Algorithmic Solutions for Interval and Rectangle Scheduling: Fairness, Prediction, and Beyond

Wenhao Zhu

A thesis submitted to the faculty of graduate studies in partial fulfillment of the requirements for the degree of master of science

graduate program in electrical engineering and computer science
York University
Toronto, Ontario

July 2025

Abstract

In the interval scheduling problem, the input is a set of intervals with integer endpoints, and the objective is to accept a maximum number of non-overlapping intervals. In the two-dimensional variant, namely rectangle scheduling, the input consists of a set of rectangles, and the goal is to select a maximum number of non-overlapping rectangles. One may consider these problems in both online and offline settings. In the offline setting, the input set is available at the beginning, and an algorithm makes a decision about selecting an interval (or rectangle) with complete information about the input. In the online setting, however, the input appears sequentially, and an online algorithm must accept an interval (or a rectangle) upon its arrival without any information about forthcoming intervals (or rectangles). The decisions of an algorithm for accepting/rejecting an item in its selected set are final and irrevocable.

Online interval scheduling has received considerable attention in the past few decades, and various algorithmic solutions have been proposed under different settings and models. In comparison, two-dimensional variants of the problem have been less studied. In this thesis, we study the online rectangle scheduling problems under the *any-order* arrival setting. Previous work on this topic has been limited to random-order arrival [40] (where rectangles are generated in parallel but their ordering is random) or under restrictions like square scheduling [2]. In comparison, we study the worst-case performance of online algorithms in the most generic setting of the problem and establish the tight upper and lower bounds of $\Theta(\log^2 T)$ on the competitive ratio of the best online algorithms. Here, T is a parameter that defines the maximum length of intervals.

Furthermore, we consider an online rectangle scheduling setting where (possibly erroneous) predictions are provided to the online algorithm. Here, predictions specify the presence or absence of rectangles in the input. Under this setting, we study an algorithm whose competitive ratio depends explicitly on the prediction error. In particular, our algorithm performs optimally when the predictions are perfect, and its performance degrades as the error increases. To make this algorithm *robust* against adversarial predictions (where error is large), we also propose a hybrid approach that combines it with a purely online algorithm and prove that this combined approach allows a trade-off between “consistency" and “robustness", which define the performance in scenarios where predictions are perfect and adversarial, respectively.

In addition, we initiate the study of the *fairness* aspects of the interval scheduling problem, where each interval belongs to a group, and the objective is to achieve a notion of fairness in which a “fair" number of intervals from each group are accepted. We study this problem under both *absolute* and *asymptotic* settings. In the asymptotic setting, the number of intervals in optimal solutions for each group is arbitrarily large, whereas this assumption is absent in the absolute setting. For each of these two

settings, we study the power of offline and online algorithms, as well as deterministic and randomized algorithms.

Contents

List of Figures

Chapter 1

Introduction

1.1 An Overview of Online Algorithms & Competitive Ratio

In theoretical computer science, many problems are studied under the assumption that all input data is available in advance. This approach aligns with the notion of *offline* algorithms, where the entire input is known before computation begins. For offline algorithms, research typically focuses on reducing *time complexity*, aiming for exact or approximate solutions within polynomial time. In contrast, *online* algorithms address scenarios where the input is revealed incrementally over time, and decisions must be made without knowledge of future data. Their performance is commonly evaluated via *competitive analysis*, which compares the performance of an online algorithm to that of an optimal offline algorithm, providing a measure of how well the online approach performs in the worst-case scenarios.

This thesis studies three problems. The first is *purely online rectangle scheduling* (Chapter 4), whose goal is to maximize the number of mutually non-overlapping rectangles when the rectangles arrive in an online manner. The second is *online rectangle scheduling with predictions* (Chapter 5), whose main objective is to maximize the number of mutually non-overlapping rectangles when a possibly erroneous prediction is provided. The third is *interval scheduling with fairness* (Chapter 6), whose main purpose is to be fair among all groups and also be competitive. These problems highlight two key challenges in online algorithm design: incorporating fairness and leveraging predictions. Both are introduced in detail in Sections 1.4 and 1.5.

Introduction to Competitive Analysis. Online algorithms are typically assessed via *competitive analysis*. Under competitive analysis, the performance of an online algorithm is compared to that of an optimal offline solution, which has full knowledge of the input and achieves the best possible outcome (e.g., the maximum number of non-overlapping intervals).

More precisely, the competitive ratio of an online algorithm A is the worst-case ratio between the benefit of the optimal offline solution OPT and that of A, over

all input sequences. Typically, the input sequence σ is assumed to be generated adversarially to realize the worst case. Sleator and Tarjan [68] introduced competitive analysis to evaluate the performance of their Move-to-Front and Least-Recently-Used algorithms for the classic list update and paging problems, respectively.

Formally, for a deterministic algorithm A for a maximization problem like interval scheduling, the performance of the algorithm is analyzed on input sequences σ generated by the adversary, and the competitive ratio is defined as

$$C.R(A) = \max_{\sigma} \frac{\textsc{Opt}(\sigma)}{A(\sigma)},$$

where $\textsc{Opt}(\sigma)$ denotes the offline optimal benefit, and $A(\sigma)$ the benefit of A on input σ.

For randomized algorithms, the competitive ratio is defined based on the expected benefit of the online algorithm A, formally as

$$C.R(A) = \max_{\sigma} \frac{\textsc{Opt}(\sigma)}{E[A(\sigma)]},$$

where $E[A(\sigma)]$ denotes the expected benefit of the online algorithm on input σ, and the expectation is taken over the internal randomness of the algorithm. When discussing randomized algorithms, we assume an *oblivious adversary* that knows how the algorithm works but is unaware of the random choices made by the algorithm. This type of adversary is the standard adversary in the realm of online algorithms. We refer to [16] for details about other types of (stronger) adversaries.

The above notions of competitive ratio are *absolute* in the sense that the ratio is considered for all input sequences. Sometimes we are interested in input sequences for which the benefit (or cost) of the optimal solution is asymptotically large. For that, we define the *asymptotic competitive ratio* of a deterministic online algorithm as

$$Asym.C.R(A) = \max_{\sigma \text{ s.t. } \textsc{Opt}(\sigma)\to\infty} \frac{\textsc{Opt}(\sigma)}{A(\sigma)}.$$

The asymptotic competitive ratio for randomized algorithms is defined analogously by replacing $A(\sigma)$ with $E[A(\sigma)]$. In this thesis, unless otherwise mentioned, by "competitive ratio", we mean "absolute competitive ratio".

Intuitively, the competitive ratio for online algorithms is defined similarly to the approximation ratio for offline problems. While the competitive ratio measures how the *online constraint* (lack of knowledge about the future) impacts the performance of algorithmic solutions, the approximation ratio measures how the lack of computational power (or other constraints faced by an offline algorithm) impacts performance.

1.2 The Interval Scheduling Problem

The interval scheduling problem is a fundamental optimization problem with applications in job scheduling [37, 49], resource allocation [62, 54], and related fields [46, 55]. The input is a set of intervals (in the offline setting) or a sequence of intervals (in the online setting), where each interval is defined by its start and end times and potentially other attributes such as a positive weight. We assume start and end times are integer values in $[1, T]$. In the online setting, we assume T is known to the algorithms. The goal is to select a maximum number of non-overlapping intervals (or, in the weighted setting, a set of non-overlapping intervals to maximize the total weight).

In the offline setting, all intervals are provided in advance, allowing for the efficient computation of optimal solutions. In the unweighted case, a simple greedy algorithm suffices [48], while the weighted case can be solved by dynamic programming [48].

In the online setting, intervals arrive one at a time, and decisions must be made immediately and irrevocably. Variants include the *timeline setting*, where intervals appear in non-decreasing order of their starting times [37, 57]; the *any-order setting* [17], where intervals may arrive in an arbitrary (in particular, adversarial) order; and the *random-order setting*, where intervals appear in random order [18, 40]. In all cases, the input is assumed to be adversarially generated.

In this thesis, we focus on the *unweighted* version, where all intervals have weight 1, and, unless otherwise stated, "interval scheduling" refers to the unweighted setting.

1.3 The Rectangle Scheduling Problem

The RECTANGLE SCHEDULING PROBLEM generalizes the classical interval scheduling problem to two dimensions, where the input consists of a sequence of axis-aligned rectangles, with integer side lengths and corners on a $T \times T$ grid, where T is a large number that is known to the algorithm. The goal is to select the largest subset of non-overlapping rectangles. While interval scheduling admits polynomial-time solutions in the offline setting, the offline RECTANGLE SCHEDULING PROBLEM is NP-hard and fundamentally more challenging in the offline setting [23].

Garg, Kar, and Khan [40] have recently studied the RECTANGLE SCHEDULING PROBLEM under the *random-order* model setting, where input rectangles are adversarially generated but randomly permuted. Note that, unlike interval scheduling, the timeline setting is not well-defined for rectangle scheduling. It is possible, however, to consider a fully-adversarial *any-order* model (similar to interval scheduling [51, 17]), where both rectangles and their ordering are defined adversarially. Advani and Asudeh [2] have recently studied the online d-cube scheduling problem under the any-order setting; in particular, they established a tight competitive ratio of $\Theta(\log T)$ for the best possible competitive ratio.

In Chapter 4, we study the RECTANGLE SCHEDULING PROBLEM in the online setting under any-order setting. We show that deterministic algorithms are ineffective and cannot achieve a competitive ratio better than $\Omega(T^2)$. Therefore, we focus on randomized algorithms and present both algorithmic and impossibility results to establish a tight competitive ratio of $\Theta(\log^2 T)$ for the RECTANGLE SCHEDULING PROBLEM. Our results indicate that scheduling rectangles is provably harder than scheduling squares.

1.4 Prediction and Advice

With the advent of modern machine learning, there is growing interest in augmenting online algorithms with *predictions* [9, 3, 66, 52, 20]. Since predictions may be inaccurate, algorithms must balance performance under perfect predictions (*consistency*) with worst-case guarantees (*robustness*). There is often a tension between consistency and robustness [43, 20], and one frequently aims to design algorithms that provide meaningful (preferably Pareto-optimal) trade-offs between the two.

In addition, many works define a notion of *error* (typically a distance measure between what is predicted and what is actually observed in the input) and express the competitive ratio as a function of this error. The consistency of an algorithm is then its competitive ratio when the error is zero, and its robustness is the competitive ratio when the error is maximal. Bounding the competitive ratio as a function of error also helps study the algorithm's smoothness, which captures how gracefully the performance degrades as the error increases [10].

A more theoretical concept related to predictions is the *advice model*. Under this model, an online algorithm receives partial but perfect information about the input sequence, generated by an offline oracle with unbounded computational power. The challenge here is to determine how much advice (measured in bits) is needed to achieve a given competitive ratio. Unlike predictions, advice is perfect but size-limited.

Interval scheduling has previously been studied by Boyar et al. [20], under a setting where it is predicted which intervals will appear in the input, and the error is modeled by the cardinality of the optimal solution of a set of intervals that are incorrectly predicted to appear (false positives) or incorrectly predicted to not appear (false negatives) in the input. In Chapter 5, we extend this prediction model to online rectangle scheduling. We study an algorithm, 2DTRUST, which has a competitive ratio of $\frac{1}{1-2\gamma}$ and show that no online algorithm can have a competitive ratio better than $\frac{1}{1-\gamma}$. We also introduce a hybrid family of algorithms that provides a trade-off between robustness and consistency. That is, one can use these algorithms to achieve better robustness at the cost of worse consistency and vice versa.

1.5 An Overview of Algorithmic Fairness

Algorithmic fairness has become a key concern in real-world algorithm design, with various fairness notions proposed in the context of online algorithms. For example, *time fairness* requires treating items equally regardless of their arrival time (see Section 2.6.1). More relevant to this thesis is *group fairness* (see Section 2.6.2), where items belong to different groups, and the goal is to ensure a balanced number of items from each group are selected.

In Chapter 6, we study fairness aspects of interval scheduling. The assumption is that each interval belongs to a group $g \in [1, k]$, where k is a known number of groups and is typically assumed to be a constant compared to input length n, or the value T (interval endpoints are integers in $[1, T]$). We define the *fairness ratio* of an algorithm as the minimum ratio between the number of accepted intervals from each group and the number of intervals in an optimal schedule of that group, where the maximum is taken over all groups. We prove that the fairness ratio of any algorithm is in the range $[0, 1/k]$ and sometimes refer to an algorithm as simply "fair" if it achieves a fairness ratio of $1/k$.

Fairness introduces challenges not only to online, but also to offline interval scheduling, and thus, we examine both offline and online settings of the problem when discussing fairness. Obviously, better fairness guarantees can be provided in the offline setting.

We also distinguish between *asymptotic* and *absolute* settings of the problem. In the asymptotic setting, the optimal solution for each group's intervals has an unbounded number of intervals within it. As we will see, the asymptotic setting gives flexibility in designing algorithms that achieve better fairness guarantees.

Chapter 2

Preliminaries & Related Work

In this chapter, we present an overview of the related work on the topics of interval scheduling, rectangle scheduling, prediction, and algorithmic fairness.

2.1 Randomized Online Algorithms & Yao's Minimax Principle

Online algorithms face scenarios where inputs are revealed sequentially, and decisions must be made immediately upon the arrival of each input. While in certain models, decisions of an online algorithm may be revoked [17, 31], in the standard setting, once a decision is made (e.g., an interval is accepted), it is final and cannot be altered later.

