil$ bits of advice.*

*Proof.* We prove this by creating a family of input sequences $\mathcal{I}_n$ with $n$ red and $n$ blue points on a circle. In all input sequences in $\mathcal{I}_n$, blue points $B = (b_1, \ldots, b_n)$ have the exact same positions: they are located on the upper half of the circle and for each $1 \leq i \leq n$, $b_i$ is the $i^{\text{th}}$ blue point from the left. Formally, $b_i$ has coordinates $(\cos \alpha_i, \sin \alpha_i)$ (Figure 5.2 left), where $\alpha_i = \pi(1 - i/(n+1))$.

For each permutation $\sigma = (\sigma_1, \ldots, \sigma_n)$ on $[n]$, we generate the online input sequence $R(\sigma) = (r_1, \ldots, r_n)$ of red points in the lower half of the circle such that $r_i$ is the $\sigma_i^{\text{th}}$ red point from the left. Formally, let $\sigma_{-1} = 0$ and $\sigma_0 = n+1$ for the ease of the notation and auxiliary points $r_{-1}$ and $r_0$ on $(0, -1)$ and $(0, 1)$ respectively. These two points are not in the input sequence and we define them for the ease of describing the input sequence. Next, we generate $(r_1, r_2, \ldots, r_n)$ in order. After generating $k$ points we have $k+1$ arcs between $r_{-1}$ and $r_0$ in counterclockwise order with points $r_{-1}, r_0, \ldots, r_k$ as boundary points between arcs. When we refer to $j^{\text{th}}$ arc, we mean $j^{\text{th}}$ arc in this

order. To generate $r_i$ let $j = \min(i, \sigma_i)$ and place $r_i$ in the middle of $j^{\text{th}}$ arc. An example of such a construction for $R(2143)$ is illustrated in Figure 5.2: prior to $r_1$ being generated there is just one arc between $r_{-1}$ and $r_0$, so $j = \min(i, \sigma_i) = \min(1, 2) = 1$ and $r_1$ is placed in the middle of this one arc. This results in two arcs: one between $r_{-1}$ and $r_1$ and another between $r_1$ and $r_0$. For $i = 2$ we have $j = \min(2, \sigma_2) = (2, 1) = 1$ so we place $r_2$ in the middle of the first arc. For $i = 3$ we have $j = \min(3, \sigma_3) = \min(3, 4) = 3$, so we place $r_3$ in the middle of the arc between $r_1$ and $r_0$. For $i = 4$ we have $\min(4, \sigma_4) = \min(4, 3) = 3$, so we place $r_4$ in the middle of the third arc, that is in the middle of the arc between $r_1$ and $r_3$. Since input sequences in $\mathcal{I}_n$ only differ in red points we refer to an input sequence by its set of red points.

Let $\sigma$ and $\sigma'$ be two 213-avoiding permutations over $[n]$ and $R(\sigma) = (r_1, \ldots, r_n)$ and $R(\sigma') = (r'_1, \ldots, r'_n)$ be their corresponding input sequences. Let $i$ be the first index that $\sigma$ and $\sigma'$ are different, i.e. $i = \arg\min\{j \in \{1, \ldots, n\} \mid \sigma_j \neq \sigma'_j\}$. It is clear from the above construction that the first $i - 1$ points of $R(\sigma)$ and $R(\sigma')$ will be identical. We now claim that the 213-avoiding property implies that $r_i$ and $r'_i$ will be placed in the same locations as well. For $i = 1$, the claim is trivial because the first red point in all $I \in \mathcal{I}_n$ is placed at $(0, -1)$. For $i > 1$, we establish the claim by contradiction.

Suppose $r_i$ and $r'_i$ are placed in different locations. Prior to the placement of the $i^{\text{th}}$ points, the two inputs partition the lower half of the circle into identical sequences of arcs. Since points are always placed in the middle of one of the arcs, $r_i$ and $r'_i$ must have been placed in different arcs. Without loss of generality, suppose $r_i$ is placed in the middle of an arc to the left of $r'_i$, and there exists some $1 \leq j < i$ such that $r_j$ lies between $r'_i$ and $r_i$.

Since $\sigma_j = \sigma'_j$, the number of red points in $R(\sigma)$ to the right of $r_j$ is equal to the number of red points in $R(\sigma')$ to the left of $r_j$. However, since $r'_i$ is on the right side of $r_j$ in $R(\sigma')$ and $r_i$ is on the left side of $r_j$ in $R(\sigma)$, there must exist some $i < k \leq n$ such that $r_k$ is placed to the right of $r_j$. This implies that $\sigma_i \leq \sigma_j \leq \sigma_k$. Since $i < j < k$, the subsequence $\sigma_j, \sigma_i, \sigma_k$ forms a 213 pattern, which contradicts the assumption that $\sigma$ is 213-avoiding. Thus, $r_i$ and $r'_i$ must be placed in the same locations, as claimed.

Note that in an input sequence $I \in \mathcal{I}_n$, for $1 \leq i \leq n$, if $r_i$ is the $j^{\text{th}}$ red point from left, it should be connected to $b_j$ in the unique perfect matching corresponding to $I$. The decision of

a deterministic online algorithm is based on the prefix of the input sequence that it has received so far. With the same prefix of input sequences, $r_i$ and $r'_i$ should be matched to $b_{\sigma_i}$ and $b_{\sigma'_i} \neq b_{\sigma_i}$ respectively, therefore one deterministic algorithm cannot solve both $R(\sigma)$ and $R(\sigma')$ optimally and by pigeonhole principle together with Fact 35 we need at least $C_n$ deterministic algorithms and at least $\lceil \log C_n \rceil$ bits of advice for solving this problem optimally. □

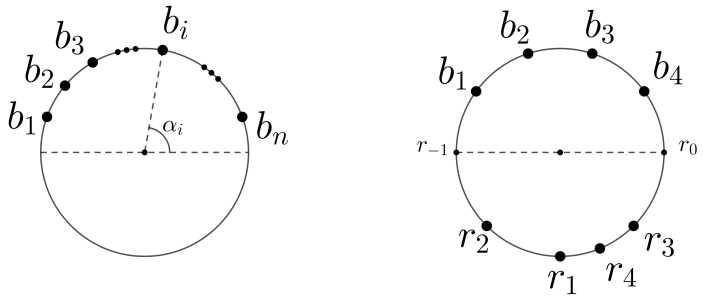



Figure 5.2: Left: positions of blue points. Right: the input sequence for $n = 4$, corresponding the permutation $\sigma = (2, 1, 4, 3)$.

Bose et al. [15] had almost the same procedure of creating instances from permutations, but mistakenly, claimed no deterministic algorithm can solve two different permutations rather than two 213-avoiding permutations. They did not consider that a single deterministic algorithm can solve inputs $R(\sigma)$ for multiple different $\sigma$ because of non-crossing constraints. For example, input sequences associated with permutations $\sigma = (2, 1, 3)$ and $\sigma' = (3, 1, 2)$ can be solved with one deterministic algorithm: the algorithm matches $r_1 = r'_1$ with $b_2$, then if $r_2$ arrives it will be in the left half-plane associated with $(r_1, b_2)$ edge, so it can be matched only with $b_1$ given the non-crossing constraint. If $r'_2$ arrives then it will be in the right half-plane associated with $(r'_1, b_2)$ edge, so it can be matched only with $b_3$ given the non-crossing constraint. Thus, once the algorithm determines to match $r_1$ with $b_2$ it can complete it to a perfect matching in both $R(\sigma)$ and $R(\sigma')$. It implies that not every permutation out of $n!$ possibilities requires a different deterministic algorithm. Another way of looking at it is that $r_i$ might be different from $r'_i$ where $i$ is the first coordinate at which $\sigma$ and $\sigma'$ differ unless $\sigma$ and $\sigma'$ are 213-avoiding. If $r_i \neq r'_i$ then a single deterministic algorithm can use this information to match $r_i$ and $r'_i$ differently.

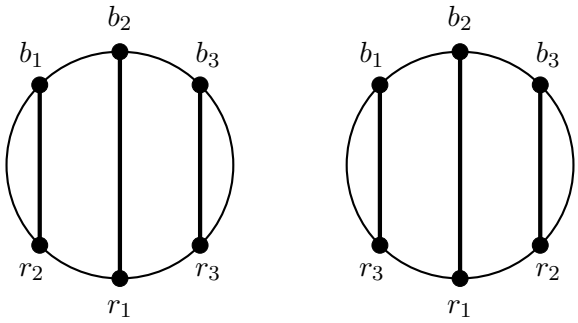



Figure 5.3: Two input sequences, associated with $\sigma = (2, 1, 3)$ on the left and $\sigma = (3, 1, 2)$ on the right, can be solved optimally using a single deterministic algorithm. This serves as a counterexample to the proof presented by Bose et al. [15].

## 5.4 Advice Complexity of MNM

In this section, we present our results for the advice complexity of MNM. Before presenting the best-known result published in [17] for points in general position, We begin by presenting a bound of $3n$ on the advice complexity of solving MNM in Subsection 5.4.1 which solves MNM even when the points are not in general position.

The algorithm using $\lceil \log C_n \rceil$ bits of advice for inputs in convex position for BNM from Section 5.3 can be viewed as an algorithm for MNM that postpones matching points as long as possible leaving the first $n$ points unmatched (treating them as "blue" points). In Subsection 5.4.2, we present a family of algorithms that have an opposite behavior – each algorithm in the family tries to match points as soon as possible, so we call it ASAP algorithm. We show that just like BTM algorithm, ASAP algorithm achieves optimality with $\lceil \log C_n \rceil$ bits of advice for MNM on inputs in convex position.

Then we present an algorithm in Subsection 5.4.3 that solves MNM in general position with $\lceil \log C_n \rceil$ bits of advice. In Subsection 5.4.4 we present an $\lfloor n/3 \rfloor$ lower bound for optimally solving MNM on a circle, and in Subsection 5.4.5 we show an $\Omega_\alpha(n)$ lower bound for achieving competitive ratio $\alpha \in (16/17, 1)$.

### 5.4.1 $3n$ Upper Bound

Bose et. al. [15] showed how to solve MNM with $(2 \log 3)n \approx 3.17n$ bits of advice. In this section, we show that their result can be improved to $3n$ bits of advice by just using a slightly more efficient way of generating advice bits.