Randomization is often used to improve the competitive ratio of online algorithms. While there are certain online problems, e.g., online bin packing problem (see, e.g., [26]), for which randomization does not help in improving competitive ratios, for most online problems, for example, the ski rental problem [44] and the list update problem [11], randomization helps. This is because deterministic algorithms are too simple and predictable, making them vulnerable in an adversarial setting where the input sequence is designed with full knowledge of the decisions made by the deterministic algorithms. As a result, a deterministic algorithm often offers a poor performance guarantee, characterized by a bad competitive ratio in the worst case.

To overcome the limitation of deterministic algorithms, randomization is employed in the decision-making process. Due to the unpredictability of its decision, which prevents an adversary from predicting its behavior in advance. This uncertainty allows randomized algorithms to achieve a better competitive ratio against the oblivious adversaries.

In the context of proving impossibility results for randomized algorithms, *Yao's Minimax Principle* provides a powerful tool for analyzing the lower bounds on the competitive ratio of a randomized algorithm. It asserts that the expected cost of a randomized algorithm is at least equal to the expected cost of a deterministic

algorithm against a distribution of random input sequences. Formally, it can be expressed as follows:

Definition 1 (Yao's Minimax Principle [16]). *Let $\mathcal{A}$ be a finite set of deterministic algorithms and $\mathcal{I}$ be a finite set of sequences. Let $c(a, i)$ be the cost of the algorithm $a \in \mathcal{A}$ on the input sequence $i \in \mathcal{I}$. Then,*

$$\max_{i \in \mathcal{I}} E[c(A, i)] \geq \min_{a \in \mathcal{A}} E[c(a, I)],$$

where A is a randomized algorithm (a distribution over $\mathcal{A}$) and I is a randomized input sequence (a distribution over $\mathcal{I}$)

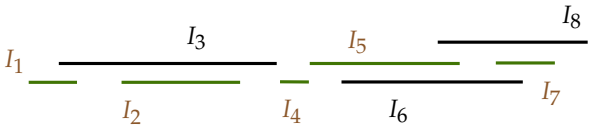
Intuitively, when proving lower bounds for the competitive ratio of randomized algorithms, we use Yao's Minimax Principle in the following way. Instead of considering an arbitrary randomized algorithm against worst-case input, we consider a deterministic algorithm faced with a stochastic input. Using Yao's Minimax Principle, a negative result for the competitive ratio of such a deterministic algorithm extends to the online algorithm.

2.2 Offline Interval Scheduling

In the offline interval scheduling problem, the input is a set of unweighted or weighted intervals, and the primary goal is to select a set of intervals with maximum cardinality or total weight, subject to the restriction that no two selected intervals intersect. In this chapter, we briefly discuss algorithms and impossibility results designed for these two settings.

Unweighted Interval Scheduling. Kleinberg and Tardos [48] introduced the offline greedy algorithm for the offline interval scheduling problem and proved that it produces an optimal schedule. The offline greedy algorithm begins by sorting the intervals in increasing order of their ending times. It then processes the input in that order and iteratively selects the next available interval that does not overlap with any of the previously accepted intervals. Note that this algorithm is inherently offline, as it requires sorting intervals before selecting any of them.

For example, consider the input below, where we have an input $I = (I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8)$. The offline greedy algorithm selected I_1, I_2, I_4, I_5, and I_7.



Weighted Interval Scheduling. In the weighted interval scheduling problem, each interval has an associated weight, so we cannot simply apply a greedy algorithm to solve the problem optimally. Kleinberg and Tardos [48] provided an optimal solution using a dynamic programming approach. First, sort all intervals by their finish times. Then, for each interval I_i, define $p(I_i)$ as the index of the last interval that finishes before I_i begins (i.e., the latest non-overlapping interval with I_i). Let $\textsc{Opt}(I_i)$ denote the optimal total weight of the subset $\{I_1, \ldots, I_i\}$. Using these definitions, the algorithm constructs the optimal solution iteratively from I_1 to I_n using recursion $\textsc{Opt}[i] = \max\big(w_i + \textsc{Opt}[p(i)], \textsc{Opt}[i-1]\big)$.

Example 2.2.1. *Consider the input described by the following table.*

i	1	2	3	4	5	6
s_i	1	2	3	4	5	6
f_i	3	6	8	9	10	12
w_i	5	6	5	4	11	2

For each interval, we compute the corresponding value of $p(i)$, as shown in the table below.

i	1	2	3	4	5	6
$p(i)$	0	0	1	1	1	2

Applying the dynamic programming recurrence to compute the maximum achievable weight at each step yields the resulting values in the table below.

j	w_i	$p(i)$	$w_i + \textsc{Opt}[p(i)]$	$\textsc{Opt}[i] = \max(w_i + \textsc{Opt}[p(i)], \textsc{Opt}[i-1])$
0	—	—	—	0
1	5	0	$5+0=5$	$\max(5,0)=5$
2	6	0	$6+0=6$	$\max(6,5)=6$
3	5	1	$5+5=10$	$\max(10,6)=10$
4	4	1	$4+5=9$	$\max(9,10)=10$
5	11	1	$11+5=16$	$\max(16,10)=16$
6	2	2	$2+6=8$	$\max(16,8)=16$

2.3 Online Interval Scheduling

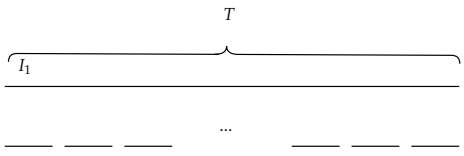
In the online interval scheduling problem, intervals arrive in an online manner, meaning decisions must be made without knowledge of future intervals. This makes

the problem more challenging, as we lack complete information to determine whether to accept or reject each incoming interval.

There are various settings for this problem, such as **preemption** [29, 57, 37, 36] (where a new interval can replace a previously accepted one), **restart** [25, 56] (where rejected intervals can be restarted later). Moreover, concerning input arrival, we have **timeline setting** (intervals appear sorted by their left endpoint) [37, 56], **any order** [17, 16, 20] (intervals arrive in an arbitrary order), and **random order** [40, 18] (intervals are generated by an adversary but arrive in a random order). In this thesis, we primarily focus on the any-order setting of the problem.

An Impossibility Result for Deterministic Algorithms.

In the adversarial setting of the online interval scheduling problem, the adversary can arrange a sequence of intervals such that the deterministic algorithm is not competitive, as discussed in [17]. Consider the figure below, assuming that the input starts with the interval $I_1 = [1, T]$.



The deterministic algorithm has to make an immediate decision to accept or reject I_1. If the algorithm decides to accept I_1, the adversary will send T smaller intervals that overlap with the longer interval I_1. Consequently, the optimal solution (OPT) can accept T intervals, whereas the algorithm can only accept one interval. If the algorithm decides to reject I_1, the adversary stops generating any additional intervals. OPT will accept the interval I_1, but the algorithm will accept no intervals. In both cases, the competitive ratio of the deterministic algorithm is poor. Therefore, other techniques, such as randomized algorithms, must be applied to improve performance.

2.3.1 CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT Algorithm

Consider an online Greedy algorithm for interval scheduling that, upon the arrival of an interval, selects it if and only if the interval does not intersect with any previously accepted intervals. This algorithm, which is deterministic, exhibits poor performance, as discussed previously. However, it can be used as a component in certain randomized algorithms, as we will discuss. Note that if all intervals in an input have a length in $[L, 2L)$, the greedy algorithm has a competitive ratio of at most 3; this is because, for every accepted interval by the greedy algorithm, at most three intervals can be accepted by OPT. This property makes greedy ideal for algorithms that classify intervals by their lengths and apply this greedy approach for intervals of the same class.

The CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT Algorithm (CLRS). Awerbuch et al. [6] introduced an online algorithm for the interval scheduling problem. We refer to this algorithm as CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT (CLRS) and extensively use it in this thesis. Suppose the value of T is a power of 2; recall that T is known to the algorithm. Each interval i from the input sequence σ has a length $\ell_i \in [1, T]$. Therefore, we can classify all intervals into $\log T$ classes, where intervals in class j have lengths in the range $[2^j, 2^{j+1})$. Hence, there are at most $\log T$ such classes. CLRS selects a class $i \in [0, \log T)$ uniformly at random and then applies the online greedy algorithm restricted to intervals within the selected class. As previously discussed, the online greedy algorithm achieves a competitive ratio of 3 when restricted to a single class of intervals. Since the class is selected uniformly at random, it is easy to establish that the competitive ratio of CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT is at most $3 \log T$ [6].

Borodin and El-Yaniv [16] present a similar algorithm, named CLASSIFY-AND-RANDOMLY-SELECT (CRS), which works by classifying intervals into classes, randomly selecting a class, and applying the greedy algorithm for members of the selected class. The only difference is that their algorithm classifies intervals based on their arrival *positions* rather than length.

CLRS and CRS both have a competitive ratio of $O(\log T)$. Indeed, it is proven that no online randomized (or deterministic) algorithm can have a competitive ratio $\Omega(\log T)$ [16]. Therefore, both CLRS and CRS are optimal with respect to competitive ratio.

2.3.2 Weighted Online Interval Scheduling

In the weighted setting, the problem becomes significantly more challenging, and thus, more relaxed settings of the problem are studied. In particular, the problem has been studied in the *timeline setting* with *preemption*. Preemption allows accepting an interval and rejecting it later (but a rejected interval can never be a part of the schedule). Woeginger [72] proposed a deterministic algorithm that uses preemption and achieves a competitive ratio of 4. Under the more restricted *non-decreasing monotone* setting, where newly arrived intervals have higher weights, Miyazawa and Erlebach [59] proved a lower bound of $5/4$ and presented a 3-competitive randomized algorithm. Fung, Poon, and Zheng [37] proposed a barely random algorithm, which achieves a competitive ratio of 2 for the general setting. These results pertain to the timeline setting, and the problem has not been explored under either *any-order* settings. The very special case, where all intervals intersect (e.g., all are [1,2]) and the input appears in random order, becomes the famous *secretary problem* [32]. Another variant, where each interval is assigned a weight proportional to its length, is studied by Lipton and Tomkins [57] under an *unbounded setting*, where the value of T is unknown. This setting provides upper and lower bounds for the competitive ratio of the best possible algorithms. In particular, under timeline setting, they introduced

an algorithm which achieves a competitive ratio of $O((\log \Delta)^{1+\epsilon})$ for the online weighted interval scheduling problem, assuming all interval lengths are in $[1, \Delta]$, even when Δ is unknown in advance.

2.4 Rectangle and Hyper-rectangle Scheduling

The Hyper-rectangle Scheduling problem generalizes interval scheduling to two or more dimensions, where the input consists of a sequence of axis-aligned hyper-rectangles. The objective is to select a maximum number of non-overlapping rectangles. In the online setting, hyper-rectangles appear sequentially and in an online manner. The Hyper-rectangle Scheduling has received considerably less attention compared to the interval scheduling problem, particularly in the online setting.

Unlike interval scheduling, which admits optimal polynomial-time algorithms in the offline setting, the offline Hyper-rectangle Scheduling problem is NP-hard even in 2 dimensions [23], and much of the research has focused on approximation algorithms [58, 38]. Early work for offline rectangle scheduling [23, 47] proposed approximation algorithms for the problem, but they did not achieve constant-factor guarantees. A breakthrough was made by Mitchell [58], who was the first to design a constant-factor approximation algorithm, achieving an approximation factor of 10. This result was later improved by Gálvez et al. [38], who developed a 3-approximation algorithm.

There have been a few recent works on online hyper-rectangle scheduling problems. Garg, Kar, and Khan [40] studied the hyper-rectangle scheduling problem in d dimensions under the *random-order* model. Their algorithm first classifies all hyper-rectangles into distinct classes and then "observes" the first half of the input sequence to identify the best class. Finally, it applies an online greedy algorithm specifically designed for this class. This approach achieves a competitive ratio of approximately $O((\log n)^d \cdot \log \log n)$.

Advani and Asudeh [2] study the hypercube scheduling problem in d dimensions under various settings. In the *dominating order* setting, where the top-right corner of each newly arrived rectangle is coordinate-wise no smaller than that of any previously arrived rectangle, they show that the online deterministic greedy algorithm results in an optimal schedule. For general settings of the problem, however, deterministic algorithms are not effective, and randomized algorithms are presented. For scheduling unit-length hypercubes, they established a lower bound of $\frac{32}{15}$ and upper bound of 2^d on the competitive ratio of any randomized algorithm. For scheduling T-bounded hypercubes of arbitrary integer length, they prove a lower bound of $\frac{\lceil \log T \rceil}{2}$ and then presented an algorithm with a competitive ratio of $3^d \log T$. Therefore, $\Theta(\log T)$ is the best attainable competitive ratio for square scheduling.

2.5 Online Algorithms with Prediction

Recent developments in *learning-augmented online algorithms* aim to bridge the gap between worst-case analysis and data-driven performance. In this framework, algorithms leverage predictions to improve either the time complexity of offline algorithms or the competitive ratio of online algorithms, with applications to classical problems such as ski rental [50, 13, 71] and secretary [35, 4]. This thesis primarily focuses on how predictions can improve the competitive ratio of online algorithms.

Definition 2. *An online algorithm may receive predictions either at the start of its execution or upon the arrival of each item a_i for $i \in [1, n]$. For each item a_i, there may exist a prediction of a_i, denoted by $\hat{a}_i$. At time t, the algorithm makes a decision upon receiving item a_i, where the decision depends on the input sequence up to a_i and a function of $\hat{a}_i$.*

Since predictions may not always be accurate, two key criteria are considered: *consistency* and *robustness*. *Consistency* measures the algorithm's performance when predictions are accurate, while *robustness* captures its worst-case performance when predictions are highly erroneous.