On input $P = (p_1, p_2, \ldots, p_{2n})$ the oracle sorts the points by their $x$-coordinates and matches consecutive pairs of points. Let $M$ be the resulting matching. Suppose $p_i$ is matched with $p_j$ in $M$, thus $p_j$ has the closest $x$-coordinate to $p_i$ either on its right or left. The advice string $A$ consists of $n$ parts, i.e. $A = a_1 a_2 \cdots a_{2n}$ such that for each $1 \leq i \leq 2n$, $a_i$ is defined as follows:

- $a_i = 0$ if $j > i$, i.e. $p_j$ comes after $p_i$;
- $a_i = 10$ if $i < j$ and $p_j$ is on the left side of $p_i$;
- $a_i = 11$ if $i < j$ and $p_j$ is on the right side of $p_i$.

The online algorithm reads $A$ and reveals the sequence $a_1, \ldots, a_{2n}$ and for $1 \leq i \leq 2n$ decides as follows:

- if $a_i = 0$, leaves $p_i$ unmatched;
- if $a_i = 10$, matches $p_i$ with its closest $x$-coordinate on the its left;
- if $a_i = 11$, matches $p_i$ with its closest $x$-coordinate on the its right.

Observe that this is just a prefix-free encoding/decoding of the three choices of an algorithm described above. The pseudocode is given in Algorithm 7.

**Theorem 38.** *$SortedMatching$ (Algorithm 7) solves* MNM *with $2n$ points in general position optimally with $3n$ bits of advice.*

*Proof.* Let $(p_i, p_j)$ be an edge in the offline matching $M$ such that $i < j$. Since $p_i$ comes before $p_j$, $a_i$ is 0 and $a_j$ is either 10 or 11 therefore for each edge in $M$ there are 3 bits of advice and the size of the advice string is $3n$. Since the points are in general position, no three points may lie on the same vertical line. It follows that the matching $M$ produced by $SortedMatchingOracle$ is non-crossing and from the description of the algorithm it is easy to see that the online matching produced by $SortedMatchingAlgorithm$ is the same as $M$. □

**Algorithm 7** $SortedMatching$ algorithm.

```
procedure SortedMatchingOracle(P = (p_1, ..., p_2n))
    let L be the sorted list of P by their x-coordinates
    M ← ∅
    for i = 1 to n do
        let p = L[2i + 1] and q = L[2i + 2]
        M.append((p, q))
    end for
    A ← [ ]
    for i = 1 to 2n do
        find j such that (p_i, p_j) ∈ M
        if j > i then
            A.append(0)
        else if p_j is on the left of p_i then
            A.append(10)
        else
            A.append(11)
        end if
    end for
    Write A on the tape
end procedure

procedure SortedMatchingAlgorithm
    while receive a new point p_i do
        read a bit b_1 from the tape
        if b_1 = 0 then
            leave p_i unmatched
        else
            read another bit b_2 from the tape
            if b_2 = 0 then
                match p_i with its closest x-coordinate point on its left
            else
                match p_i with its closest x-coordinate point on its right
            end if
        end if
    end while
end procedure
```

### 5.4.2 $\lceil \log C_n \rceil$ Upper Bound for Convex Position

In this section, the set of *available* points plays a crucial role. It is defined for an online algorithm as follows: when $p_i$ arrives, a point $p_j$ for $1 \leq j < i$ is called *available* if matching $p_i$ with $p_j$ does not create any crossing with the existing edges in the matching constructed by the algorithm by time

$i$. Let $A_i$ denote the set of all available points for $p_i$. Observe that since the decisions of the online algorithm are deterministic, the oracle knows existing matching edges at time $i$, and hence it knows $A_i$.

Recall that $\chi(p)$ denotes the parity of $p$ as described in Subsection 5.2.1. First, observe that in a perfect non-crossing matching $M$ if $p_i$ is matched with $p_j$ then the two points have opposite parities, for otherwise there would be an odd number of points in half-planes associated with the edge $(p_i, p_j)$, guaranteeing that at least one point on each side must remain unmatched. Our advice algorithm is based on the observation that this claim can be "reversed": as long as an online algorithm matches points of opposite parities without creating any crossings, this partial non-crossing matching remains valid, that is it can be extended to a perfect non-crossing matching of the whole instance. Of course, the parities are not known to the online algorithm and they cannot be inferred from the input seen so far, since they are based on a complete instance. One possibility is for an oracle to specify all parities with $2n$ bits of advice, but we can do it slightly more efficiently with $\lceil \log C_n \rceil$ bits of advice if $n$ is known to the algorithm in advance. If $n$ is not known then the savings from our more efficient encoding are diminished because of the need to encode numbers up to $C_n$ with a prefix-free code.

Observe that we can insist that as soon as $p_i$ arrives such that $A_i$ contains a point of opposite parity to $p_i$ then $p_i$ is matched. We refer to this property as "as soon as possible" or ASAP, for short. In an ASAP algorithm, if one point in $A_i$ has parity opposite to $p_i$ then all points in $A_i$ have parity opposite to $p_i$. This means that as long as ASAP algorithm can infer that it is possible to match $p_i$ it can choose any point in $A_i$ for such matching. Therefore ASAP algorithm forms a family of algorithms, where the specific algorithm is determined by how ties are broken. The tie-breaking rule is not important for our analysis, so we do not specify it explicitly.

Next, we describe the details of how our algorithm (see Algorithm 8) works. The oracle creates a binary string $D = (a_1, \ldots, a_{2n})$. For $i \in \{1, \ldots, 2n\}$, if there exists an available point $p_j \in A_i$ such that $\chi(p_j) \neq \chi(p_i)$ the oracle sets $a_i$ to 1, otherwise it sets it to 0. Thus, $a_i$ indicates whether $p_i$ could be matched at step $i$. The number of edges in a perfect matching is $n$ and for every $1 \leq i \leq 2n$, at step $i$, the number of points matched at the time of their arrival is not more than the number of points that were left unmatched at the time of their arrival hence $D$ is a Dyck word. Let $\langle D \rangle$ denote

the encoding of a Dyck word in binary. If $n$ is known to the algorithm then $\langle D \rangle$ can consist of just $\lceil \log C_n \rceil$ bits. If $n$ is not known to the algorithm the oracle can encode $n$ using Elias delta coding followed by the encoding of $D$ as before, resulting in $\lceil \log C_n \rceil + \log n + O(\log \log n) \sim 2n - \frac{1}{2} \log n + O(\log \log n)$ bits. When the online algorithm is executed, it starts by reading $D$ from the tape. When an online point $p_i$ arrives, if $a_i = 1$ the algorithm matches $p_i$ with one arbitrary point of $A_i$ and if $a_i = 0$ it leaves $p_i$ unmatched.

**Algorithm 8** ASAP algorithm.

```
procedure ASAPMatchingOracle
    D ← [0]
    for i = 2 to 2n do
        if there exist an available point p_j for p_i and χ(p_i) ≠ χ(p_j) then
            D.append(1)
        else
            D.append(0)
        end if
    end for
    write ⟨D⟩ on the tape
end procedure

procedure ASAPAlgorithm
    read D from the tape
    while receive a new point p_i do
        if d_i = 1 then
            connect p_i to one of the available points
        else
            leave p_i unmatched
        end if
    end while
end procedure
```

**Theorem 39.** ASAP *(see Algorithm 8) solves* MNM *with* $2n$ *points in convex position optimally with* $\lceil \log C_n \rceil + \log n + O(\log \log n)$ *bits of advice. Moreover, if* $n$ *is known to the algorithm only* $\lceil \log C_n \rceil$ *bits are required.*

*Proof.* By Fact, 35we know that the number of Dyck words with length $2n$ is $C_n$. Hence, the encoding described prior to this theorem achieves the desired advice complexity.

ASAP matches $p_i$ with its available vertices thus it does not create crossing edges. It is left to see that the constructed matching is perfect. We can demonstrate it by strong induction on $n$. For

$n = 1$ we have $\chi(p_1) \neq \chi(p_2)$ and we are done. For $n \geq 2$ define $i$ to be the smallest index such that $\chi(p_i) \neq \chi(p_1)$. ASAP then matches $p_i$ with $p_j$ for some $j \in \{1, \ldots, i-1\}$. The line passing through $p_i$ and $p_j$ splits the plane into two half-planes. Let $P_1$ consist of the points from the input that lie in one half-plane. Note that $P_1$ is a sequence and the order of points is the same as their order in $P$. We define $P_2$ similarly but for the other half-plane. Thus $P_1 = (p_{i_1}, p_{i_2}, \ldots, p_{i_k})$ and $P_2 = (p_{j_1}, p_{j_2}, \ldots, p_{j_m})$. Since every point other than $p_i$ and $p_j$ appears either in $P_1$ or $P_2$ we have $k + m + 2 = 2n$. Also, since $\chi(p_i) \neq \chi(p_j)$ we have that $P_1$ and $P_2$ contain even number of points each. Define $D_1 = (a_{i_1}, a_{i_2}, \ldots, a_{i_k})$ and $D_2 = (a_{j_1}, a_{j_2}, \ldots, a_{j_m})$ be the portions of the advice $D$ corresponding to items in $P_1$ and $P_2$, respectively. It is easy to see that $P_1$ forms an input to an MNM problem of size $k/2 < n$ and $D_1$ is a correct advice string corresponding to this input. Similarly for $P_2$. Thus, by induction assumption ASAP creates a perfect matching $M_1$ when it runs on $P_1$ with advice $D_1$, and it also creates a perfect matching $M_2$ when it runs on $P_2$ with $D_2$. Lastly, we observe that the decision of the algorithm for an input point $p$ depends on the advice bit and the set of available points. Since the advice bit and the set of available points for each $p \in P_1$ coincides with the advice bit and the set of available points for $p \in P$ after $p_i$ is matched with $p_j$, the algorithm will construct matching $M_1$ in $P$. Similarly, it will construct matching $M_2$ in $P$, and together with $p_i$ being matched with $p_j$ it gives a perfect matching for the entire instance $P$. □

### 5.4.3 $\lceil \log C_n \rceil$ Upper Bound for General Position

Now we propose an online algorithm with advice, which we call "Split-And-Match" (SAM), and show that it achieves optimality. For input sequences of size $2n$, SAM uses a family of $C_n$ advice strings. SAM oracle and algorithm jointly maintain a partitioning of the plane into convex regions, and a responsibility relation, where a point can be assigned to be responsible for at most one region, and each region can have at most one point responsible for it. When a new point $p$ arrives in a region $R$, if $R$ does not have a responsible point, then $p$ is assigned to be the responsible point for $R$, and $p$ is left unmatched at this time. Otherwise, suppose that $q$ is the responsible point for $R$ at the time $p$ arrived. In this case, the responsibility of $q$ is removed, and the plane partition is refined by subdividing $R$ into $R_1$ and $R_2$ – the sub-regions of $R$ formed by $\overline{pq}$. If the total number of points (including future points, but excluding $p$ and $q$) in $R_i$ is even for each $i \in \{1, 2\}$, then

$p$ and $q$ are matched (we refer to this event as a "safe match"), and $R_1$ and $R_2$ do not have any responsible points assigned to them. Otherwise, $p$ and $q$ are not matched, and $q$ is made responsible for $R_1$, and $p$ is made responsible for $R_2$. Note that when a region has a responsible point, that point is assumed to lie in the region by convention (if it lies on the boundary, this condition implies how the algorithm breaks ties).