Definition 3. *Let A be a learning-augmented online algorithm. We say that:*

- *A is α-*consistent *if it achieves a competitive ratio of at most α when the predictions are perfectly accurate.*
- *A is β-*robust *if it guarantees a competitive ratio of at most β even when the predictions are arbitrarily inaccurate.*

Normally, there exists a trade-off between consistency and robustness, which reflects the extent to which we are willing to trust the predictions. An algorithm that relies more heavily on predictions typically achieves higher consistency but may perform poorly when the predictions are inaccurate.

Example 2.5.1. *Consider the classical* ski rental problem. *In this problem, a skier must decide on each day whether to continue renting equipment at a unit cost or to purchase it outright for a fixed cost b. Suppose the skier is given a prediction that the total number of skiing days y will exceed b. If the skier fully trusts this prediction, they may choose to purchase the equipment on day 1. However, if the prediction turns out to be inaccurate and the skiing ends on the next day, this decision leads to poor robustness, as the skier would incur the full cost b while using the equipment for only one day.*

Prediction Model. When incorporating predictions into algorithms, it is important to consider what type of information will most effectively enhance the algorithm's performance. For example, in the classical ski rental problem, the predicted duration of the snow season can be provided at the outset [50, 67, 13]. In the secretary problem, predictions may focus on the value of the best candidate [35].

In practice, prediction models may also leverage techniques based on the *Probably Approximately Correct (PAC) learning framework* [70], which aim to generate predictions based on a limited number of samples. Specifically, a PAC-learnable algorithm outputs a hypothesis that, with high probability at least $1 - \delta$, incurs an error of at most ϵ, provided access to a small, representative sample drawn from the input distribution.

Online Interval Scheduling Problem with Prediction. Previous works have addressed the online interval scheduling problem with the assistance of predictions. Boyar et al. [20] proposed algorithms that leverage (potentially erroneous) predictions of incoming intervals. In other words, for any possible interval, the prediction specifies whether it is in the input or not. Note that this is a rather strong prediction. Their main algorithm involves constructing an optimal offline schedule, a *plan*, based on a predicted set of intervals using the offline greedy algorithm (which greedily selects non-overlapping intervals in the order of their finishing time). When an interval from the actual input sequence arrives, the algorithm proceeds as follows:

- If the interval is part of the algorithm's plan, it is accepted.
- Else, if the interval is not in the plan but does not overlap with any interval currently in the plan, it is accepted and added to the plan.
- Else, if the interval overlaps with exactly one interval in the plan, ends earlier than that interval, and replacing it does not reduce the total number of intervals in the plan, then the plan is updated to include the new interval.
- Else, the interval is rejected.

This algorithm performs well when the prediction is accurate. However, its performance may degrade significantly when predictions are inaccurate.

To address this issue, Boyar et al. [20] propose a hybrid strategy that applies the CLASSIFY-AND-RANDOMLY-SELECT (CRS) algorithm with probability α and their prediction-based algorithm with probability $1 - \alpha$. This probabilistic approach ensures that the algorithm remains both consistent and robust under varying levels of prediction quality.

Recently, Karavasilis [43] studied the problem under a weaker prediction model. In their setting, when an interval arrives, the algorithm queries the predictor to determine whether the interval belongs to the optimal solution. This binary guidance helps the algorithm make more informed decisions without requiring full knowledge of the future or exact optimal schedule.

2.5.1 Online Algorithm with Advice

In this section, we explore the *power of advice*, which allows an online algorithm to access partial information about the future input sequence before it is revealed. The advice model, introduced in [19], can be viewed as providing the online algorithm with a limited amount of perfect information about the input. This model is particularly useful for establishing impossibility results for learning-augmented online algorithms when the prediction is of relatively small length (i.e., can be encoded in a small number of bits).

Definition 4. *An* online algorithm with advice *has access to an* advice tape *of infinite length. Before the execution of the algorithm starts, it can read any number of advice bits from the advice tape. The number of such bits defines the* advice complexity *of the algorithm. The advice bits are written on the tape by a benevolent offline oracle which has unbounded computational power and has access to the entire input sequence. The advice bits are assumed to be correct in the sense that they encode exactly what they are supposed to convey. Both the algorithm and the oracle know the meaning of advice bits.*

Generally speaking, the more advice bits an algorithm has, the better the performance of the algorithm (in terms of competitive ratio).

Reductions from the *string guessing problem* are a standard approach to establishing the lower bound of the advice complexity [30, 15]. In the *binary guessing problem*, the input consists of a sequence of binary bits, and for each bit, the algorithm must guess whether it is a 0 or 1 before the bit is revealed. The algorithm will receive a benefit of 1 if it guesses one input correctly. If no advice is provided, an adversary can ensure that a deterministic algorithm makes a mistake at every guess (it simply reveals the opposite bit). With the help of a single bit of advice, however, the algorithm can be informed whether the number of 0-bits exceeds the number of 1-bits. It then always guesses the majority bit, which inherently ensures a benefit of at least half of OPT. However, one cannot achieve a better competitive ratio (i.e., correctly guess a fraction $1/2 + \epsilon$ of all bits) with a sublinear amount of advice [14]. Reductions from Binary Guessing to various online problems are used to prove the impossibility results for advice (and thus prediction) settings. We will present one such result in Section 5.6.

2.6 Fairness Notions in Algorithm Design

In recent years, there has been an increasing interest in designing "fair" algorithmic solutions that not only aim to achieve good performance (e.g., competitiveness) but also guarantee a notion of fairness, which is often problem-specific.

2.6.1 Time Fairness

Consider an online problem, like the proportional knapsack problem, where the input is formed by a sequence of items, where each item i has a size $s_i \leq \epsilon$, for some small ϵ, and a value v_i. An algorithm should accept or reject each item upon its arrival, and the goal is to accept a set of items with a total size of at most 1 while maximizing the total value of the accepted items. A natural algorithm is a *threshold algorithm* [74], which maintains a threshold that increases as the knapsack becomes fuller and accepts an item if and only if its value/size ratio is no less than the threshold. This algorithm, however, is not *fair* in the sense that two equivalent items of the same size and weight might be treated differently (the item that appears first has a higher chance of inclusion in the knapsack, as the threshold is smaller earlier in the input). This notion of fairness, which requires similar items in the input sequence to be treated similarly regardless of their position in the input, is called *time fairness*. Lechowicz et al. [53] studied time fairness for knapsack, where it is shown to achieve a meaningful notion of fairness, predictions are necessary and sufficient. Other problems studied under time fairness requirements include prophet inequalities [5] and the secretary problem [22].

2.6.2 Group Fairness

A more relevant notion of fairness to the topics of this thesis is *group fairness*. This type of fairness concerns algorithmic solutions that are *fair* across groups that generate or label the input sequence. For example, consider a bipartite matching problem, where candidates need to be assigned to jobs. Suppose jobs are fixed and candidates appear online; this is the standard notion of online bipartite matching [45]. Now, under the group fairness setting, we may assume that the candidates belong to various groups, and we need a fair solution where a balanced number of candidates from each group are matched. This problem is studied in [42]. Other problems that are studied under the group fairness constraints include clustering [12, 1, 24], ranking [39, 73, 41], and knapsack problem [63, 33].

Proportional Fairness. A notion of group fairness in online problems is *proportional fairness* in non-centralized, multi-agent settings, which requires that each agent receive an equitable share of the overall benefit in resource-sharing environments. Consider the classic *cake-cutting problem*, where the goal is to allocate an entire cake among k agents. Proportional fairness aims to guarantee that each agent receives a fair portion of the cake, ideally of equal value. For example, if two agents should share a cake in which a quarter is chocolate and the rest is strawberry, the proportionally fair way to divide the cake is to give each agent half of the chocolate and half of the strawberry.

Envy-Free Fairness. Envy-free fairness [64, 27, 42, 8] is relevant in scenarios where agents put a value on their received allocation of the resource. A solution is said to be

envy-free if it ensures that no agent prefers another agent's allocation over their own. In the context of online algorithms, this typically means that each agent values their own allocation at least as much as they value any other agent's allocation.

In our cake-cutting example, suppose agent A prefers the strawberry portion (which is larger) while agent B prefers the chocolate portion (despite being smaller). To satisfy envy-free fairness, an algorithm would allocate the strawberry part to agent A and the chocolate part to agent B. Even if agent A receives a larger portion, agent B remains satisfied with the outcome because they do not prefer agent A's allocation over their own.

For two agents, a well-known procedure called *Divide and Choose* [69] can be used to achieve envy-freeness: one agent divides the cake into two pieces, and the other chooses the piece they prefer. However, the problem becomes more complex with more than two agents [21].

2.6.3 Max-Min, Ex-Ante, and Ex-Post Fairness Notions

Max–Min Fairness [65, 61] is another widely used notion of fairness. Its objective is to *maximize the minimum* benefit obtained by any subgroup. In other words, the algorithm aiming for Max-Min Fairness seeks to maximize the benefit of the least advantaged group, ensuring its benefit is as high as possible.

Formally, let n be the number of agents and let $A = (A_1, A_2, \ldots, A_n)$ denote an allocation, where A_i represents the benefit assigned to agent i. The objective in Max–Min Fair allocation is to maximize the smallest value received by any agent and therefore guarantees every group to receive a lower bound on their benefit

For deterministic algorithms, ensuring fairness is often challenging. As a result, randomization is commonly employed to achieve fairness across groups. However, randomized algorithms typically only guarantee *ex-ante fairness* [34], which ensures that the *expected* benefit is distributed fairly among groups. In practice, however, ex-ante fairness does not guarantee that each group will receive a fair share in the realized outcome. For instance, consider the simple case where a single item must be assigned to one of three individuals, each having an equal probability of receiving it. Although the expected benefit is identical for all three, only one individual ultimately receives the item. This outcome is fair in expectation but may appear unfair in the realized allocation. To address this, *ex-post fairness* is introduced as a stronger notion of fairness that aims to ensure that each group receives at least a proportional share of the *actual* outcome, rather than potentially nothing [34].

Chapter 3

Problem Definitions and Initial Observations

3.1 Overview

In this chapter, we first formally define the interval and rectangle scheduling problems. Additionally, we introduce the fairness measures for interval scheduling that will be used throughout this thesis.

3.2 Problem Definitions

We provide a formal definition of *interval scheduling*, *rectangle scheduling*, and *online rectangle scheduling with prediction* problems.

Interval Scheduling: Problem Definition

Definition 5 (Interval Scheduling)**.** *The input to an instance of the* interval scheduling problem *is a set of n intervals* $\{I_1, I_2, \ldots, I_n\}$*, where* $I_i = [s_i, f_i)$ *is the i'th interval, and its endpoints* s_i, f_i *are non-negative integers in* $[1, T]$ *and* $s_i < f_i$*. The objective is to select a set of intervals of maximum cardinality such that no two selected intervals have a common point.*

- *In the offline setting, the entire set of n intervals* $\{I_1, I_2, \ldots, I_n\}$ *is known in advance. The algorithm has full knowledge of all intervals before making any selection decisions.*
- *In the online setting, the n intervals appear sequentially as a sequence* $\sigma = \langle I_1, I_2, \ldots, I_n \rangle$*. Upon arrival of each interval* I_i*, an irrevocable decision must be made*

to select (accept) or reject I_i under the same restriction that no two accepted intervals can share a point. We consider the any-order setting, *where the intervals appear in an arbitrary order defined by an adversary.*

Rectangle Scheduling: Problem Definition

Definition 6 (Rectangle Scheduling)**.** *The input to an instance of the* rectangle scheduling problem *is a set of n rectangles $\{R_1, R_2, \ldots, R_n\}$, where $R_i = (x_i, y_i, w_i, h_i)$ is the i'th rectangle in 2D, where x_i, y_i are integers in $[0, T]$ and denote the bottom-left corner of the rectangle, and w_i, h_i are integers that denote the width and height of R_i, respectively. We have $x_i + w_i \leq T + 1$ and $y_i + h_i \leq T + 1$. That is, all corners of all rectangles are located in a $[0, T] \times [0, T]$ grid. The objective is to select a set of rectangles of maximum cardinality such that no two selected rectangles intersect.*

- *In the offline setting, the entire set of n rectangles $\{R_1, R_2, \ldots, R_n\}$ is known in advance. The algorithm has full knowledge of all rectangles before making any selection decisions.*
- *In the online setting, the n rectangles appear sequentially as a sequence $\sigma = \langle R_1, R_2, \ldots, R_n \rangle$ and, upon arrival of each R_i, an irrevocable decision must be made to accept or reject R_i under a restriction that no two accepted rectangles must intersect. As before, the objective is to maximize the number of accepted rectangles. We consider the any-order setting, where rectangles appear in an arbitrary order determined by an adversary.*

Rectangle Scheduling with Prediction: Problem Definition

Definition 7 (Online Rectangle Scheduling with Predictions)**.** *The input to an instance of the* Online Rectangle Scheduling with Predictions *problem is a sequence R of n rectangles, denoted as $\sigma = \langle R_1, R_2, \ldots, R_n \rangle$, where each rectangle R_i finds its four corners on the $T \times T$ grid. Additionally, the algorithm is provided with a prediction set $\hat{R}$, which is a bitstring of length $O(T^4)$ and specifies, for each possible rectangle R with four corners on the $T \times T$ grid, whether R appears in the input sequence σ or not. The set of predicted rectangles $\hat{R}$ is given to the online algorithm beforehand. As before, rectangles in σ can arrive in any order, and, upon*

the arrival of the i'th rectangle $R_i \in \sigma$, an irrevocable decision needs to be made immediately to accept or reject it under the restriction that no two rectangles intersect. The objective is to leverage the predicted rectangles $\hat{R}$ to improve performance and to select a set of rectangles such that no two rectangles intersect.

3.3 Fairness Measures for Interval Scheduling

One of the main objectives in this thesis is to investigate "fair" solutions for the interval scheduling problem. We particularly study group fairness, where each interval belongs to one group, and the objective is to achieve solutions that somehow balance the number of intervals accepted from each group. Even though the Greedy algorithm achieves optimality in the offline setting, it does not address fairness concerns. In fact, ensuring fairness among groups is a fundamental challenge in both offline and online interval scheduling.

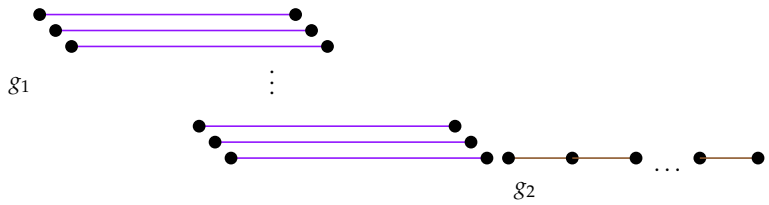
One difficulty in studying the problem under fairness considerations is "how to model fairness?". In what follows, we first explore the limitations of some standard fairness notions and introduce the fairness notion that is used in this thesis.

3.3.1 Potential Fairness Notions for Interval Scheduling

Several fairness notions are commonly studied in this thesis, such as *envy-free fairness* [7, 21, 64], *proportional fairness* [28], and *max-min fairness* [60], as discussed in Section 2.6. Before introducing our fairness measure tailored to this setting, we briefly review some standard fairness concepts and highlight their limitations:

- **Equal Allocation Fairness:** The simplest measure is perhaps an *equality type of fairness*, where the goal is to accept roughly the same number of intervals from each group. This measure is, unfortunately, not satisfactory and does not yield to competitive solutions. Consider an input formed by a single long interval $[1, T]$ that belongs to group g_1 and T intervals of the form $[i, i+1)$ that belong to group g_2. An equality fairness measure requires accepting the same number of intervals from each group. The following approach can achieve such fairness guarantees: flip a coin to accept the single interval from g_1 or exactly one interval from g_2 (while rejecting all other intervals from g_2). On expectation, we accept 0.5 interval from each group. This approach is not, however, competitive: the competitive ratio would be $\frac{T}{1} = T$, which is not good. Moreover, fairness is achieved by artificially rejecting items from group 2, which is undesirable.

- **Proportional Fairness:** Another approach to *proportional fairness*, where the goal is to accept roughly the same percentage of intervals from each class. This measure is also, unfortunately, not satisfactory and does not yield competitive solutions. Consider an input formed by two groups of intervals. Group g_1 contains $T/4$ intervals of the form $[i, i + T/4)$, where $i \in [0, T/4)$ is an integer. Group g_2 contains $T/4$ intervals of the form $[j + T/2, j + 1 + T/2)$, where $j \in [0, T/4)$ is an integer. Given that there are $T/4$ intervals from each class, a proportional fairness measure requires accepting the same number of intervals from each group. On the other hand, no two intervals of group g_1 can be accepted (they all overlap), and thus, the proportional fairness can be achieved by rejecting all intervals of group g_2 except for one. In conclusion, under this fairness measure, no algorithm can be competitive (the competitive ratio would be at least $\frac{T/4+1}{2}$), and the fairness is again achieved only by artificially rejecting items of group g_2 that could be accepted.



3.3.2 Fairness Ratio

We present the *fairness ratio* as our main criterion for evaluating fairness of interval scheduling algorithms.

Definition 8. *(Fairness for deterministic algorithms) Fix an (offline or online) deterministic algorithm* ALG*. For an input σ and a group g, let $\text{ALG}_g(\sigma)$ be the number of intervals accepted by* ALG *from group g when the input is σ. Define the* fairness ratio *of* ALG *as:*

$$\rho(\text{ALG}) = \min_{\sigma} \min_{g \in G} \left(\frac{\text{ALG}_g(\sigma)}{\text{OPT}_g(\sigma)} \right).$$

Example 3.3.1. *Let* ALG *be a deterministic algorithm. Let σ_1 and σ_2 be two input sequences. Assume there are exactly four groups. Suppose the ratio between the benefit of A and* OPT *for the four groups of the input σ_1 for the four groups are given by $\left(\frac{3}{10}, \frac{1}{5}, \frac{1}{10}, \frac{1}{2}\right)$, which gives a minimum of $1/10$ for σ_1. Similarly, these ratios on input σ_2 are $\left(\frac{1}{100}, \frac{1}{1000}, \frac{1}{10}, \frac{889}{1000}\right)$, which gives a ratio of $1/10$ for σ_2. Hence, we can conclude that the fairness ratio of* ALG *is at most $1/1000$ (it could be worse if a smaller ratio is realized by another sequence).*

Ideally, one wants to achieve deterministic fairness ratio guarantees, as they are the strongest notion of fairness. As we will see, however, an adversary can often force the fairness ratio of deterministic algorithms to be arbitrarily bad. Therefore, we often need randomized algorithms to achieve fair solutions. The fairness ratio for randomized algorithms is defined analogously to deterministic algorithms.

Definition 9. *(Fairness for randomized algorithms). Let* ALG *be a fixed randomized algorithm. For an input σ and a group g, let $E(\text{ALG}_g(\sigma))$ be the expected number of intervals accepted by* ALG *from group g when the input is σ. Define the* fairness ratio *of an algorithm* ALG *as:*

$$\rho(\text{ALG}) = \min_{\sigma} \min_{g \in G} \left(\frac{E(\text{ALG}_g(\sigma))}{\text{OPT}_g(\sigma)} \right).$$

Asymptotic vs Absolute Settings. We further distinguish an *asymptotic* setting of the problem from its default *absolute* setting. Under the asymptotic setting, we assume that the benefit of an optimal scheduling for the set of intervals from each group is arbitrarily large. In other words, there is a large number of non-overlapping intervals from each group. This assumption is absent in the absolute setting. Intuitively, achieving fairness is easier under the asymptotic setting, as there is more flexibility in accepting/rejecting intervals; for example, in the asymptotic setting, an interval of length T can be safely rejected, as it is guaranteed that the input contains a large number of non-overlapping small intervals of the same group.

Chapter 4

Purely Online Rectangle Scheduling Problem

4.1 Overview and Summary of Results

In this chapter, we study the *online rectangle scheduling problem*, a two-dimensional generalization of the classical interval scheduling problem, where instead of intervals, we need to schedule rectangles. Recall that we assume the *any-order* setting for rectangle scheduling, where no assumptions are made on the order in which rectangles arrive. To the best of our knowledge, this setting of the problem has not been previously studied.

Proposition 10. *The (absolute) competitive ratio of any deterministic square (and rectangle) scheduling algorithm is* $\Omega(T^2)$. *The asymptotic competitive ratio of any deterministic square (and rectangle) scheduling algorithm is* $\Omega(T)$.

Proof. In the absolute case, a single $T \times T$ square appears, and an algorithm either accepts it, in which case T^2 non-overlapping unit squares appear within the accepted square, or rejects it, in which case the input ends. For the asymptotic setting, consider a sequence of length $\Omega(\sqrt{T})$ formed by $\sqrt{T}$ phases. For each phase, if a deterministic algorithm accepts the square, T unit squares appear within the accepted square, which cannot be accepted by the algorithm. Given that for each accepted square by the algorithm, T squares are accepted by OPT, the lower bound for the competitive ratio would be T. □

In light of the above observation, we focus solely on the randomized algorithm. Our contributions for this chapter are listed below:

- In Section 4.2, we present an extension of the CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT (CLRS) algorithm for interval scheduling to the online rectangle scheduling problem. We prove that 2DCLRS has a competitive ratio of $O(\log^2 T)$.
- In Section 4.3,we show that 2DCLRS is optimal in terms of competitiveness (Theorem 11). That is, we prove that any randomized algorithm has a competitive ratio $\Omega(\log^2 T)$ (Theorem 12).

Our results establish a separation between rectangle scheduling and square scheduling. Recall that [2] has proved that $\Theta(\log T)$ is the best attainable competitive ratio for square scheduling, while our results show that this bound is $\Theta(\log^2 T)$ for the more general rectangle scheduling problem.

4.2 Algorithm and Upper Bound for Competitive Ratio

In this section, we propose a simple randomized algorithm for online rectangle scheduling with a competitive ratio of $O(\log^2 T)$.

We present the 2DCLRS algorithm for online rectangle scheduling, defined as follows:

2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT (2DCLRS)

Input: An online sequence $\langle R_1, R_2, \ldots, R_n \rangle$, for some known T, where R_i is a rectangle on $T \times T$ grid (corners are integers).

Output: A subset of rectangles that are mutually non-overlapping.

Classification: Consider the following classification by side-lengths; we say a rectangle belongs to class (c, c'), if its width is in $[2^c, 2^{c+1})$ and its height is in $[2^{c'}, 2^{c'+1})$. We have $c, c' \in [0, \log T]$ Given that all widths and heights are bounded by T, the set C of all classes has $(\log T + 1) \times (\log T + 1)$ possible classes in it.

Random Selection: Pick two values $c_w, c_h \in [0, \log T]$, each independently from the other and uniformly at random in the range.

Online Process: Suppose rectangle R_j appears. Accept R_j if and only if it does not intersect a previously accepted rectangle, and it belongs to class (c_w, c_h).

Theorem 11. *The* 2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT *algorithm has a competitive ratio of* $O(\log^2 T)$.

Proof. We use ALG to refer to 2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT. For a class (c, c'), let $\text{OPT}_{(c,c')}$ be the optimal scheduling of an input formed by

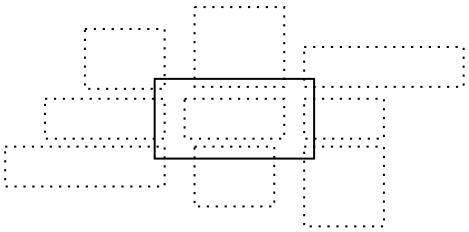


FIGURE 4.1: Illustration showing that accepting one rectangle blocks at most 9 rectangles in the same class (c, c').

items of class (c, c'). We claim that, if the algorithm selects class (c, c'), its benefit is at least $\text{OPT}(c, c')/9$. This holds because the algorithm greedily accepts rectangles of class (c, c'), and accepting a rectangle of such class can result in rejecting at most 9 rectangles that follow it. This is because the side lengths of the rectangles of class (c, c') differ by at most a factor of 2. Therefore, if 2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT accepts a rectangle R, at most one rectangle of the same class fits fully inside R, and one would intersect any of its four sides and any of its four corners. Figure 4.1 provides an illustration.

So, the benefit of a Greedy approach when applied to rectangles of class (c, c') is at least $\text{OPT}_{(c,c')}/9$. Given that there are a total of $(\log T + 1)^2$ classes and ALG selects one uniformly at random, for the expected benefit of the algorithm, we can write

$$E[\text{ALG}] = \frac{1}{(\log T + 1)^2} \sum_{(c,c') \in C} \text{OPT}_{c,c'}/9. \tag{4.1}$$

Now, let $O_{c,c'}$ be the set of rectangles in the optimal solution from class (c, c'). So, we have $\text{OPT} = \sum_{(c,c') \in C} O_{c,c'}$. Note that $O_{c,c'} \leq \text{OPT}_{(c,c')}$, this is because the rectangles in $O_{(c,c')}$ form a valid (but possibly sub-optimal) solution for the input rectangles that belong to class (c, c'). Summing up over all classes, we can write $\sum_{(c,c')} \text{OPT}_{(c,c')} \geq \sum_{(c,c')} O_{(c,c')} = \text{OPT}$. Combining with equality 4.1, we can write:

$$E[\text{ALG}] \geq \frac{\text{OPT}}{9 \cdot (\log T + 1)^2}$$

Thus, the competitive ratio of the algorithm is at most $9(\log T + 1)^2$, establishing the claimed bound of $O(\log^2 T)$. □

4.3 General Lower Bound for Competitive Ratio

In this section, we establish a lower bound of $\Omega(\log^2 T)$ for the competitive ratio of any randomized (or deterministic) algorithm for the online rectangle scheduling problem based on [16].

Overview of the proof. Our proof relies on *Yao's Minimax Principle* (See Definition 1 in Section 2.1). In what follows, we fix a deterministic algorithm ALG. Then we will define a family of input sequences $\sigma_{i,j}$ and assume ALG is faced with a member of this family that is drawn randomly according to a distribution that we will describe. We show that the expected benefit of ALG when faced with a random input sequence $\sigma_{i,j}$ is no better than 1, while the expected benefit of OPT is $\Theta(\log^2 T)$. This shows the ratio between the expected benefit of OPT and ALG is at least $\Omega(\log^2 T)$. Applying Yao's theorem, the proof completes.