To implement the above procedure in the advice model, the SAM oracle (see Algorithm 9 ) creates a binary string $D$ of length $2n$, where the $i^{\text{th}}$ bit indicates whether $p_i$ arrives in a region which has some responsible point $p_j$ assigned to it, and $p_i$ and $p_j$ form a safe match. The string $D$ is encoded succinctly on the tape and is passed to SAM. The SAM algorithm (see Algorithm 10) begins by reading the encoding of $D$ from the tape (prior to the arrival of any online points), recovers $D$ from the encoding, and then uses the information in $D$ to run the above procedure creating safe matches.

Observe that we aim to show the bound $\log C_n + \log n \sim 2n - \frac{1}{2}\log n$ on the advice complexity. The reason for the additive savings of $\frac{3}{2}\log n$ in the $\log C_n$, as compared to $2n$, is that not all binary strings of length $2n$ can be generated as $D$. Thus, the oracle and the algorithm can agree beforehand on the ordering of the universe of possible strings $D$, which we call the advice family. Then the oracle writes on the tape the index of a string in this ordering that corresponds to $D$ for the given input. The following theorem establishes the correctness of this algorithm, as well as the claimed bound on the advice complexity.

**Theorem 40.** SAM *achieves a perfect matching with the advice family of the size of* $C_n$.

*Proof.* It is easy to see that there are only two kinds of regions that can be encountered during the execution of SAM:

- type I: this region does not have a responsible point, it is empty at the time of creation, and there is an even number of points arriving in this region in the future, and
- type II: this region has a responsible point, which is the only point in the region at the time of its creation, and there is an odd number of points arriving in this region in the future.

We argue inductively (on the number of future points arriving in a region) that the algorithm ends up matching all points inside a region, regardless of their type.

**Algorithm 9** $Split\text{-}And\text{-}Match$ Oracle.

**procedure** $Split\text{-}And\text{-}Match\text{-}Oracle$
  $D \leftarrow [0]$
  make $p_1$ responsible for the plane
  **for** $i = 2$ to $2n$ **do**
    let $R$ be the region that $p_i$ arrives in
    **if** $R$ has a responsible point $p_j$ **then**
      revoke the responsibility of $p_j$
      divide $R$ into $R_1$ and $R_2$ by $\overline{p_i p_j}$
      **if** $R_L$ (and $R_R$) is going to contain an even number of points in total **then**
        $D.append(1)$
      **else**
        make $p_j$ and $p_i$ responsible for $R_1$ and $R_2$ respectively
        $D.append(0)$
      **end if**
    **else**
      make $p_i$ responsible for $R$
      $D.append(0)$
    **end if**
  **end for**
  pass $D$ to the algorithm
**end procedure**

**Algorithm 10** $Split\text{-}And\text{-}Match$ Algorithm.

**procedure** $Split\text{-}And\text{-}Match(D)$
  **while** receive a new point $p_i$ **do**
    let $R$ be the region that $p_i$ arrives in
    **if** $R$ has a responsible point $p_j$ **then**
      revoke the responsibility of $p_j$
      divide $R$ into $R_1$ and $R_2$ by $\overline{p_i p_j}$
      **if** $D[i] == 1$ **then**
        match $p_i$ with $p_j$
      **else**
        make $p_j$ and $p_i$ responsible for $R_1$ and $R_2$ respectively
        leave $p_i$ unmatched
      **end if**
    **else**
      make $p_i$ responsible for $R$
      leave $p_i$ unmatched
    **end if**
  **end while**
**end procedure**

The base case for type I region is trivial: the number of future points is 0, and there is nothing to prove. The base case for type II region is easy: one point arrives in the region, then according to the algorithm it will be matched to the responsible point (since $R_1$ and $R_2$ are empty).

For the inductive step, consider type I region $R$, and suppose that $2k$ points arrive inside the region. The first point that arrives in the region, becomes responsible for this region, changing its type to II. There are $2k-1$ future points arriving in this region, and the claim follows by inductive assumption applied to the type II region.

Now, consider type II region $R$, and suppose that $2k-1$ points arrive inside the region. Let $q$ be the responsible point for $R$, and let $p$ be the first point arriving inside $R$. Note that $\overline{pq}$ partitions $R$ into $R_1$ and $R_2$. There are two possible cases. Case 1: $R_1$ and $R_2$ are both of type I, then $p$ is matched with $q$ and the inductive step is established for $R$ by invoking induction on $R_1$ and $R_2$. Case 2: $R_1$ and $R_2$ are both of type II, then inductive step is established for $R$ by invoking induction on $R_1$ and $R_2$.

Observe that the entire plane is a region of type I at the beginning of the execution of the algorithm (prior to arrival of any points). Thus, correctness of the algorithm follows by applying the above claim to this region.

To establish the bound on advice complexity, observe that by the definition of the algorithm, SAM matches the most recent point whenever $D[i]$ is 1 and does not match otherwise. Thus, $D$ has an equal number of zeros and ones and no prefix of $D$ has more ones than zeros. This makes $D$ a Dyck word and it is known that there are $C_n$ Dyck words of size $2n$. □

### 5.4.4 $\lfloor n/3 \rfloor - 1$ Lower Bound

**Theorem 41.** *Any online algorithm that solves* MNM *on a circle with $2n$ points optimally uses at least $\lfloor n/3 \rfloor - 1$ bits of advice.*

*Proof.* Let $n = 3k$. We create a family of adversarial input instances $\mathcal{I}_n$ such that each instance $I \in \mathcal{I}_n$ contains $6k = 2n$ points on the perimeter of a circle. The first $4k$ points $p_1, \ldots, p_{4k}$ are exactly the same in all instances $I \in \mathcal{I}_n$ and are positioned on the perimeter of a circle at regular angular intervals. These points arrive clockwise with $p_1$ located at the North pole. The interval between $p_i$ and $p_{i+1}$ is called the $i^{\text{th}}$ interval (the $4k^{\text{th}}$ interval is between $p_{4k}$ and $p_1$).

Choose $j \in \{0, 1, 2, \ldots, 2k\}$ and choose a subset $S$ of $j$ intervals from the first $4k-1$ intervals. The next $j$ points are placed in the middle of each interval in $S$ and arrive in clockwise order. The remaining $2k-j$ points are placed in the $4k^{\text{th}}$ interval in clockwise order at regular angular intervals within the $4k^{\text{th}}$ interval. The family $\mathcal{I}_n$ arises out of all possible choices of $j$ and $S$.

$$|\mathcal{I}_n| = \sum_{j=0}^{2k} \binom{4k-1}{j} \geq 2^{4k-2} + 1. \tag{2}$$

For each $I = (p_1, \ldots p_{6k}) \in \mathcal{I}_n$, let $X(I)$ be the binary sequence $(\chi(p_1), \ldots, \chi(p_{4k}))$ of parities of the first $4k$ points of $I$ (recall that $\chi(p)$ is the parity of $p$ as described in Subsection 5.2.1). Consider $I \neq I' \in \mathcal{I}_n$ and take the smallest index $j > 4k$ such that $p_j$ belongs to different intervals in $I$ and $I'$. Without loss of generality suppose that the location of $p_j$ in $I$ is before the location of $p_j$ in $I'$ in clockwise order from the North pole. Let $p_\ell$ be the clockwise neighbor of $p_j$ in $I$. Then the parity of $p_\ell$ in $I$ is different from the parity of $p_\ell$ in $I'$. This implies that $X(I) \neq X(I')$ demonstrating that $X : \mathcal{I}_n \to \{0,1\}^{4k}$ is one-to-one.

If $M$ is a non-crossing matching on $p_1, \ldots, p_{4k}$ we call it a *prior* matching. We say a prior matching $M$ is *consistent* with an input sequence $I \in \mathcal{I}_n$ if it can be completed to a perfect non-crossing matching on points in $I$. The size of a consistent $M$ should be at least $k$ otherwise with more than $2k$ unmatched points and $2k$ arriving points it can not become a perfect matching. Moreover, for every $(p_i, p_j) \in M$, parities of $p_i$ and $p_j$ should be different, i.e. $\chi(p_i) \neq \chi(p_j)$. Otherwise $(p_i, p_j)$ splits the points of $I$ into two odd sets and it cannot become a perfect non-crossing matching.

Since $X$ is one-to-one, a single prior matching $M$ can be consistent with at most $2^{3k}$ input sequences: $2^k$ different parities for points with opposite parities in $k$ matched edges and at most $2^{2k}$ different parities for other points.

The first $4k$ point of every in input sequence in $\mathcal{I}_n$ is the same, therefore for a deterministic algorithm, there is a one-to-one relation between the set of the prior matchings that it makes and the set of advice strings from the oracle for solving $\mathcal{I}_n$. Thus we say the advice string $A$ is consistent with input sequence $I \in \mathcal{I}_n$ if the prior matching that the algorithm makes with advice $A$ is consistent with $I$.

The size of the input family is greater than $2^{4k-2}$ and each advice string can be consistent with

at most $2^{3k}$ input sequences therefore there should be more than $2^{k-2}$ different advice strings to cover all input sequences in $\mathcal{I}_n$ hence the algorithm needs at least $k-1$ bits of advice.

For $n = 3k+1$ and $n = 3k+2$, we create a family inputs the same way as $n = 3k$ case but with $4k+1$ and $4k+2$ points in the fixed prefix respectively. With the same argument, we also need at least $k-1$ bits of advice in both cases. □

### 5.4.5 $\Omega_\alpha(n)$ Lower Bound for $\alpha$-Approximation

In this subsection, we prove a lower bound on the amount of advice needed for an online algorithm to match at least $2\alpha n$ points for $\alpha \in (16/17, 1)$, without creating any crossing line segments. That is we study the advice complexity of achieving a strict competitive ratio $\alpha$. It is not clear whether it is possible to reduce from the binary string guessing problem [11] to our problem of interest. As such, instead of using the binary string guessing problem as a black box, we argue from first principles similar to the proof of the lower bound on string guessing. The argument is probabilistic: an algorithm that uses $k$ bits of advice and guarantees that at least $2\alpha n$ points are matched gives rise to a randomized algorithm that matches at least $2\alpha n$ points with a probability of at least $2^{-k}$ on every input (and consequently with respect to random inputs). This can be achieved by setting $k$ advice bits uniformly at random as opposed to having an oracle generate them.