Input Family. Let R_p denote a general rectangle, written as $R_p = (x_p, y_p, w_p, h_p)$, where x_p, y_p specify the coordinates of the bottom-left corner, and w_p, h_p specify the width and height, respectively. For any $i, j \in [0, \log T]$, define $C_{i,j}$ as the set of rectangles obtained by partitioning the square $[0, 0, T, T]$ into 2^i vertical slices This means that the rectangles in $C_{i,j}$ all have a width and height of $T/2^i$ and $T/2^j$, respectively. In particular, a member $R_{x,y}$ of class $C_{i,j}$ for $x \in [0, 2^i)$ and $y \in [0, 2^j)$ can be specified as

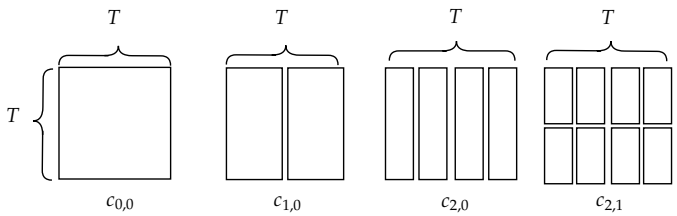
$$R_{x,y} = \left\{ \left(\frac{xT}{2^i}, \frac{yT}{2^j}, \frac{T}{2^i}, \frac{T}{2^j} \right) \mid x \in [0, 2^i),\ y \in [0, 2^j) \right\}.$$

Provided with the above definition of $C_{i,j}$, define an online input sequence $\sigma_{i,j}$ as follows (as before $i, j \in [0, \log T]$)

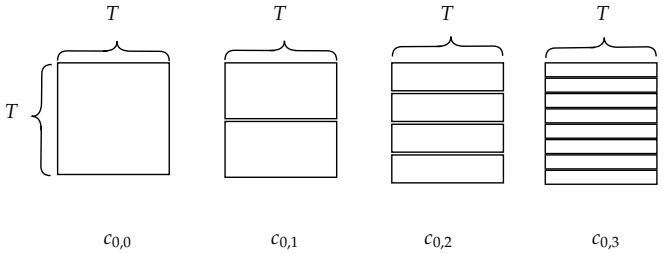
$$\sigma_{i,j} = \langle C_{0,0}, C_{1,0}, \ldots, C_{i,0}, C_{i,1}, \ldots, C_{i,j} \rangle.$$

In the above definition, presence of $C_{a,b}$ in $\sigma_{i,j}$ means inclusion of all rectangles in the set $C_{a,b}$ in the $\sigma_{i,j}$ (and they appear in an arbitrary order).

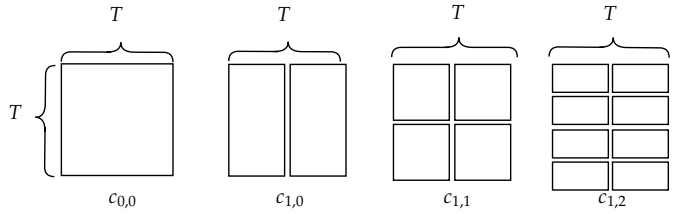
Example 4.3.1. *The following shows $\sigma_{2,1}$. Note that all rectangles' endpoints are on the $[0, T] \times [0, T]$ grid; they are, however, depicted separately for illustrative reasons.*



The following shows $\sigma_{0,3}$:



The following shows $\sigma_{1,2}$:



Input Distribution. The input distribution is as follows: Select $\sigma_{i,j}$ with probability

$$p_{i,j} = \frac{T^2}{(2T-1)^2 \cdot 2^{i+j}}.$$

We can show that this defines a probability distribution, that is, the sum of all probabilities is exactly 1:

$$\begin{aligned}
\sum_{i=0}^{\log T} \sum_{j=0}^{\log T} p_{i,j} &= \sum_{i=0}^{\log T} \sum_{j=0}^{\log T} \frac{T^2}{(2T-1)^2 \cdot 2^{i+j}} \\
&= \frac{T^2}{(2T-1)^2} \sum_{i=0}^{\log T} \left(\frac{1}{2^i} \sum_{j=0}^{\log T} \frac{1}{2^j} \right) \\
&= \frac{T^2}{(2T-1)^2} \sum_{i=0}^{\log T} \left(\frac{1}{2^{i+1}} \frac{2T-1}{T} \right) \\
&= \frac{T}{2T-1} \sum_{i=0}^{\log T} \left(\frac{1}{2^i} \right) \\
&= \frac{T}{2T-1} \cdot \frac{2T-1}{T} = 1.
\end{aligned}$$

Intuitively, to generate a member of this input family, we can consider the following stochastic process, which works in two phases.

- **Initialization:** At the beginning, we are at Phase I, and $C_{0,0}$ (a large T by T square) appears.
- **Phase I:** Suppose we are at Phase I, and rectangles in $C_{i,0}$ have just appeared. If $i = \log T$, Phase I ends and Phase II begins. If $i < \log T$, with a chance of $1/2$, rectangles in $C_{i+1,0}$ appear next, and with a chance of $1/2$, Phase I ends and Phase II begins.
- **Phase II:** Suppose we are at Phase II, and rectangles in $C_{i,j}$ have just appeared; with a chance of $1/2$, this is followed by rectangles in $C_{i,j+1}$ and with a chance of $1/2$, the input ends. If $i = \log T$, Phase II also ends.

Theorem 12. *The lower bound on the competitive ratio of any randomized algorithm for the online rectangle scheduling problem is* $\Omega(\log^2 T)$.

Proof. We show that the benefit of OPT for a random member $\sigma_{i,j}$ of the input distribution we described is $\Omega(\log^2 T)$, while that of any deterministic algorithm ALG is at most 1. Applying Yao's theorem (Definition 1), we can conclude that no randomized algorithm can have a competitive ratio better than $O(\log^2 T)$.

Expected Benefit of OPT.

We first establish the described lower bound for the benefit of OPT. Consider OPT on a random input σ. If the input terminates at level (i, j), then OPT accepts all rectangles from $C_{i,j}$ and rejects all rectangles from any other level (i', j') with $i' < i$ and $j' < j$. Given that the probability the input terminates at $C_{i,j}$ is $\frac{1}{2^{i+j}} \cdot \left(\frac{T}{2T-1}\right)^2$, and there are 2^{i+j} rectangles in $C_{i,j}$, we can write:

$$E[\text{OPT}(\sigma)] = \sum_{i=0}^{\log T} \sum_{j=0}^{\log T} 2^{i+j} \cdot \frac{1}{2^{i+j}} \cdot \left(\frac{T}{2T-1}\right)^2 \tag{4.2}$$

$$= \frac{T^2}{(2T-1)^2} \sum_{i=0}^{\log T} \sum_{j=0}^{\log T} 1 \tag{4.3}$$

$$= \frac{T^2}{(2T-1)^2} \cdot (\log T + 1)^2 > \frac{(\log T)^2}{4}. \tag{4.4}$$

Expected Benefit of ALG.

In order to analyze the benefit of ALG, we establish four claims that, together,

establish that the best strategy for any online algorithm is to accept the first large square (of size $T \times T$) at the beginning. In other words, the expected benefit of ALG cannot be better than 1.

- The first claim shows that the chance of seeing another set of rectangles in Phase II of input generation is no more than $1/2$. In other words, if we are at Phase II, then with a chance of at least $1/2$, the current set is the last set of rectangles we see in Phase II. Formally, this claim can be stated as follows.

  Claim 1: *For a fixed value of $i \in [0, \log T]$ and any value of $j \in [1, \log T]$, we have:*

  $$\Pr[\text{ALG sees rectangles in } C_{i,j} \text{ in the input } \mid \text{ ALG sees rectangles in } C_{i,j-1} \text{ in the input }] \leq \frac{1}{2}.$$

  proof of Claim 1: Suppose the algorithm sees rectangles in $C_{i,j-1}$; that means the input must end with at some level i, ℓ such that $\ell \geq j-1$, which means that means the input has started with

  $$\langle C_{0,0}, C_{1,0}, \ldots, C_{i,0}, C_{i,1}, \ldots, C_{i,j-1} \rangle.$$

  Thus, for the chance of the input ending at $C_{i,j-1}$ or later (i.e., it sees $C_{i,j-1}$ and possibly more rectangles after), we can write

  $$\begin{aligned}
  \Pr[\text{ALG sees rectangles from } C_{i,j-1}] &= \sum_{k=j-1}^{\log T} p_{i,k} \\
  &= \sum_{k=j-1}^{\log T} \frac{T^2}{(2T-1)^2 \cdot 2^{i+k}} \\
  &= \frac{T^2}{(2T-1)^2 \cdot 2^i} \sum_{k=j-1}^{\log T} \frac{1}{2^k} \\
  &= \frac{T^2}{(2T-1)^2 \cdot 2^i} \cdot (2^{2-j} - \frac{1}{T}).
  \end{aligned}$$

  Moreover, for the algorithm to see rectangle in $C_{i,j}$, the input must end with at some level $\{i, \ell'\}$ such that $\ell' \geq j$; that means the input has started with

  $$\langle C_{0,0}, C_{1,0}, \ldots, C_{i,0}, C_{i,1}, \ldots, C_{i,j} \rangle.$$

We can conclude:

$$\begin{aligned}\Pr[\text{ALG sees rectangles from } C_{i,j}] &= \sum_{k=j}^{\log T} p_{i,k} \\ &= \sum_{k=j}^{\log T} \frac{T^2}{(2T-1)^2 \cdot 2^{i+k}} \\ &= \frac{T^2}{(2T-1)^2 \cdot 2^i} \sum_{k=j}^{\log T} \frac{1}{2^k} \\ &= \frac{T^2}{(2T-1)^2 \cdot 2^i} \cdot (2^{1-j} - \frac{1}{T})\end{aligned}$$

To summarize, we can write:

$$\begin{aligned}&\Pr[\text{ALG sees rectangles from } C_{i,j} \mid \text{ALG sees rectangles from } C_{i,j-1}] \\ &\qquad = \frac{\Pr[\text{ALG sees rectangles from } C_{i,j}]}{\Pr[\text{ALG sees rectangles from } C_{i,j-1}]} \\ &\qquad = \frac{2^{1-j} - \frac{1}{T}}{2^{2-j} - \frac{1}{T}} \\ &\qquad = \frac{1}{2} - \frac{1}{2 \cdot T \cdot (2^{2-j} - \frac{1}{T})} < \frac{1}{2}.\end{aligned}$$

This concludes the proof of Claim 1.

- The second claim states that there is no benefit in rejecting a rectangle in Phase II.

Claim 2: *If* ALG *rejects a rectangle* $R' \in C_{i,j}$ *for a fixed value of* $i \in [0, \log T]$ *and any value of* $j \in [0, \log T]$*, and* R' *does not intersect any previously accepted rectangle (i.e., it could be accepted), then*

$$E[\textit{eventual benefit of rejecting } R'] \leq 1 = \textit{eventual benefit of accepting } R'.$$

proof of Claim 2. We prove this claim by *backward induction* on $j = (\log T, \log T - 1, \ldots, 0)$.

Base case. When $j = \log T$, there are no future rectangles after this level. Therefore, if ALG rejects R', the actual benefit of rejecting is 0, whereas accepting

R' yields a benefit of 1. Hence, the claim trivially holds.

Inductive step. Assume the claim holds for level (i, j); that is, rejecting any rectangle at level (i, j) gives an expected benefit at most 1.

We now prove the claim for level $(i, j-1)$. Consider rejecting a rectangle R' at level $(i, j-1)$. By construction, rejecting R' potentially allows acceptance of two rectangles R'_1 and R'_2, whose union is exactly R'. These rectangles belong to the next level (i, j).

By Claim 1, with probability of at most $\frac{1}{2}$, the algorithm will see R'_1 and R'_2. For each of these rectangles, by the inductive hypothesis, the expected benefit of rejecting is at most 1. Thus, the expected benefit of rejecting R' is at most

$$\frac{1}{2}\left(E[\text{Benefit from } R'_1] + E[\text{Benefit from } R'_2]\right) \leq \frac{1}{2}(1+1) = 1.$$

Therefore, ALG can either accept R' and obtain a benefit of 1, or reject it and obtain an expected benefit at most 1. In either case, the benefit from rejecting does not exceed the benefit from accepting.

- The third claim is similar to Claim I, except that it concerns Phase I of input generation. It asserts that the chance of seeing the next set of rectangles in Phase I of input generation is no more than $1/2$. In other words, with a chance of at least $1/2$, the current set is the last set of rectangles we see in Phase I. Formally, this claim can be stated as follows.

Claim 3: *For a value of $i \in [1, \log T]$, we have:*

$$\Pr[\text{ALG sees rectangles from } C_{i,0} \mid \text{ALG sees rectangles from } C_{i-1,0}] \leq \frac{1}{2}.$$

proof of Claim 3. The proof is very similar to that of Claim 1. Suppose the algorithm sees rectangles in $C_{i-1,0}$. That means the input must end with at some level $\{\ell', 0\}$ such that $\ell' \geq i-1$. Hence, the input has started with

$$\langle C_{0,0}, C_{1,0}, \ldots, C_{i-1,0} \rangle.$$

Now, we can write

$$\begin{aligned}\Pr[\text{ALG sees rectangles from } C_{i-1,0}] &= \sum_{k=i-1}^{\log T} p_{k,0} \\ &= \sum_{k=i-1}^{\log T} \frac{T^2}{(2T-1)^2 \cdot 2^k} \\ &= \frac{T^2}{(2T-1)} \cdot (2^{2-i} - \frac{1}{T}).\end{aligned}$$

Moreover, for the algorithm to see rectangles in $C_{i,0}$, Phase I must end with some level $\{\ell, 0\}$ such that $\ell' \geq i$. Thus, the input should start with

$$\langle C_{0,0}, C_{1,0}, \ldots, C_{i,0} \rangle$$

We can conclude:

$$\begin{aligned}\Pr[\text{ALG sees rectangles from } C_{i,0}] &= \sum_{k=i}^{\log T} p_{k,0} \\ &= \sum_{k=i}^{\log T} \frac{T^2}{(2T-1)^2 \cdot 2^k} \\ &= \frac{T^2}{(2T-1)^2} \cdot (2^{1-i} - \frac{1}{T}).\end{aligned}$$

To summarize, we can state

$$\begin{aligned}&\Pr[\text{ALG sees rectangles from } C_{i,0} \mid \text{ALG sees rectangles from } C_{i-1,0}] \\ &\qquad = \frac{\Pr[\text{ALG sees rectangles from } C_{i,0}]}{\Pr[\text{ALG sees rectangles from } C_{i-1,0}]} \\ &\qquad = \frac{2^{1-i} - \frac{1}{T}}{2^{2-i} - \frac{1}{T}} \\ &\qquad = \frac{1}{2} - \frac{1}{2 \cdot T \cdot (2^{2-i} - \frac{2}{T})} < \frac{1}{2}.\end{aligned}$$

- The last claim states that there is no benefit in rejecting a rectangle in Phase I.