We present a distribution on inputs of length $2n$, similar to the one used in Chapter 4 for proving negative results on randomized algorithms (Subsection 4.4.1) and demonstrate that a randomized algorithm cannot match many points with high probability. The input is generated by a Markov chain process and all the points are on a circle. We maintain an active arc such that all future input points will be generated within this arc. The next point $p_i$ to arrive is placed in the middle of this arc. This splits the current arc into two and the process continues on one of these two arcs chosen at random.

However, it is not enough to simply continue the process on a randomly chosen arc, since an algorithm might decide to keep matching points as soon as they arrive and can be matched. Thus, we need to set up a probabilistic trap for this choice. We do this by deciding at random to insert a "fake" point in one arc and continue the process on another arc. We leave a possibility of not having a "fake" point and immediately continuing into one of the arcs chosen at random.

This Markov chain process is designed so that with a constant probability we can guarantee a future unmatched point no matter whether $p_i$ is matched at time $i$ or is left unmatched by the algorithm. There are several possibilities to consider and the fact that the above trap works in all cases are a bit subtle. Nonetheless, the analysis only requires elementary probability theory and the following well-known tail bound:

**Fact 42.** *If $X$ is a binomial random variable with variables $n$ and $p$, then by the Chernoff bound for $\alpha \in (0, p)$ we have:*

$$P\{X \leq \alpha n\} \leq 2^{-nD(\alpha||p)}$$

*Where $D(\alpha||p) = \alpha \log_2 \frac{\alpha}{p} + (1-\alpha)\log_2 \frac{1-\alpha}{1-p}$ is the relative entropy between an $\alpha$-coin and a $p$-coin.*

Now, we are ready to present the main result of this subsection.

**Theorem 43.** *Any online algorithm with advice for* MNM *that guarantees at most $2(1-\alpha)n$ unmatched points, where $\alpha \in (16/17, 1)$, reads at least $\frac{\alpha}{2} D(\frac{2(1-\alpha)}{\alpha}||1/4)n$ bits of advice.*

*Proof.* We begin by describing a distribution of inputs consisting of $2n$ points on a circle, as discussed at the beginning of this section. Each point is either "fake" or "parent" – terms that will become clear after we describe the distribution.

Create two sequences $F_1, \ldots, F_{2n}$ and $R_1, \ldots, R_{2n}$ of i.i.d. Bernoulli random variables with parameter $1/2$. Put points $p_1$ and $p_2$ on the North and the South poles of $\mathbb{S}^1$ ($(0,1)$ and $(0,-1)$) respectively and make $p_2$ a *parent*. Starting from $i = 2$, follow the process: if $p_i$ is a parent, let $s_i^0$ and $s_i^1$ be its left and right adjacent arcs (maximal portions of the perimeter of a circle starting with $p_i$ and continuing until a previously generated point is encountered in clockwise and counterclockwise directions, respectively). Place $p_{i+1}$ in the middle of $s_i^{R_i}$. If $F_i = 0$ make $p_{i+1}$ a parent and continue the process with $p_{i+1}$. If $F_i = 1$, make $p_{i+1}$ a *fake* point, place $p_{i+2}$ in the middle of $s_i^{1-R_i}$, make $p_{i+2}$ a parent and continue with $p_{i+2}$.

From the above process we see that $F_i$ controls whether a point $p_i$ generated after a parent is fake, and $R_i$ controls whether $p_i$ is placed in the right arc or the left arc. We introduce $P_i$ as the indicator that $p_i$ is a parent. We have $P_i = 1 - P_{i-1}F_i$. Next, we analyze the probability with

which a randomized algorithm can achieve a large matching. Let ALG be an arbitrary randomized algorithm. Note that at time $i$, $P_i, F_i$, and $R_i$ are not known to the algorithm. Let $M$ be the size of the matching (random variable) constructed by ALG and $T_1 < ... < T_M$ be the times at which the algorithm matched points. Let $A_i$ be the set of all available points to which $p_{T_i}$ can be connected. Suppose $p_{T_i}$ is parent and the algorithm matches it to $p_j \in A_i$, this matching splits $A_i \setminus \{p_j\}$ into $A_i^0$ and $A_i^1$, points on the left side and the right side of the matching respectively.

A point becomes isolated if it is unmatched and cannot be matched afterward. If $|A_i^x|$ is zero and $p_{T_i+1}$ is fake and goes to $s_{T_i}^x$ it will be isolated. If $|A_i^x|$ is greater than zero and $p_{T_i+1}$ is a parent and it goes in $s_{T_i}^{1-x}$, points in $A_i^x$ become isolated.

For $1 \leq i \leq M$ let $a_i \in \{00, 0+, +0, ++\}$ indicate the number of points in $A_i^0$ and $A_i^1$ at time $T_i$ (e.g. $a_i = +0$ indicates that $|A_i^0| > 0$ and $|A_i^1| = 0$). Let $X_i$ be the indicator that $p_{T_i}$ is a parent and the matching at time $T_i$ creates at least one isolated point. Then we have:

$$X_i = \begin{cases} (1 - P_{T_i-1}F_{T_i})F_{T_i+1} & \text{if } a_i = 00 \\ (1 - P_{T_i-1}F_{T_i})F_{T_i+1}(1 - R_{T_i}) + (1 - P_{T_i-1}F_{T_i})(1 - F_{T_i+1})R_{T_i} & \text{if } a_i = 0+ \\ (1 - P_{T_i-1}F_{T_i})(1 - F_{T_i+1})(1 - R_{T_i}) + (1 - P_{T_i-1}F_{T_i})F_{T_i+1}R_{T_i} & \text{if } a_i = 0+ \\ (1 - P_{T_i-1}F_{T_i})(1 - F_{T_i+1}) & \text{if } a_i = ++ \end{cases}$$

Values of $X_1$ and $X_M$ do not follow this equation if $T_1 = 2$ or $T_M = 2n$ but it does not affect the asymptotic result and we can ignore these scenarios. Now we define an auxiliary random sequence $Y_1, \ldots, Y_M$ as follows:

$$Y_i = \begin{cases} (1 - F_{T_i})F_{T_i+1} & \text{if } a_i = 00 \\ (1 - F_{T_i})F_{T_i+1}(1 - R_{T_i}) + (1 - F_{T_i})(1 - F_{T_i+1})R_{T_i} & \text{if } a_i = 0+ \\ (1 - F_{T_i})(1 - F_{T_i+1})(1 - R_{T_i}) + (1 - F_{T_i})F_{T_i+1}R_{T_i} & \text{if } a_i = 0+ \\ (1 - F_{T_i})(1 - F_{T_i+1}) & \text{if } a_i = ++ \end{cases}$$

It is easy to check that $Y_i \leq X_i$ for $1 \leq i \leq M$, for every sequence of decisions of an algorithm, and every outcome of $F_1, \ldots, F_M$ and $R_1, \ldots, R_M$. Note that $Y_2, Y_4, \ldots, Y_{\lfloor \frac{M}{2} \rfloor}$ are i.i.d. Bernoulli random variables with parameter $1/4$. Let $U$ be the number of unmatched points by ALG then

$U + 2M = 2n$ and $\sum_{i=1}^{M} X_i \leq U$ thus $\sum_{i=1}^{\lfloor \frac{M}{2} \rfloor} Y_{2i} \leq \sum_{i=1}^{M} X_i \leq 2n - 2M$. We have:

$$P\{M \geq \alpha n\} = P\{\sum_{i=1}^{M} X_i \leq 2n - 2M, M \geq \alpha n\} \leq P\{\sum_{i=1}^{\alpha n/2} Y_{2i} \leq 2(1-\alpha)n\}$$

For $\alpha \in (16/17, 1)$ by Fact 42:

$$P\{M \geq \alpha n\} \leq 2^{-\frac{\alpha n}{2} D(\frac{4(1-\alpha)}{\alpha} || 1/4)}$$

To conclude the statement of the theorem, suppose we have an algorithm that guarantees at most $2(1-\alpha)n$ unmatched points with $f(\alpha)n$ bits of advice. If we replace advice bits with random bits and run this algorithm then with probability at least $2^{-f(\alpha)n}$ the algorithm creates at most $2(1-\alpha)n$ unmatched points. This implies that $f(\alpha) \geq \frac{\alpha}{2} D(\frac{4(1-\alpha)}{\alpha} || 1/4)$. □

# Chapter 6

# Online Interval Selection

In this chapter,[1] we study the *Online Interval Selection Problem* in both the random order and any order (adversarial) input models. Additionally, we provide a lower bound on the advice complexity of this problem. In both input models, we restrict the interval sets to *simple chains*, defined as follows.

**Definition 3.** *The interval graph of a set of intervals is a graph where each interval is represented by a node, and there is an edge between two nodes if their corresponding intervals overlap. A set of intervals $I = \{I_1, I_2, \ldots, I_n\}$ is called a simple chain if its interval graph forms a path, with the intervals ordered from left to right (see Figure 6.1).*

## 6.1 Introduction

In the Online Interval Selection problem, a set of intervals $I = \{I_1, I_2, \ldots, I_n\}$ arrives one by one, and the algorithm must decide whether to accept or reject each interval as it arrives, ensuring that the selected intervals do not overlap. In the unweighted version of this problem, discussed in this chapter, the objective is to maximize the number of selected intervals. In the revocable acceptance setting, the algorithm is allowed to revoke previously selected intervals to accept new ones, while maintaining a set of non-overlapping intervals at all times.

[1] The results presented in this chapter are based on joint work by the author in collaboration with Yaqiao Li and Denis Pankratov submitted to the journal of the Discrete Applied Mathematics.

In most of the Interval Selection literature, the *real-time model* is commonly studied, where intervals arrive in increasing order of their start times. In this model, Faigle and Nawijn [28] presented a deterministic greedy algorithm with revoking that achieves the optimal competitive ratio of 1. However, Lipton and Tomkins [46] showed that, without revoking, even randomized algorithms can have an unbounded competitive ratio.

In the *any-order (adversarial)* model, the sequence of intervals arrives in the adversarial order. Garay et al. [31] demonstrated that, even with revoking, any deterministic algorithm can still have an unbounded competitive ratio in this adversarial model. Borodin and Karavasilis [12] proposed a deterministic algorithm with revoking for intervals of $k$ different lengths, achieving a competitive ratio of $1/2k$, which they proved to be optimal. In a subsequent paper [13], they showed that the same algorithm achieves a competitive ratio of 0.4 in the *random-order* model, where intervals are adversarially chosen but arrive in a random sequence.