  Claim 4: If ALG rejects a rectangle $R^* \in C_{i,0}$, and R^* does not intersect any previously accepted rectangle, then

$$E[\text{eventual benefit of rejecting } R^*] \leq 1 = \text{eventual of accepting } R^*.$$

We prove this claim by *backward induction* on $i = (\log T, \log T - 1, \ldots, 0)$.

Base case. When $i = \log T$, Phase I ends, and Phase II begins. By Claim II, the benefit of Alg at Phase II is maximized if it accepts the first rectangle R^*.

Inductive step. Assume the claim holds for level $(i, 0)$; that is, rejecting any rectangle at level $(i, 0)$ gives an expected benefit at most 1.

We now prove the claim for level $(i - 1, 0)$. Consider rejecting a rectangle R^* at level $(i, 0)$. By construction, rejecting R^* potentially allows acceptance of two rectangles R_1^* and R_2^*, whose union is exactly R^*. These rectangles belong to the next level $(i, 0)$.

By Claim 3, with probability $\frac{1}{2}$, the algorithm will see R_1^* and R_2^*. For each of these rectangles, by the inductive hypothesis, the expected benefit of rejecting is at most 1. Thus, the expected benefit of rejecting R^* is at most

$$\frac{1}{2}\left(E[\text{Benefit from } R_1^*] + E[\text{Benefit from } R_2^*]\right) \leq \frac{1}{2}(1+1) = 1.$$

Therefore, ALG can either accept R^* and obtain a benefit of 1, or reject it and obtain an expected benefit at most 1. In either case, the benefit from rejecting does not exceed the benefit from accepting.

By Claim 4, the expected benefit of ALG is maximized if it accepts the first rectangle (of size $T \times T$), which yields a benefit of 1. Given that the expected benefit of OPT is at least $\frac{\log^2 T}{4} = \Theta(\log^2 T)$ (Equation 4.4), it follows that the lower bound for the competitive ratio of ALG is $\Omega(\log^2 T)$. □

Therefore, we have established tight bounds of $\Theta(\log^2 T)$ for the competitive ratio of the online rectangle scheduling problem, and 2DCLRS is the optimal algorithm. In the next chapter, we will discuss how predictions can be leveraged to the online rectangle scheduling problem.

Chapter 5

Online Rectangle Scheduling with Predictions

5.1 Overview and Summary of Results

In this chapter, we study the RECTANGLE SCHEDULING PROBLEM with predictions. Building on recent advances in learning-augmented algorithms, it is natural to explore the problem under such a setting. In particular, we extend the framework from [20], originally devised for online interval scheduling, to the two-dimensional rectangle scheduling problem. Specifically, we assume access to an oracle that gives a prediction that specifies the rectangles in the input.

Our contributions for this chapter are summarized as follows:

- In Section 5.2, we describe the prediction and error model we used in our algorithms and analysis.
- In Section 5.3, we prove that no online algorithm with predictions can achieve a competitive ratio better than $\frac{1}{1-\gamma}$, where γ is the error quantifier (Theorem 13).
- In Section 5.4, we analyze an algorithm named 2DTRUST, which strictly follows the prediction and rejects all other rectangles. We provide matching upper and lower bounds to prove that the competitive ratio of 2DTRUST is exactly $\frac{1}{1-2\gamma}$ (Theorems 14 and 15). In particular, 2DTRUST is not *robust* in the sense that its competitive ratio is high when the error γ takes larger values.
- In Section 5.5, we propose and analyze a hybrid algorithm that combines 2DTRUST with 2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT to achieve a more robust algorithm. The algorithm has a parameter λ which allows a trade-off between consistency (performance when error is 0) and robustness (performance when error takes its maximum value). In particular, we prove that this algorithm has λ-consistency and $\frac{1-\lambda}{9\cdot(\log T+1)^2}$-robustness (Theorem 16).

- In Section 5.6, we use connection to advice to show that other types of predictions, in particular small predictions that can be encoded in a sublinear number of bits, cannot yield optimal solutions, even if they are error-free. More precisely, we show that with sublinear advice (thus predictions of sublinear size), the competitive ratio cannot be better than $\frac{4}{3}$ (Theorem 18).

Our positive (algorithmic) results in this section hold for the general version of the problem (that is, inputs with arbitrary rectangles), while our negative results apply to the restricted setting of square scheduling. In other words, there is no separation between square scheduling and the general formulation of rectangle scheduling in the prediction setting.

5.2 Prediction Model & Error

In this section, we formalize the prediction model and define the corresponding error metrics used to evaluate the performance of our algorithm. Following Boyar et al. [20], we assume that the prediction specifies, for each possible rectangle, whether it is present or absent from the input sequence. Observe that there are $\binom{(T+1)^2}{2} = O(T^4)$ possible rectangles on a $T \times T$ grid. For each of these rectangles, a bit in the prediction sequence indicates whether the rectangle is present in the input or not. Although this constitutes a long prediction, the limitations of shorter and simpler predictions are discussed in Section 5.6.

5.2.1 Classification of Rectangles and Error Metrics

We denote by R the set of rectangles in the actual input and by $\hat{R}$ the set of rectangles predicted to appear in the input. In our analysis, rectangles can be classified into the following four categories:

- **True Positives (TP):** Rectangles that appear in both $\hat{R}$ and R.
- **True Negatives (TN):** Rectangles that appear in neither $\hat{R}$ nor R.
- **False Positives (FP):** Rectangles that appear in $\hat{R}$ but not in R.
- **False Negatives (FN):** Rectangles that appear in R but not in $\hat{R}$.

Note that the actual input R consists of the rectangles in $\mathrm{TP} \cup \mathrm{FN}$, while the prediction consists of the rectangles in $\mathrm{TP} \cup \mathrm{FP}$. We define the *error parameter* $\eta(\hat{R}, R)$ as the size of the optimal solution for the set of the incorrectly predicted rectangles, i.e., the rectangles in $\mathrm{FN} \cup \mathrm{FP}$. The *error parameter ratio* $\gamma(\hat{R}, R)$ is defined as the ratio between this error parameter and the optimal cardinality of the actual input:

$$\eta(\hat{R}, R) = \textsc{Opt}(\mathrm{FN} \cup \mathrm{FP}), \quad \text{and} \quad \gamma(\hat{R}, R) = \min\left\{\frac{\eta(\hat{R}, R)}{|\textsc{Opt}(R)|}, 1\right\}.$$

When there is no risk of confusion, we simply write η for $\eta(\hat{R}, R)$ and γ for $\gamma(\hat{R}, R)$.

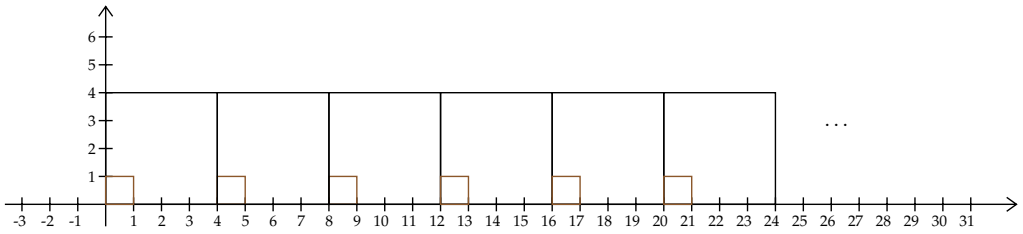
5.3 Impossibility Results for General Algorithms

We begin by presenting a negative result regarding the competitive ratio of learning-augmented algorithms.

Theorem 13. *Let* ALG *be any learning-augmented deterministic algorithm for the online square scheduling problem. For any positive integer p and any constant $c \geq 2$, there exists an instance R and a prediction $\hat{R}$ such that*

$$\text{ALG}(\hat{R}, R) \leq (1 - \gamma)\text{OPT}(R).$$

Proof. Let the prediction $\hat{R}$ contain $2p$ squares, formed across phases p indexed by $i \in [0, p-1]$. At each phase i, the prediction $\hat{R}$ specifies the presence of two squares: one with bottom-left corner at $(c \cdot i, 0)$ and side length c, and one square of unit side length with bottom-left corner also at $(c \cdot i, 0)$. The predicted squares are illustrated one the figure below.



Let ALG be any deterministic online algorithm. The adversary reveals the actual input R only after observing the first decision made by ALG at each phase. The actual input R consists of p independent *phases*, denoted p_i for $i \in [0, p-1]$. We denote $\hat{R}_i$ as the prediction for phase p_i, and R_i as the actual input revealed in phase p_i.

Each phase in the actual input R_i begins with a large square whose bottom-left corner is at the point $(c \cdot i, 0)$ and whose side length is c. Depending on the decision made by ALG on the first square in phase p_i, the adversary reveals the subsequent square in that phase as follows:

- **Case 1:** If ALG accepts the large square, the adversary reveals c non-overlapping unit squares fully contained within it (as illustrated in Figure 5.1a). Since these unit squares intersect the accepted large square, ALG cannot accept any of them. The optimal solution OPT accepts all c unit squares, among which $c-1$ are false negatives. Therefore:

$$\text{ALG}_i = 1, \quad \text{OPT}_i = c, \quad \eta_i = c - 1.$$

- **Case 2:** If ALG rejects the large square, no further square appears in that phase (as illustrated in Figure 5.1b). The optimal solution OPT accepts the large square, and no further rectangles are revealed. In this case:

$$\text{ALG}_i = 0, \quad \text{OPT}_i = 1, \quad \eta_i = 1.$$

In both cases, it holds that

$$\text{ALG}(\hat{R}_i, R_i) \leq \text{OPT}(R_i) - \eta.$$

Since the phases are disjoint, summing over all phases yields:

$$\text{ALG}(\hat{R}, R) \leq \text{OPT}(R) - \eta = (1-\gamma)\text{OPT}(R).$$

□

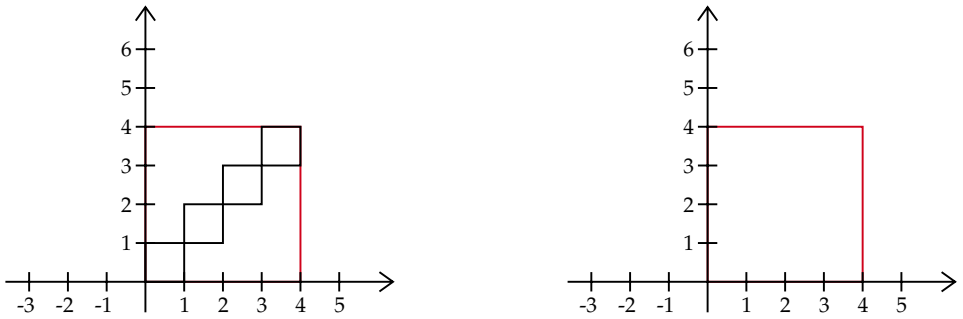
5.4 The 2DTRUST Algorithm

Next, we present the 2DTRUST algorithm, which fully trusts the optimal solution generated by the given prediction. The details of the algorithm are provided below.

The 2DTRUST Algorithm for Rectangle Scheduling

Input. A predicted set of rectangles $\hat{R} = \{\hat{R}_1, \hat{R}_2, \ldots, \hat{R}_n\}$ and an online sequence $R = \langle R_1, R_2, \ldots, R_n \rangle$.

Output. A subset of rectangles that are mutually non-overlapping.



(A) Case 1: Large Square Accepted, Unit Squares Revealed

(B) Case 2: Large Square Rejected, No Further Rectangles

FIGURE 5.1: Construction of Adversarial Inputs under Phased Rectangles (With $c = 4$)

Planning. An optimal solution $\hat{R}^*$ is computed based on the set of predicted rectangles $\hat{R}$.

Online Process. The online process works as follows: the actual input sequence R is revealed in an online manner. For each arriving rectangle $r \in R$, the algorithm proceeds as follows:

- If $r \in \hat{R}^*$, accept r.
- If $r \notin \hat{R}^*$, reject r.

In what follows, we show that the competitive ratio of 2DTRUST is at most $\frac{1}{1-2\gamma}$.