Borodin and Karavasilis [12] also explored *memoryless* algorithms, which only retain the set of selected intervals without tracking previous rejections. They showed that the $1/2k$ upper bound applies to randomized memoryless algorithms as well. Furthermore, they demonstrated that for unit-length intervals in the random-order model, only algorithms that revoke in one direction can benefit from the random order and achieve a competitive ratio better than $1/2$.

In Section 6.2, we consider a simple deterministic greedy algorithm that only revokes to the left. We prove that this algorithm achieves a competitive ratio of $2(1-1/\sqrt{e}) \approx 0.786$ on a simple chain in the random order model. However, in this specific setting, this algorithm performs worse than the basic greedy algorithm without revoking, which has a competitive ratio of $(1-1/e^2) \approx 0.864$. In Section 6.3 we show an upper-bound of $3/4$ for any deterministic algorithm with revoking on a simple chain in the adversarial model. Finally in Section 6.4 we prove a lower bound of $n/4$ for the advice complexity of Online Interval Selection using only a simple chain, which, to the best of our knowledge, is the first lower bound on the advice complexity of this problem.

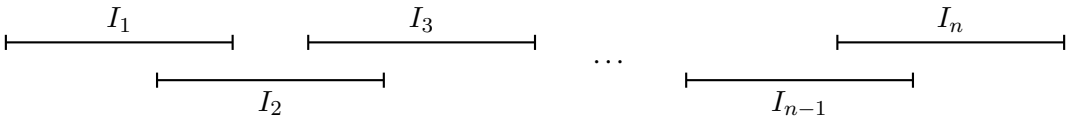



Figure 6.1: A simple chain of intervals with size $n$.

## 6.2 Random Order Model

Before introducing the main algorithm and in order to have some context, let us consider the basic greedy algorithm without revoking that selects an interval whenever it does not overlap other selected intervals. We will show that the performance of this algorithm on a simple chain in random order model is the same as the Unfriendly Seating Arrangement problem by Freedman and Shepp [30, problem 62-3] defined as follows.

**Definition 4** (Unfriendly Seating Arrangement)**.** *There are $n$ seats in a row at a luncheonette and people sit down one at a time at random. They are unfriendly and so never sit next to one another (no moving over). What is the expected number of persons to sit down?*

**Fact 44** (Freedman and Shepp [30])**.** *The expected number of seats taken in the Unfriendly Seating Arrangement goes to $n(1-\frac{1}{e^2})/2+O(1)$.*

Next we compute the competitive ratio of the basic greedy algorithm by a showing a one-to-one mapping between the decisions of this algorithm and set of seated people in the Unfriendly Seating Arrangement problem.

**Theorem 45.** *The basic greedy algorithm achieves a competitive ratio $(1-\frac{1}{e^2}) \approx 0.864$ when the input sequence is simple chain in the random order model.*

*Proof.* Suppose the intervals arrive according to an arbitrary permutation $\sigma = \langle \sigma_1 \sigma_2 \dots \sigma_n \rangle$ on $[n]$, meaning the $i^{\text{th}}$ arriving interval is the $\sigma_i^{\text{th}}$ interval from the left in the chain. We aim to prove that if people arrive according to $\sigma$ in the Unfriendly Seating Arrangement problem (where the $i^{\text{th}}$ arriving person wants to sit in the $\sigma_i^{\text{th}}$ chair), then the $i^{\text{th}}$ interval will be selected by the basic greedy algorithm if and only if the $i^{\text{th}}$ person is seated.

Initially, no one is seated, and no interval is selected. Inductively, assume the claim holds before step $i$. When the $i^{\text{th}}$ interval arrives, it will be selected if and only if the $\sigma_i - 1$ and $\sigma_i + 1$ intervals (if they exist) are not selected. This happens if and only if, by the induction hypothesis, the seats numbered $\sigma_i - 1$ and $\sigma_i + 1$ are not occupied. On the other hand, the $i^{\text{th}}$ person will be seated if and only if the $\sigma_i - 1$ and $\sigma_i + 1$ seats are not occupied. Hence, the $i^{\text{th}}$ person will be seated if and only if the basic greedy algorithm selects the $i^{\text{th}}$ interval, which proves the claim.

Since for every permutation $\sigma$, the number of selected intervals equals the number of seated people, their expectations are also equal when they arrive in random order. By Fact 44 the asymptotic competitive ratio of basic greedy is $(1 - \frac{1}{e^2})$. □

Now we consider the "Revoke-to-the-Left" (RevtoL) algorithm. It can be defined for general input sequences not only chains. When an interval $I_i = [s_i, f_i]$ arrives, if it overlaps with a selected interval $I_j = [s_i, f_j]$ such that $s_i < f_j < f_i$ (i.e., $I_j$ is on the left side of $I_i$), $I_i$ is rejected. Otherwise, $I_i$ is accepted, revoking any previously selected interval that overlaps with $I_i$ (see Figure 6.2).

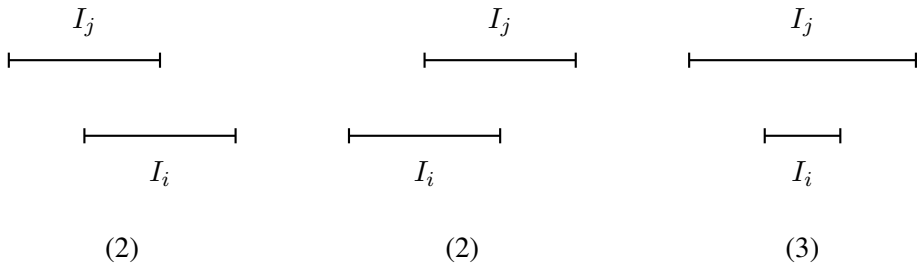



Figure 6.2: In all cases, $I_j$ is already selected when $I_i$ arrives. In (1), $I_i$ is rejected. In (2) and (3), $I_j$ is revoked, and $I_i$ is accepted. However, in a simple chain setting, scenario (3) would not occur. The RevtoL algorithm handles general inputs, but in this work, we analyze it specifically for simple chain graphs under the random-order input model.

Here, we analyze the algorithm. Let $n$ denote the number of intervals. An online instance is simply a permutation of these $n$ intervals, and the intervals are presented to the algorithm in the order given by the permutation.

**Theorem 46.** *The RevtoL algorithm achieves an asymptotic competitive ratio of* $2(1-\frac{1}{\sqrt{e}}) \approx 0.786$ *on a simple chain in the random-order model.*

To prove this theorem, we define some notations. Let $I$ be a simple chain with size $n$, $S_n$ the set of all permutations on $[n]$, and $\sigma$ a permutation in $S_n$. Let RevtoL $(I, \sigma)$ be the set of intervals chosen by RevtoL on the set $I$ when they arrive in the order of $\sigma$. Let $d_{n,i} = \#\{\sigma \in S_n : \text{ interval } i \text{ is chosen by RevtoL}\}$, $d_n := d_{n,n}$, and $D_n := \sum_{i=1}^{n} d_{n,i}$. By definition, $D_n = \sum_{\sigma \in S_n} \text{RevtoL}(I, \sigma)$, which means the competitive ratio of RevtoL is $D_n/n!$.

The following is some data for the first few $n$.

Table 6.1: Data for $d_n, D_n$.

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $d_n$ | 1 | 0 | 4 | 6 | 56 | 260 |
| $D_n$ | 1 | 2 | 10 | 46 | 286 | 1976 |
| $d_{n,n} - d_{n,n-1}$ | 1 | $-2$ | 4 | $-10$ | 26 | $-76$ |

**Theorem 47.** *For $n \geq 3$, $d_n = (n-1)(D_{n-2} + d_{n-2})$.*

**Theorem 48.** *For $n \geq 4$,*

$$d_{n,n} - d_{n,n-1} = 2d_{n-1,n-2} - d_{n-1,n-1} - d_{n-1,n-3}. \tag{3}$$

The following is a very useful fact.

**Lemma 49.** *For $1 \leq i \leq n-1$,*

$$d_{n,i} = nd_{n-1,i} = n(n-1)\cdots(i+1)d_i.$$

*Proof.* It suffices to show $d_{n,i} = nd_{n-1,i}$. This is because for every $1 \leq i \leq n-1$, the presence of interval $n$ does not influence whether interval $i$ is chosen or not chosen. This is because interval $n$ has the least priority and RevtoL does not revoke or reject any other interval because of interval $n$. Since interval $n$ can be in $n$ different positions, the equality follows. □

**Claim.** Theorem 47 and Theorem 48 are equivalent.

*Proof.* Theorem 47 $\Longrightarrow$ Theorem 48: Theorem 47 says

$$D_{n-2} = \frac{d_n}{n-1} - d_{n-2}. \tag{4}$$

Apply Theorem 47 for $n-1$, we get $d_{n-1} = (n-2)(D_{n-3} + d_{n-3})$, hence

$$(n-2)D_{n-3} = d_{n-1} - (n-2)d_{n-3}. \tag{5}$$

By the definition of $D_{n-2}$, one has

$$D_{n-2} = (n-2)D_{n-3} + d_{n-2}. \tag{6}$$

Combine these three equations, we get

$$\frac{d_n}{n-1} - d_{n-2} = D_{n-2} = (n-2)D_{n-3} + d_{n-2} = d_{n-1} - (n-2)d_{n-3} + d_{n-2}. \tag{7}$$

Apply Lemma 49, this is

$$d_n - d_{n-1,n-2} = d_{n,n-1} - d_{n-1} - d_{n-1,n-3} + d_{n-1,n-2}. \tag{8}$$

Rearrange (8) gives the claimed equality (3).

Theorem 47 $\Longleftarrow$ Theorem 48: we prove Theorem 47 by induction. The base case is true from Table 6.1. Now assume the claim of Theorem 47 is true for $n-1$, i.e., assume (5), then we can deduce (4) by equation (5), (6), and (8). □

Now we prove Theorem 48.

*Proof of Theorem 48.* We give a formula for $d_n$. For $1 \leq i \leq n$, define

$$p_i = \#\{\sigma \in S_n : \text{ interval } n \text{ is chosen by RevtoL, while interval } i \text{ is the last interval of the input}\}. \tag{9}$$

Then, $d_n = p_1 + \ldots + p_n$. Below we give formulas for computing $p_i$.

- $p_n = (n-1)! - d_{n-1}$.

  This is because in this case interval $n$ is chosen iff interval $n-1$ is *not* chosen among intervals $1, \ldots, (n-1)$.

- $p_{n-1} = (n-1)d_{n-2} = d_{n-1,n-2}$.

  This is because in this case interval $n$ is chosen if and only if interval $n-2$ is chosen among intervals $1, \ldots, (n-2)$, and we have the multiplicative factor $n-1$ because interval $n$ can be arbitrarily placed in any of the $n-1$ positions.

- for $2 \leq i \leq n-2$, $p_i = (n-1)\cdots(n-i+1)d_{n-i} = d_{n-1,n-i}$.

  Consider the two blocks of intervals: intervals $1, \ldots, (i-1)$, and intervals $(i+1), \ldots, n$. Since interval $i$ is not present when these $n-1$ intervals appear, we know that these two blocks of intervals do not interfere with each other. Hence, interval $n$ is chosen if and only if it is chosen among intervals $(i+1), \ldots, n$. But this is equivalent to interval $n-i$ is chosen among intervals $1, \ldots, (n-i)$. Since the other block of intervals $1, \ldots, (i-1)$ can be arbitrarily placed in $n-1$ positions, we have the multiplicative factor $(n-1)\cdots(n-i+1)$.

- $p_1 = d_{n-1}$.

  This is because in this case interval $n$ is chosen if and only if interval $n$ is chosen among intervals $2, \ldots, n$, which is equivalent to interval $n-1$ is chosen among intervals $1, \ldots, (n-1)$.

Note that we have applied Lemma 49 whenever applicable in the above formulas for $p_i$. Hence, we have

$$d_n = (n-1)! + 2d_{n-1,n-2} + \sum_{j=2}^{n-3} d_{n-1,j}. \tag{10}$$

With this, we have (apply Lemma 49 whenever necessary)

$$d_{n,n} - d_{n,n-1} = d_{n,n} - nd_{n-1,n-1} = d_{n,n} - (n-1)d_{n-1,n-1} - d_{n-1,n-1}$$

$$= \left((n-1)! + 2d_{n-1,n-2} + \sum_{j=2}^{n-3} d_{n-1,j}\right) - (n-1)\left((n-2)! + 2d_{n-2,n-3} + \sum_{j=2}^{n-4} d_{n-2,j}\right) - d_{n-1,n-1}$$

$$= \left((n-1)! + 2d_{n-1,n-2} + \sum_{j=2}^{n-3} d_{n-1,j}\right) - \left((n-1)! + 2d_{n-1,n-3} + \sum_{j=2}^{n-4} d_{n-1,j}\right) - d_{n-1,n-1}$$

$$= 2d_{n-1,n-2} - d_{n-1,n-1} - d_{n-1,n-3}$$

as claimed. □

Let us define an auxiliary integer sequence $a_n$ as follows. The recurrence relation is given by $a_n = a_{n-1} + (n-1)a_{n-2}$, with initial conditions $a_0 = a_1 = 1$. The first few elements of this sequence are $1, 1, 2, 4, 10, 26, 76, 232, \ldots$. This sequence is known to represent the number of self-inverse permutations as well as other important combinatorial objects [56]. Here, we focus only on the recursive definition of $a_n$.

**Corollary 50.**

$$d_{n,n} - d_{n,n-1} = (-1)^{n-1} a_n$$

*Proof.* The base case can be verified from Table 6.1. Then, it suffices to prove $a_n = (-1)^{n-1}(d_{n,n} - d_{n,n-1})$ satisfy the recurrence formula of $a_n$, i.e., $a_n = a_{n-1} + (n-1)a_{n-2}$. Indeed, by Theorem 48 and Lemma 49,

$$\begin{aligned} d_{n,n} - d_{n,n-1} &= 2d_{n-1,n-2} - d_{n-1,n-1} - d_{n-1,n-3} \\ &= -(d_{n-1,n-1} - d_{n-1,n-2}) + d_{n-1,n-2} - d_{n-1,n-3} \\ &= -(d_{n-1,n-1} - d_{n-1,n-2}) + (n-1)(d_{n-2,n-2} - d_{n-2,n-3}). \end{aligned}$$ □

**Corollary 51.** $D_n = (n+1)D_{n-1} + (-1)^{n-1} a_{n-1}$.

*Proof.* By Theorem 47, Lemma 49, and Corollary 50, we have

$$\begin{aligned} D_n = \sum_{i=1}^{n} d_{n,i} &= n\left(\sum_{i=1}^{n-1} d_{n-1,i}\right) + d_{n,n} \\ &= nD_{n-1} + (n-1)(D_{n-2} + d_{n-2}) \\ &= nD_{n-1} + D_{n-1} - d_{n-1,n-1} + d_{n-1,n-2} \\ &= (n+1)D_{n-1} + (-1)^{n-1} a_{n-1}. \end{aligned}$$ □

**Corollary 52.**

$$\frac{D_n}{(n+1)!} = \sum_{i=0}^{n-1} \frac{(-1)^i a_i}{(i+2)!}$$

*Proof.* Let us prove this by induction. For the base case $n = 1$ this is trivial. Assuming it is true for $n = k$ let us write down the the induction step for $n = k + 1$ expanding $D_{k+1}$ by Corollary 51.

$$\frac{D_{k+1}}{(k+2)!} = \frac{(k+2)D_k + (-1)^k a_k}{(k+2)!} = \frac{D_k}{(k+1)!} + \frac{(-1)^k a_k}{(k+2)!} =$$

$$= \sum_{i=0}^{k-1} \frac{(-1)^i a_i}{(i+2)!} + \frac{(-1)^k a_k}{(k+2)!} = \sum_{i=0}^{k} \frac{(-1)^i a_i}{(i+2)!}$$

Which proves the claim. □

**Lemma 53.**

$$\lim_{n\to\infty} \frac{D_n}{(n+1)!} = \sum_{i=0}^{\infty} \frac{(-1)^i a_i}{(i+2)!} = 1 - \frac{1}{\sqrt{e}}. \tag{11}$$

*Proof.* We aim to evaluate the infinite series

$$S = \sum_{n=0}^{\infty} \frac{(-1)^n a_n}{(n+2)!}.$$

To accomplish this, we will employ generating functions and integral representations. First, we consider the exponential generating function for the sequence $\{a_n\}$:

$$G(x) = \sum_{n=0}^{\infty} \frac{a_n x^n}{n!}.$$

We claim

$$G''(x) = (1 + x)G'(x) + G(x). \tag{12}$$

Indeed, a direct calculation gives

$$G'(x) = \sum_{n=0}^{\infty} \frac{a_{n+1}}{n!} x^n,$$
$$G''(x) = \sum_{n=0}^{\infty} \frac{a_{n+2}}{n!} x^n.$$

Hence, by the formula $a_n = a_{n-1} + (n-1)a_{n-2}$, we have

$$\begin{aligned} G''(x) - G'(x) = \sum_{n=0}^{\infty} \frac{a_{n+2} - a_{n+1}}{n!} x^n &= \sum_{n=0}^{\infty} \frac{(n+1)a_n}{n!} x^n \\ &= \left( \sum_{n=0}^{\infty} \frac{a_n}{n!} x^{n+1} \right)' \\ &= (xG(x))' = G(x) + xG'(x), \end{aligned}$$

as claimed. Now, solving the differential equation (12) with the boundary condition $G(0) = 1$ and $G'(0) = 1$, we get

$$G(x) = e^{\frac{x(x+2)}{2}}.$$

Next, we express $S$ in terms of the generating function $G(x)$. Observe that

$$S = \sum_{n=0}^{\infty} \frac{(-1)^n a_n}{n!} \cdot \frac{1}{(n+1)(n+2)}.$$

To relate this to the generating function $G(x)$, we use the integral representation

$$\frac{1}{(n+1)(n+2)} = \int_0^1 x^n (1-x)\, dx.$$

Substituting this into the expression for $S$, we obtain

$$S = \sum_{n=0}^{\infty} \frac{(-1)^n a_n}{n!} \int_0^1 x^n (1-x)\, dx = \int_0^1 (1-x) \sum_{n=0}^{\infty} \frac{a_n (-x)^n}{n!}\, dx.$$

Recognizing the sum inside the integral as the generating function evaluated at $-x$, we have

$$\sum_{n=0}^{\infty} \frac{a_n (-x)^n}{n!} = G(-x) = e^{-x + \frac{x^2}{2}}.$$

Therefore, the sum $S$ can be expressed as

$$S = \int_0^1 (1-x) e^{-x + \frac{x^2}{2}}\, dx.$$

To evaluate the integral

$$S = \int_0^1 (1-x)e^{-x+\frac{x^2}{2}}\,dx,$$

we perform a substitution to simplify the exponent. Notice that

$$-x + \frac{x^2}{2} = -\frac{1}{2} + \frac{(x-1)^2}{2}.$$

Substituting this into the integral, we obtain

$$S = e^{-\frac{1}{2}} \int_0^1 (1-x)e^{\frac{(x-1)^2}{2}}\,dx.$$

Next, we perform a change of variable to further simplify the integral. Let

$$u = x - 1 \quad \Rightarrow \quad du = dx.$$

When $x = 0$, $u = -1$, and when $x = 1$, $u = 0$. Substituting these into the integral, we have

$$S = e^{-\frac{1}{2}} \int_{-1}^0 (-u)e^{\frac{u^2}{2}}\,du = -e^{-\frac{1}{2}} \int_{-1}^0 ue^{\frac{u^2}{2}}\,du.$$

To evaluate the integral, we use substitution again. Let

$$v = \frac{u^2}{2} \quad \Rightarrow \quad dv = u\,du.$$

Thus,

$$\int ue^{\frac{u^2}{2}}\,du = \int e^v\,dv = e^v + C = e^{\frac{u^2}{2}} + C.$$

Evaluating from $u = -1$ to $u = 0$, we obtain

$$\int_{-1}^0 ue^{\frac{u^2}{2}}\,du = \left[e^{\frac{u^2}{2}}\right]_{-1}^0 = e^0 - e^{\frac{1}{2}} = 1 - e^{\frac{1}{2}}.$$

Substituting back, we find

$$S = -e^{-\frac{1}{2}}(1 - e^{\frac{1}{2}}) = 1 - e^{-\frac{1}{2}}.$$

□

Now we have everything to prove Theorem 46.

*Proof of Theorem 46.* Recall that by definition $D_n/n!$ is the expected number of segments chosen by RevtoL on a simple chain of size $n$ in the random order model. It is easy to see that OPT in this setting is always $\lceil n/2 \rceil$. Thus by applying Lemma 53 the asymptotic competitive ratio of RevtoL is

$$\rho(\text{RevtoL}) = \lim_{n\to\infty} \frac{D_n}{n!\lceil n/2 \rceil} = \lim_{n\to\infty} \frac{2D_n}{(n+1)!} = 2(1 - \frac{1}{\sqrt{e}})$$

□

## 6.3 Adversarial Model

Next we show an upper bound on deterministic algorithms with revoking in adversarial model when it is guaranteed that the input is a simple chain. We say that interval $I$ overlaps interval $I'$ on the left if $I$ starts before $I'$ and ends before $I'$ finishes.