Theorem 14. *The algorithm* 2DTRUST *is at most* $\frac{1}{1-2\gamma}$*-competitive.*

Proof. Given the prediction $\hat{R}$, 2DTRUST computes the offline solution $\hat{R}^*$. Since the prediction consists of TP and FN, the $\hat{R}^*$ will be at least $\textsc{Opt}(TP)$. That is,

$$\hat{R}^* = \textsc{Opt}(TP \cup FP) \geq \textsc{Opt}(TP). \tag{5.1}$$

Moreover, the optimal online solution on the input R will be at most the sum of the optimal solution of TP and FN. We can write

$$\textsc{Opt}(R) = \textsc{Opt}(\text{TP} \cup \text{FN}) \leq \textsc{Opt}(\text{TP}) + \textsc{Opt}(\text{FN}) \tag{5.2}$$

$$\implies \textsc{Opt}(\text{TP}) \geq \textsc{Opt}(R) - \textsc{Opt}(\text{FN}). \tag{5.3}$$

Moreover, if we only follow the 2DTRUST algorithm, we only accept the rectangles belonging to the optimal offline solution $\hat{R}^*$ and reject the other rectangles. The largest

number of rectangles that can be deducted from 2DTRUST is the optimal solution on the FP, which are the rectangles predicted to appear but do not appear. Hence, we will have the following inequalities:

$$
\begin{aligned}
2\text{DTRUST}(\hat{R}, R) &\geq \hat{R}^* - \text{OPT}(\text{FP}) \\
&\geq \text{OPT}(\text{TP}) - \text{OPT}(\text{FP}) && \text{from (5.1)} \\
&\geq \text{OPT}(R) - \text{OPT}(\text{FN}) - \text{OPT}(\text{FP}) && \text{from (5.3)} \\
&\geq \text{OPT}(R) - \text{OPT}(\text{FN} \cup \text{FP}) - \text{OPT}(\text{FP} \cup \text{FN}) \\
&\geq \text{OPT}(R) - 2\text{OPT}(\text{FP} \cup \text{FN}) \\
&= \text{OPT}(R) - 2\eta(\hat{R}, R) = (1 - 2\gamma)\text{OPT}(R)
\end{aligned}
$$

□

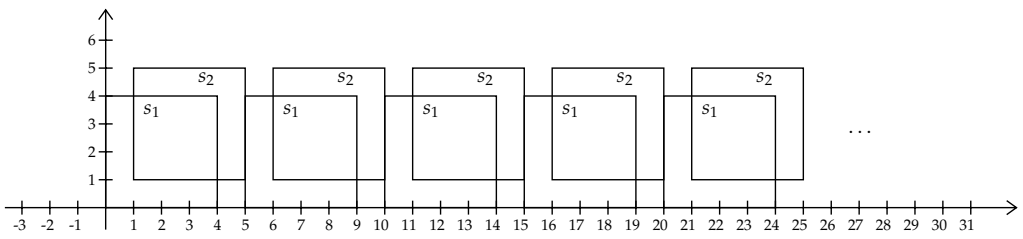
Next, we show the upper bound in the above theorem is indeed tight.

Theorem 15. *For any positive integer p and some constant $c \geq 2$, there exists a prediction $\hat{R}$ and an input sequence R such that*

$$2\text{DTRUST}(\hat{R}, R) = (1 - 2\gamma)\text{OPT}(R).$$

Proof. Consider a prediction $\hat{R}$ consisting of p phases. In each phase $p_i \in [0, p-1]$, the prediction includes two squares of side length c, as illustrated below:

- *Square 1*: A square S_1 with bottom-left corner at $((c+1) \cdot i, 0)$;
- *Square 2*: Another square S_2 with bottom-left corner at $((c+1) \cdot i + 1, 1)$.



In each phase p_i, $\text{OPT}(\hat{R})$ can select only one of the two squares since they overlap.

W.l.o.g., $\textsc{Opt}(\hat{R})$ selects S_1 of each phase. Then, the adversary reveals the following squares (as shown in Figure 5.2):

- A square S_2 with bottom-left corner at $((c+1)\cdot i+1, 1)$ and of side length c. This is Square S_2 of phase i and contributes to TP.
- A unit square S_s with bottom-left corner at $((c+1)\cdot i, 0)$. This square is absent in $\hat{R}$ and contributes to FN.

Note that S_1 in phase i, which was in the schedule of the algorithm (to be accepted), never appears and contributes to FP.

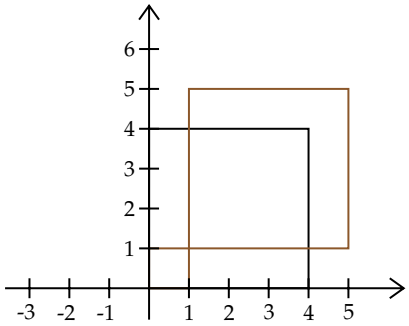


FIGURE 5.2: First square selected; adversary reveals large and unit squares.

2DTRUST accepts zero rectangles from the revealed input, while the optimal solution accepts two rectangles. Therefore, for each phase i, we have

$$\text{2DTRUST}_i = 0, \quad \textsc{Opt}_i = 2, \quad \textsc{Opt}(\text{FP} \cup \text{FN}) = 1.$$

That is,

$$\text{2DTRUST}_i = 0 = \textsc{Opt}_i - 2\textsc{Opt}(\text{FP} \cup \text{FN}).$$

With p phases, this extends to

$$\begin{aligned}\text{2DTRUST}(\hat{R}, R) &= 0 = \textsc{Opt}(R) - 2\textsc{Opt}(\text{FP} \cup \text{FN}) \\ &= (1-2\gamma)\textsc{Opt}(R).\end{aligned}$$

□

From the above two theorems, we conclude that the competitive ratio of the 2DTRUST algorithm is $\frac{1}{1-2\gamma}$. Note that, as γ approaches to $1/2$, the competitive ratio

$\frac{1}{1-2\gamma}$ declines arbitrarily large. In particular, the performance of 2DTRUST becomes pretty poor when the prediction error increases.

5.5 A Robust Algorithm & Consistency-Robustness Trade-off

Recall that consistency of a learning-augment online algorithm captures its performance when prediction is perfect (the competitive ratio when error is $\gamma = 0$), and robustness captures the competitive ratio when the predictions are adversarial (the competitive ratio when error is $\gamma = 1$).

By Theorem 14 and Theorem 15, the competitive ratio of 2DTRUST is $\frac{1}{1-2\gamma}$ if $\gamma < 0.5$ and infinity otherwise (in the latter case, the adversary can ensure that it never accepts an input while OPT does). Therefore, the consistency of 2DTRUST is 1 (its competitive ratio is 1 when $\gamma = 0$), and its robustness is unbounded. Intuitively, this algorithm relies too heavily on predictions being correct and blindly trusts them. In contrast, a purely online algorithm such as 2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT is optimally robust because it has an optimal competitive ratio.

In this section, we describe a combined family of algorithms that benefits from the supreme consistency of 2DTRUST and the robustness of 2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT. Any member of this family of algorithms has a parameter $\lambda \in [0, 1]$ which specifies how much it trusts the prediction.

The HYBRID-ROBUST Algorithm for Rectangle Scheduling with parameter λ

Input A predicted set of rectangles $\hat{R} = \{\hat{R}_1, \hat{R}_2, \ldots, \hat{R}_n\}$ and an online sequence $R = \langle R_1, R_2, \ldots, R_n \rangle$.

Output A subset of mutually non-overlapping rectangles.

Random Selection: Pick a value of $x \in (0, 1)$ uniformly at random.

Choice Of Algorithms:

- If $x \leq \lambda$, run 2DTRUST with prediction $\hat{R}$ on the online input R
- Else, when $x > \lambda$, run 2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT on the online input R

In particular, when $\lambda = 1$, the algorithms becomes 2DTRUST and when $\lambda = 0$, it becomes CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT

Theorem 16. *The HybridTrustGreedy algorithm has consistency at most $\frac{1}{\lambda}$ and robustness of at most $\frac{9(\log T+1)^2}{1-\lambda}$.*

Proof. Assume that all predictions are accurate, that is, $\gamma = 0$. The *HybridTrustGreedy* algorithm applies the 2DTRUST algorithm with probability λ. On the other hand, 2DTRUST is optimal under accurate predictions by Theorem 14. Intuitively, when predictions are perfect, 2DTRUST follows an optimal schedule that is eventually realized (an item appears if and only if it is predicted, and 2DTRUST's schedule becomes the optimal schedule calculated from the prediction). As a result, the benefit of HYBRID-ROBUST is at least λOPT; that is, the consistency of HYBRID-ROBUST is at least $\frac{1}{\lambda}$.

Conversely, if all predictions are inaccurate, HYBRID-ROBUST applies the 2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT algorithm for online rectangle scheduling (see Section 4.2) with a probability of $1 - \lambda$. The 2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT algorithm guarantees a competitive ratio of at least $\frac{1}{9\lceil(\log T+1)^2\rceil}$ by Theorem 11. Therefore, the benefit of HYBRID-ROBUST is at least $\frac{1-\lambda}{9\lceil(\log T+1)^2\rceil}$OPT. That is, the robustness of HYBRID-ROBUST is at most $\frac{9(\log T+1)^2}{1-\lambda}$. □

We can establish a negative result concerning the consistency-robustness of *deterministic* algorithms. This result holds even for a special setting of scheduling squares.

Theorem 17. *Any deterministic learning-augmented algorithm for the online square scheduling problem that has consistency $\lambda \in \Theta(1)$ has robustness $\Omega(T^2)$*

Proof. Let ℓ be a positive integer such that T is divisible by ℓ and let $p = T/\ell$. We assume $\ell \in \Theta(T)$ is a large value and thus p is a constant. Suppose the prediction $\hat{R} = \{S_0, S_1, \ldots, S_{p-1}\}$ where S_i is of side length ℓ and finds its bottom-left corner at point $(i, 0)$. Fix a deterministic online algorithm ALG. We consider the following two scenarios for the consistency and robustness of ALG:

Consistency Suppose the prediction is perfect. Given that no two squares in $\hat{R}$ (and thus R) intersect, the optimal schedule includes all of the p squares. Thus, for an algorithm to be λ-consistent, it must accept at least $\frac{p}{\lambda}$ squares.

Robustness Suppose the actual input starts with $\langle S_0, S_1, \ldots\rangle$, except that, as soon as $\frac{p}{\lambda}$ squares are accepted by ALG, no further square from $\hat{R}$ is revealed. Note

that, by the consistency requirement, we know that at some point $\frac{p}{\lambda}$ squares will be accepted. At that point, the input continues with ℓ^2 unit squares within any of these accepted squares. Therefore, we have $\text{OPT} = \ell^2 \frac{p}{\lambda}$. On the other hand, ALG cannot accept any of these unit squares, and we have $\text{ALG} = \frac{p}{\lambda}$. This shows a robustness of $\Omega(\ell^2)$. Given that $\ell = \Theta(T)$, the robustness of ALG is $\Omega(T^2)$.

□

5.6 Advice & Impossibility Result for Small Prediction

In this section, we show that, any online algorithm for the RECTANGLE SCHEDULING PROBLEM that has sublinear advice (that is, advice of size $o(n)$, where n is the number of input rectangles) cannot achieve a nearly optimal solution. Indeed, it cannot be better than $\frac{4}{3}$-competitive, even for the SQUARE SCHEDULING PROBLEM (when all input rectangles are squares). This negative result, in a sense, complements our algorithmic result in Chapter 5 in the following sense. Recall that advice can be thought of as a sort of perfect prediction. So, if we cannot achieve a competitive ratio of $\frac{4}{3}$ with sublinear advice, we cannot do that with any prediction of sublinear size, even if the prediction is perfect. For example, statistical information about the input (e.g., average area, average length/width, etc.) cannot help us achieve a near-optimal solution.

The proof of the following theorem uses a reduction from the BINARY GUESSING PROBLEM (see Section 2.5.1 for a formal definition). Recall that an instance of the BINARY GUESSING PROBLEM is a sequence of binary bits appearing online. While it is trivially possible to correctly guess a fraction 0.5 of all bits in the input when provided by a single bit (that specifies the majority), guessing a fraction of $(0.5 + \epsilon)$ requires advice of at least linear size [14].

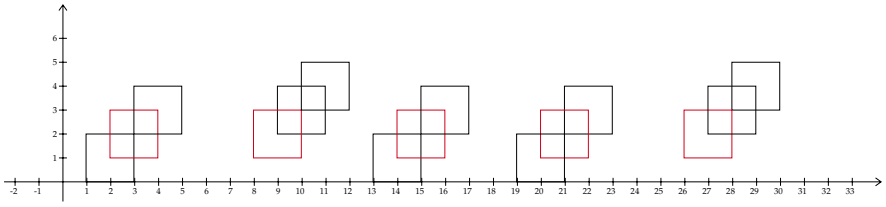
Theorem 18. *The competitive ratio of any online algorithm for the* SQUARE SCHEDULING PROBLEM *(and thus* RECTANGLE SCHEDULING PROBLEM*) that has an advice of size $o(n)$ is at least 4/3. In other words, any algorithm with advice (thus prediction) of sublinear size cannot be better than $\frac{4}{3}$-competitive.*

Proof. Consider an instance $I = \langle 0, 1, \ldots, m-1 \rangle$ of the BINARY GUESSING PROBLEM. We create an instance $\sigma(I)$ of length $n = 3m$ as follows. For the i'th bit in I, there is a *phase* p_i in $\sigma(I)$ that is formed by three squares, all of side length 2, as we explain. Regardless of the bit i being '0' or '1' in I, p_i starts with a square with the bottom-left

corner at point $(6i+2, 1)$ and of side length 2. Now, depending on the bit value, the other two requests in the phase of i differ:

- If the bit is a '1', the other two squares in the phase are I) a square with bottom-left corner at $(6i+1, 0)$, II) a square with bottom-left corner at $(6i+3, 2)$.
- If the bit is a '0', the other two squares in the phase are I) a square with bottom-left corner at $(6i+3, 2)$, II) a square with bottom-left corner at $(6i+4, 3)$.

For example, if I starts with $\langle 1, 0, 1, 1, 0 \ldots, \rangle$, $\sigma(I)$ would start with:



Note that, when the first rectangle (associated with a bit '1') appears, an online algorithm has to decide whether to reject or accept it, and that translates to guessing the first bit I is 1 (in case of rejection) or 0 (in case of acceptance).