**Theorem 54.** *No deterministic algorithm with revoking can achieve an asymptotic competitive ratio better than $3/4$ in the adversarial model, even when the input is guaranteed to form a simple chain and the intervals have unique lengths.*

*Proof.* Fix a deterministic algorithm ALG. The adversarial input begins with $k$ pairs of intervals $(I_1, J_1), (I_2, J_2), \ldots, (I_k, J_k)$ such that $I_i$ overlaps $J_i$ on the left and intervals from different pairs do not overlap (see Figure 6.3). Because of revoking, without loss of generality we can assume that ALG chooses exactly one interval in each pair. Note that up to this point the number of selected intervals is $k$.

Now, to connect these disconnected pairs and form a simple chain, we look at the decisions made by ALG. For connecting the $i^{\text{th}}$ and $(i+1)^{\text{th}}$ pairs, we consider the ALG decision on $J_i$ and $I_{i+1}$ and we call it $i^{\text{th}}$ *gap decision*. If we denote accept as A and reject as R, a gap decision for $J_i$ and $I_{i+1}$ can be RR, RA, AR, or AA. If only one of $J_i$ and $I_{i+1}$ is selected (RA or AR), we introduce a new interval $X_i$ that intersects only these two intervals (see Figure 6.3 (1)). In this

case, selecting $X_i$ and revoking the previously selected interval will not increase the total number of selected intervals.

If neither or both of $J_i$ and $I_{i+1}$ are selected (RR or AA), we introduce two new intervals $Y_i$ and $Z_i$, where $Y_i$ intersects only $J_i$ and $Z_i$, and $Z_i$ intersects only $Y_i$ and $I_{i+1}$ (see Figure 6.3 (2)). In the case of AA, to select $Y_i$ and $Z_i$ ALG should revoke $J_i$ and $I_{i+1}$ respectively, and therefore it cannot increase the number of selected intervals. In the case of RR, ALG can accept one of $Y_i$ or $Z_i$ which increases the total number of selected intervals by one.

Let $k_{RR}$ be the number of RR gap decisions. By the above discussion, the number of selected intervals can be at most $k + k_{RR}$. Two consecutive gap decisions cannot both be RR (otherwise, there would exist a pair $(I_i, J_i)$ such that both are rejected), implying $k_{RR} \leq \frac{k}{2}$. Since in each pair $(I_i, J_i)$ exactly one interval is selected, there must be an AA gap between every two RR gaps. Therefore, there are at least $2k_{RR} - 1$ gap decisions that are either RR or AA, and we placed two new intervals between them. We placed at least one interval between the remaining pairs, and thus, the total number of intervals is at least

$$\underbrace{2k}_{\text{pairs}} + \underbrace{(k-1)}_{\text{all gaps}} + \underbrace{(2k_{RR}-1)}_{\text{RRs and AAs}} = 3k + 2k_{RR} - 2.$$

Let $n$ be the number of intervals. Since we are interested in asymptotic analysis, we can say $3k + 2k_{RR} \leq n$. Putting all the above together, we have the following linear programming problem:

$$\begin{aligned} \text{Maximize:} \quad & k + k_{RR} \\ \text{Subject to:} \quad & k_{RR} \leq \frac{k}{2} \\ & 3k + 2k_{RR} \leq n \end{aligned}$$

Which achieves a maximum value of $3n/8$. On the other hand, OPT achieves $\lceil n/2 \rceil$, proving a competitive ratio of $3/4$. The only remaining task is to show that we can construct the described input using intervals of unique lengths. We place the intervals as follows: $I_i = [12i, 12i + 5]$, $J_i = [12i{+}4, 12i{+}9]$, $X_i = [12i{+}8, 12i{+}13]$, $Y_i = [12i{+}6, 12i{+}11]$, and $Z_i = [12i{+}10, 12i{+}15]$. It is straightforward to verify that by placing the intervals in this pattern, we achieve the desired

topology and ensure the required intersections between intervals.

□

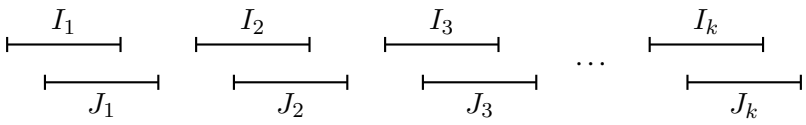
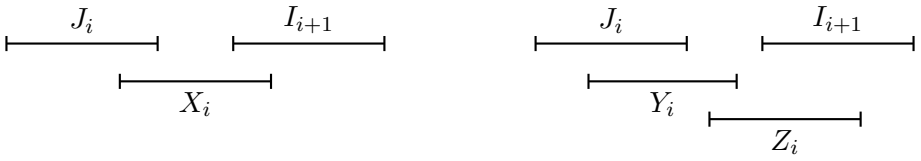



Figure 6.3: The first $2k$ intervals.




(1) (2)

Figure 6.4: Connecting $i^{\text{th}}$ and $i+1^{\text{th}}$ pairs by putting (1) one interval (2) two intervals, in the gap between $J_i$ and $I_i$

.

## 6.4 Advice Complexity

In this chapter, we consider the advice complexity of the interval Selection problem without revoking, under the condition that the input forms a simple chain. It is trivial that $n$ bits of advice are sufficient to achieve an optimal solution, even without restrictions on the input sequence (one bit per interval indicating whether the algorithm should select it or not). Here, we establish a constant-factor lower bound on the advice complexity for this problem.

**Theorem 55.** *The advice complexity of solving interval Selection of a simple chain is at least* $n/4$.

*Proof.* We construct a family of $2^{k-1}$ input sequences similar to those used in the proof of Theorem 54. All input sequences begin with a prefix of $k$ pairs of intervals $(I_1, J_1), (I_2, J_2), \ldots, (I_k, J_k)$. For each $i \in [k]$ intervals $I_i$ overlaps $J_i$ on the left and intervals from different pairs do not overlap. These intervals are arranged sequentially from left to right and arrive in the same order (see Figure 6.3).

After this prefix, we can connect pairs to make a chain by placing either one or two intervals in the gap between them (see Figure 6.3), which results in $2^{k-1}$ different input sequences. For each input sequence, we extend the chain by adding intervals at the end, ensuring that all input sequences have exactly $4k+1$ intervals. In each sequence, if we number the intervals $1, \ldots, 4k+1$, the optimal solution is to select the odd-numbered intervals.

The parity assigned to the intervals in the prefix differs for each input sequence, which means that $2^{k-1}$ distinct algorithms would be required to make the correct decisions for the first $2k$ intervals. Consequently, at least $k-1$ bits of advice are needed. □

# Chapter 7

# Conclusion

In this concluding chapter, we summarize the key findings and contributions of the thesis, highlighting the progress made in tackling the central problems introduced throughout the work. We reflect on the theoretical advancements achieved across the different problem settings, and we discuss the implications of these results for future research in the field. Additionally, we identify open questions that arise from our results, paving the way for further investigation into the challenges and opportunities presented by the studied problems.

## 7.1 Maximum Weight Convex Polytope

We extended our understanding of the complexity of MWCP as a function of the ambient dimension $d$. Based on our findings and the previous work of Bautista et al. [9], the complexity landscape is as follows: For $d = 1$, MWCP is solvable in $O(n \log n)$ time exactly (Theorem 7). In the case of $d = 2$, MWCP is solvable in $O(n^3)$ time, as shown by Bautista et al. [9], along with an alternative algorithm presented in this thesis. For $d = 3$, we demonstrated that MWCP is not solvable in polynomial time unless $\mathcal{P} = \mathcal{NP}$. For $d \geq 4$, MWCP becomes $\mathcal{NP}$-hard to approximate within a factor of $n^{1/2-\epsilon}$ for any $\epsilon > 0$.

The above list immediately suggests several open problems, the following two of which are of particular interest:

**Open Problem 1.** *Find an algorithm with better time complexity than $O(n^3)$ for* MWCP *in* 2 *dimensions or prove a lower bound probably with some fine-grained hypothesis.*

**Open Problem 2.** *Determine if* MWCP *can be approximated within a constant factor in* 3 *dimensions.*

We conjecture that the answer to the first open problem is that there is no algorithm significantly faster than $O(n^3)$.

## 7.2 Weighted Online Non-Crossing Matching

We introduced the weighted version of the Online Non-Crossing Matching Problem, where the objective is to maximize the total weight of the matched pairs. We first showed that deterministic algorithms can have arbitrarily bad competitive ratios under the adversarial input model. To address this challenge, we explore three distinct approaches: (1) Parameterizing the weights by bounding them within the range $[1, U]$, (2) Randomization, and (3) Allowing revoking of previously made matches.

In the bounded weight version, we presented a deterministic algorithm with a competitive ratio of $\Omega\left(2^{-2\sqrt{\log U}}\right)$. Furthermore, we established an upper bound of $O\left(2^{-\sqrt{\log U}}\right)$ for the competitive ratio of any deterministic algorithm in this setting.

For the case where revoking is permitted, we developed an algorithm that achieves a competitive ratio of approximately $0.28$. We also proved that in the unweighted case, no deterministic algorithm can exceed a competitive ratio of $2/3$. This indicates that while revoking does not enhance the performance of deterministic algorithms in the unweighted version, it does provide significant advantages in the weighted setting.

Next, we introduced a randomized algorithm with a competitive ratio of $1/3$, and showed that in the unweighted case, no randomized algorithm can exceed a competitive ratio of $8/9$. It is worth noting that the constant competitive ratios achieved by both the deterministic algorithm with revoking and the randomized algorithm hold for arbitrary weight functions.

Finally, we considered the version of the problem where all points arrive on a line. We showed that a deterministic algorithm with revoking and a randomized algorithm without revoking can

have arbitrary bad competitive ratio even in the unweighted version. However, we introduced a randomized algorithm with revoking that achieves a competitive ratio of $0.5$ in the unweighted version.

**Open Problem 3.** *Can a randomized algorithm with revoking achieve a constant competitive ratio in the weighted version when points are collinear?*

## 7.3 Online Non-Crossing Matching with Advice

We showed that the advice complexity of solving BNM on a circle (or, more generally, on inputs in a convex position) is tightly bounded by the logarithm of the $n^{\text{th}}$ Catalan number from above and below. This result corrects the previous claim of Bose et al. [15] that the advice complexity is $\log(n!)$. At the heart of the result was a connection between non-crossing constraints in online inputs and the 231-avoiding property of permutations of $n$ elements. The advice complexity of BNM on a plane is left as an open problem:

**Open Problem 4.** *What is the advice complexity of* BNM *on a plane?*

We gave the SAM algorithm to achieve $\log C_n$ advice complexity of MNM on a plane which can be seen as the extension of ASAP. We aslo exponentially improved the lower bound on the advice complexity of solving MNM optimally from $O(\log n)$ (due to Bose et al. [15]) to $n/3 - 1$.

**Open Problem 5.** *What is the advice complexity of* MNM *on a plane and a circle?*

We also established $\Omega_\alpha(n)$ lower bound for achieving competitive ratio $\alpha \in (16/17, 1)$. All our lower bounds are obtained on input points that are located on a common circle. The non-crossing constraint presented an obstacle to using standard proof techniques, such as a reduction from the string guessing problem. This motivates the following open problem.

**Open Problem 6.** *Does there exist a reduction from the string guessing problem to* MNM/BNM *on a circle/plane?*

## 7.4 Online Interval Scheduling

We demonstrated that this algorithm achieves a competitive ratio of $2(1 - 1/\sqrt{e}) \approx 0.786$ on a simple chain in the random order model, which, in this case, is outperformed by the basic greedy algorithm without revoking, which has a competitive ratio of $(1 - 1/e^2) \approx 0.864$. Additionally, we established an upper bound of $3/4$ for any deterministic algorithm with revoking on a simple chain in the adversarial model. Finally, we provided a lower bound of $n/4$ for the advice complexity of Online Interval Selection using only a simple chain.

A natural generalization of the simple chain is the $k$-chain, where each interval intersects with $k$ neighboring intervals on either side. The following open problem offers a potential direction for gaining deeper insights into the performance of deterministic algorithms with revoking in the random order model.

**Open Problem 7.** *What is competitive ratio of RevtoL in the random order model when the input is a $k$-chain?*

# Bibliography


[1] A. Aamand, J. Chen, and P. Indyk. Optimal online bipartite matching with degree information. *NeurIPS*, 35:5724–5737, 2022.

[2] M. A. Abam, A. Mohammad Lavasani, and D. Pankratov. Maximum weight convex polytope. In *Proc. of CCCG*, pages 257–263, 2022.

[3] M. A. Alomrani, R. Moravej, and E. B. Khalil. Deep policies for online bipartite matching: A reinforcement learning approach. *Transactions on Machine Learning Research*, 2022.

[4] G. Aloupis, J. Cardinal, S. Collette, E. D. Demaine, M. L. Demaine, M. Dulieu, R. Fabila-Monroy, V. Hart, F. Hurtado, S. Langerman, M. Saumell, C. Seara, and P. Taslakian. Non-crossing matchings of points with geometric objects. *Computational Geometry*, 46(1):78–92, 2013.

[5] S. Angelopoulos, C. Dürr, S. Jin, S. Kamali, and M. Renault. Online computation with untrusted advice. In *Proc. of ITCS*, pages 52:1–52:15, 2020.

[6] M. J. Atallah. A matching problem in the plane. *Journal of Computer and System Sciences*, 31(1):63–70, 1985.

[7] B. Bahmani and M. Kapralov. Improved bounds for online stochastic matching. In *Proc. of European Symposium on Algorithms*, pages 170–181, 2010.

[8] K. Barhum, H.-J. Böckenhauer, M. Forišek, H. Gebauer, J. Hromkovič, S. Krug, J. Smula, and B. Steffen. On the power of advice and randomization for the disjoint path allocation problem.

In *Proc. of International Conference on Current Trends in Theory and Practice of Informatics*, pages 89–101, 2014.

[9] C. Bautista-Santiago, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urrutia, and I. Ventura. Computing optimal islands. *Operational Research Letters*, 39(4):246–251, 2011.

[10] B. Birnbaum and C. Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, 2008.

[11] H.-J. Böckenhauer, J. Hromkovič, D. Komm, S. Krug, J. Smula, and A. Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theoretical Computer Science*, 554:95–108, 2014.

[12] A. Borodin and C. Karavasilis. Any-order online interval selection. In *International Workshop on Approximation and Online Algorithms*, pages 175–189. Springer, 2023.

[13] A. Borodin and C. Karavasilis. Random-order interval selection. *arXiv preprint arXiv:2407.20941*, 2024.

[14] A. Borodin, D. Pankratov, and A. Salehi-Abari. On conceptually simple algorithms for variants of online bipartite matching. *Theory of Computing Systems*, 63(8):1781–1818, 2019.

[15] P. Bose, P. Carmi, S. Durocher, S. Kamali, and A. Sajadpour. Non-crossing matching of online points. In *Proc. of CCCG*, 2020.

[16] J. Boyar, L. M. Favrholdt, C. Kudahl, K. S. Larsen, and J. W. Mikkelsen. Online algorithms with advice: a survey. *ACM SIGACT News*, 47(3):93–129, 2016.

[17] J. Boyar, S. Kamali, K. S. Larsen, A. M. Lavasani, Y. Li, and D. Pankratov. On the online weighted non-crossing matching problem. In *Proc. of SWAT*, 2024.

[18] B. Brubach, K. A. Sankararaman, A. Srinivasan, and P. Xu. New algorithms, better bounds, and a novel model for online stochastic matching. In *Proc. of European Symposium on Algorithms*, 2016.

[19] S. Cohen. *Finding color and shape patterns in images*. PhD dissertation, Stanford University, 1999.

[20] C. Cortés, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, J. Urrutia, and I. Ventura. Bichromatic separability with two boxes: A general approach. *Journal of Algorithms*, 64(2-3):79–88, 2009.

[21] G. Das and M. T. Goodrich. On the complexity of optimization problems for 3-dimensional convex polyhedra and decision trees. *CGTA*, 8(3):123–137, 1997.

[22] N. R. Devanur, K. Jain, and R. D. Kleinberg. Randomized primal-dual analysis of ranking for online bipartite matching. In *Proc. of SODA*, pages 101–107, 2013.

[23] D. P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *Journal of computer and system sciences*, 52(3):453–470, 1996.

[24] R. Duan and S. Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM*, 61(1):1:1–1:23, 2014.

[25] A. Dumitrescu and W. Steiger. On a matching problem in the plane. *Discrete Mathematics*, 211(1-3):183–195, 2000.

[26] C. Dürr, C. Konrad, and M. Renault. On the power of advice and randomization for online bipartite matching. In *Proc. of European Symposium on Algorithms*, 2016.

[27] J. Eckstein, P. L. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its application to data analysis. *Computational Optimization and Applications*, 23(3):285–298, 2002.

[28] U. Faigle and W. M. Nawijn. Note on scheduling intervals on-line. *Discrete Applied Mathematics*, 58(1):13–17, 1995.

[29] J. Feldman, A. Mehta, V. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *Proc. of FOCS*, pages 117–126, 2009.

[30] D. Freedman and L. Shepp. An unfriendly seating arrangement (problem 62-3). *SIAM Review*, 4(2):150, 1962.

[31] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. *Journal of Algorithms*, 23(1):180–194, 1997.

[32] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[33] H. González-Aguilar, D. Orden, P. Pérez-Lantero, D. Rappaport, C. Seara, J. Tejel, and J. Urrutia. Maximum rectilinear convex subsets. In *Proc. of International Symposium on Fundamentals of Computation Theory*, pages 274–291. Springer, 2019.

[34] B. Haeupler, V. S. Mirrokni, and M. Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In *International workshop on internet and network economics*, pages 170–181. Springer, 2011.

[35] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT Numerical Mathematics*, 32(2):249–267, 1992.

[36] J. Hershberger and S. Suri. Efficient breakout routing in printed circuit boards. In *Workshop on Algorithms and Data Structures*, pages 462–471, 1997.

[37] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. on Comput.*, 2(4):225–231, 1973.

[38] J. Håstad. Clique is hard to approximate within $n^{(1-\epsilon)}$. In *Acta Mathematica*, pages 627–636, 1996.

[39] P. Jaillet and X. Lu. Online stochastic matching: New algorithms with better bounds. *Mathematics of Operations Research*, 39(3):624–646, 2014.

[40] S. Kamali, P. Nikbakht, and A. Sajadpour. A randomized algorithm for non-crossing matching of online points. In *Proc. of CCCG*, pages 198–204, 2022.

[41] M. Kano and J. Urrutia. Discrete geometry on colored point sets in the plane—a survey. *Graphs and Combinatorics*, 37(1):1–53, 2021.

[42] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proc. of STOC*, pages 352–358, 1990.

[43] A. W. Kolen, J. K. Lenstra, C. H. Papadimitriou, and F. C. Spieksma. Interval scheduling: A survey. *Naval Research Logistics*, 54(5):530–543, 2007.

[44] M. Y. Kovalyov, C. T. Ng, and T. E. Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European journal of operational research*, 178(2):331–342, 2007.

[45] M. Kudo, Y. Torii, Y. Mori, and M. Shimbo. Approximation of class regions by quasi convex hulls. *Pattern Recognition Letters*, 19(9):777–786, 1998.

[46] R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. of SODA*, volume 94, pages 302–311, 1994.

[47] Y. Liu and M. Nediak. Planar case of the maximum box and related problems. In *Proc. of CCCG*, volume 3, pages 11–13, 2003.

[48] A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Proc. of FOCS*, pages 253–262, 2013.

[49] A. Madry. Computing maximum flow with augmenting electrical flows. In *Proc. of FOCS*, pages 593–602, 2016.

[50] V. H. Manshadi, S. O. Gharan, and A. Saberi. Online stochastic matching: Online actions based on offline statistics. *Mathematics of Operations Research*, 37(4):559–573, 2012.

[51] A. Mehta et al. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2013.

[52] S. Miyazaki. On the advice complexity of online bipartite matching and online stable marriage. *Information Processing Letters*, 114(12):714–717, 2014.

[53] A. Mohammad Lavasani and D. Pankratov. Advice complexity of online non-crossing matching. *Computational Geometry*, 110:101943, 2023.

[54] M. Mucha and P. Sankowski. Maximum matchings via gaussian elimination. In *Proc. of FOCS*, pages 248–255, 2004.

[55] OEIS Foundation Inc. The Catalan numbers, Entry A000108 in The On-Line Encyclopedia of Integer Sequences. https://oeis.org/A000108, 2021.

[56] OEIS Foundation Inc. Number of self-inverse permutations of $1, ..., n$, Entry A000085 in The On-Line Encyclopedia of Integer Sequences. https://oeis.org/A000085, 2024.

[57] A. Sajadpour. Non-crossing matching of online points. Master's thesis, University of Manitoba, 2021.

[58] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[59] G. M. Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.

[60] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, 2007.