To be more formal, an online algorithm ALG with sublinear advice for the SQUARE SCHEDULING PROBLEM can be used to guess bits in an instance of the BINARY GUESSING PROBLEM as follows. As the first square in the i'th phase of $\sigma(I)$ appears, A either accepts it or not. If it accepts, guess the value of the i'th bit as "0" and if A rejects the first square, guess the i'th bit as "1". We show that, if ALG has a competitive ratio better than $4/3$, then it must be that more than half of the bits in I are guessed correctly, which contradicts the fact that no algorithm for the BINARY GUESSING PROBLEM with sublinear advice can guess more than half of the bits correctly.

An optimal algorithm for $\sigma(I)$ has a profit of exactly $2m$; this is because, regardless of the i'th bit of I, two squares from each phase can be accepted. An incorrect guess for the i'th bit by ALG, on the other hand, translates to accepting at most 1 square from the i'th phase of $\sigma(I)$. Guessing exactly half of the phases correctly, therefore, translates to a total profit of at most $\frac{m}{2} \times 2 + \frac{m}{2} \times 1 = 1.5m$ or a ratio of $\frac{2m}{1.5} = 4/3$ between the profit of an optimal solution and ALG. Given that ALG is claimed to have a competitive ratio strictly better than $4/3$, it must be that it has guessed a

fraction strictly more than 0.5 of the bits in I. This leads to a contradiction with the fact that ALG has only sublinear advice. □

Chapter 6

Interval Scheduling with Fairness

6.1 Overview and Summary of Results

In this chapter, we examine the Interval Scheduling problem from a fairness perspective. The notion of fairness we adopt aligns with the concepts of *group fairness* and *max-min fairness* reviewed in Section 1.5. Specifically, we assume that each interval belongs to a group and aim to ensure equity across these groups by maximizing the fairness ratio (See Defintioin 3.3). We analyze the problem under the *absolute* and the *asymptotic* settings. Recall that, in the asymptotic setting, it is assumed that there is an arbitrarily large number of non-overlapping intervals from each group. As we will see, there are significant distinctions between these two settings when studying algorithmic fairness.
Our contributions in this chapter are summarized as follows:

- In Section 6.2, we first study the *Interval Scheduling with Fairness* problem in the absolute setting:
  - We show that no algorithm can have a fairness ratio better than $1/k$ (Theorem 19). This holds even for randomized and offline algorithms. Given this result, we sometimes refer to an algorithm as simply “fair" iff its fairness ratio is $1/k$.
  - We establish that any deterministic algorithm has a fairness ratio of 0 and a competitive ratio of at least $T/2$ in the worst-case inputs (Theorem 20). This rules out deterministic algorithms as viable solutions for fair interval scheduling in the absolute setting.

    Offline Setting:
  - We establish a fairness-performance impossibility result that holds for randomized and offline algorithms. In particular, we show that if an algorithm is fair (has fairness ratio $1/k$), its approximation ratio cannot be better than k (Theorem 22).

- We propose a randomized offline algorithm, OFISAAS, which is fair (has fairness ratio $1/k$) and has an approximation ratio of k (Theorem 23). In light of Theorem 22, this algorithm achieves Pareto-optimality in terms of fairness and overall performance. That is, one cannot improve its fairness ratio and approximation factor simultaneously.

  Online Setting:

- In the online setting, we prove that a randomized online algorithm that is fair (has fairness ratio $1/k$) cannot achieve a competitive ratio better than $\log T$ (Theorem 24).
- We introduce an online algorithm, OLCFISAS, which achieves a competitive ratio of $O(k \log T)$, and a fairness ratio of $\frac{1}{3k \log T}$ (Theorem 25).

- In Section 6.3, we study the fair interval scheduling problem in the *asymptotic* setting, and propose the Block Partitioning Algorithm, an offline and deterministic algorithm which achieves a fairness ratio of $1/(3k)$ and a competitive ratio of $3k$ (See Theorem 26). This shows that it is more plausible to achieve fair solutions, even using deterministic algorithms, under the asymptotic setting.

6.2 Fair Interval Scheduling in Absolute Setting

In this section, we consider the fair interval scheduling problem in the absolute setting, i.e., when we make no assumption for the size of the optimal scheduling of individual groups

6.2.1 Impossibility Results for All Algorithms

We begin with the following observation, which indicates that no algorithm can have a fairness ratio better than $1/k$.

Theorem 19. *Let k be the number of groups. No (randomized or deterministic) algorithm can have a fairness ratio larger than $1/k$.*

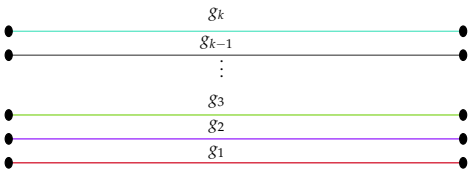
Proof. Consider an instance where each of the k groups contributes exactly one interval, and all k intervals have identical start and end times. Since the intervals fully overlap, any feasible algorithm can select at most one of them. See figure below for an illustration.

Suppose a randomized algorithm ALG selects an interval according to some probability distribution over the groups. Since only one interval can be accepted, choosing an interval from one group necessarily excludes all others. In the worst case,

the algorithm assigns probability $\frac{1}{k}$ to each group, resulting in an expected selection of $\frac{1}{k}$ for each group. Therefore, the group with the minimum expected selection also receives at most $\frac{1}{k}$, yielding a fairness ratio of

$$\min_{i \in [k]} \frac{E[\text{ALG}_{g_i}]}{\text{OPT}_{g_i}} = \frac{1/k}{1} = \frac{1}{k}.$$

Any attempt to increase the expected selection probability for one group necessarily reduces it for others, due to the mutual exclusivity of interval acceptance. Thus, no randomized algorithm can achieve a strictly better ratio in this instance, and the bound is tight. □



In light of the above observation, we sometimes refer to an algorithm simply as a *fair algorithm* iff its fairness ratio is $1/k$.

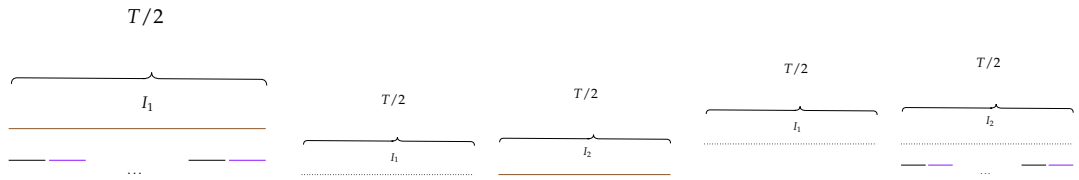
6.2.2 Impossibility Results for Deterministic Algorithms

In light of the above observation, we focus on the fairness ratio in the remainder of this section. The following theorem demonstrates that deterministic algorithms cannot achieve $1/k$ fairness ratio.

Theorem 20. *Any deterministic algorithm for interval scheduling in the absolute setting has a competitive ratio of at least $T/2$ and a fairness ratio of 0. This holds even for the restricted case of $k = 2$ groups.*

Proof. Fix a deterministic algorithm ALG. Consider an adversarial setting, where the adversary sends a long interval $I_1 = [1, T/2)$ of group 1.

- If ALG accepts I_1, the input follows with $\frac{T}{2} - 2$ non-intersecting unit intervals in $[1, \frac{T}{2})$, half of each belong to group 1 and half to group 2. The benefit of ALG is 1, and it accepts 0 interval from group 2. On the other hand, OPT accepts all unit intervals, that is $T/2$ intervals, out of which $\frac{T}{4} - 1$ belong to group 1



(A) ALG accepts I_1, the input is followed by $T/2$ unit intervals from each group.

(B) ALG rejects I_1 and accepts I_2

(C) ALG rejects both I_1 and I_2, and the input is followed by $T/2$ unit intervals from each group.

FIGURE 6.1: Illustration of the three possible scenarios.

and $\frac{T}{4} - 1$ to group 2. The competitive ratio of ALG is thus $\frac{T/2-2}{1} \approx \frac{T}{2}$ and its fairness ratio (realized by group 2) is $\frac{0}{T/4} = 0$. See Figure 6.1a.

- If ALG rejects I_1, the input continues with an interval $I_2 = [T/2, T)$ of group 2.
  - If ALG rejects I_2, the input stops. The benefit of ALG is 0. OPT accepts both I_1, I_2; it has a benefit of 2, and it accepts one interval from each group. The competitive ratio of ALG is thus unbounded, and its fairness ratio (realized by both groups) is $\frac{0}{1} = 0$. See in Figure 6.1b.
  - If ALG accepts I_2, the argument is symmetric to the case when it accepts I_1: the input follows with $\frac{T}{2} - 2$ non-intersecting unit intervals in $[T/2, T)$, half of each belong to group 1 and half to group 2. The benefit of ALG is 1, and it accepts 0 interval from group 1. On the other hand, OPT accepts all unit intervals, that is $\frac{T}{2} - 2$ intervals, out of which $\frac{T}{4} - 1$ belong to group 1 and $\frac{T}{4} - 1$ to group 2. The competitive ratio of ALG is thus $\frac{T/2-2}{1} \approx \frac{T}{2}$ and its fairness ratio (realized by group 1) is $\frac{0}{T/4} = 0$. See in Figure 6.1c.

□

In light of the above theorem, we will focus on randomized algorithms in the remainder of this section.

6.2.3 Offline Interval Scheduling In Absolute Setting

The offline version of the problem is non-trivial in the fairness setting. In particular, the optimal offline greedy algorithm (that maximizes the number of accepted intervals) does not provide a fairness guarantee (Observation 21). In what follows, we first show that any deterministic algorithm ALG is either not fair (has a fairness

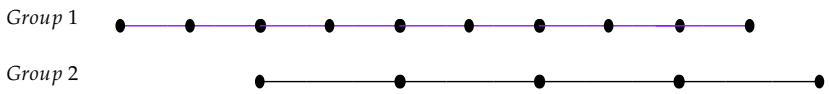
ratio less than $1/k$) or it has a competitive ratio of at least k (Theorem 22). Moreover, we introduce an offline algorithm that is k-competitive and has a fairness ratio of $1/k$ (Theorem 23), and thus is Pareto-optimal with respect to competitiveness and fairness.

Negative Results

Recall that there is an offline greedy algorithm for interval scheduling that sorts intervals by their endpoints and accepts them greedily in this order. While this algorithm is optimal when all intervals belong to one group, it is easy to observe that it is not fair even if there are only $k = 2$ groups:

Observation 21. *The offline algorithm that sorts intervals by their right endpoint and accepts them greedily in that order has a fairness ratio of 0.*

Proof. Consider an input formed by $\lfloor 3n/2 \rfloor$ intervals, specifically n intervals of the form $[i, i+1)$ that belong to group 1 and $\lfloor n/2 \rfloor$ intervals in the form $[2i, 2i+2)$ that belong to group 2, as shown below.



The Greedy algorithm picks all the group 1 intervals and none from group 2. Given that no two intervals of group 2 intersect, the fairness ratio of the greedy algorithm, realized by group 2, is $\frac{0}{n/2} = 0$ □

Next, we show that achieving an optimal fairness ratio of $1/k$ forces an approximation ratio of k or worse for any offline algorithm.

Theorem 22. *Consider the interval scheduling problem in the absolute setting with k groups. For any offline algorithm* ALG*, at least one of the following holds:*

- ALG *is not fair, i.e., it has a fairness ratio of at most* $1/k$*; or*
- ALG *has a competitive ratio of at least k (i.e., the ratio between the number of accepted intervals by* ALG *and that of* OPT *could be as large as k).*

Proof. Let ALG be a fair algorithm with a fairness ratio of $1/k$. We show that the competitive ratio of ALG is at least k.

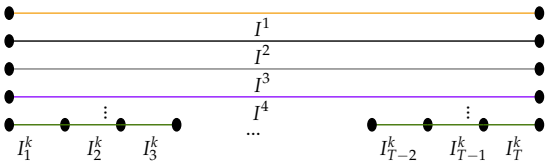


FIGURE 6.2: Instance illustrating the fairness–competitiveness trade-off with $k-1$ long intervals and T short disjoint ones.

Consider an instance with intervals $(I^1, I^2, \ldots, I^{k-2}, I^{k-1}, I_1^k, I_2^k, \ldots, I_T^k)$, where for each $j \in [1, k-1]$, the interval I^j belongs to group j and we have $I^j = [1, T)$. Moreover, for each $i \in [1, T]$, $I_i^k = [i, i+1)$ belongs to group k (See Figure 6.2). All intervals from group k are disjoint and lie strictly inside each long interval from g_j ($j \neq k$), so no feasible schedule can include both types.

Given that there is at least one interval from each group, we have $\text{OPT}_j \geq 1$ for any group j. Thus, to have a fairness ratio of $\frac{1}{k}$ for ALG, we should have $E_g(A) \geq 1/k$ for each group g. In other words, ALG must accept each of the long intervals of group $g \in [1, k-1]$ with a chance of at least $1/k$. This means the chance of accepting intervals from group k cannot be more than $1/k$. Therefore, the probability of accepting each group $g \in [1, k]$ is $1/k$, and for the expected benefit of the algorithm, we can write

$$E(A) = \frac{1}{k}(k-1) + \frac{1}{k} \cdot T < \frac{T}{k} + 1$$

Given that the benefit of OPT is T for the entire input (realized by accepting unit intervals of group k), we can write

$$\frac{\text{OPT}}{E(A)} \geq \frac{T}{T/k+1},$$

which approaches to k for large values of T. □

Positive Results

In this section, we will explain an offline algorithm that has a fairness ratio of $1/k$ and also achieves an approximation ratio of k. In light of Theorem 22, this algorithm would be Pareto-optimal, i.e., it achieves the best possible competitive ratio among fair algorithms (algorithms with a fairness ratio of $1/k$).

Offline Fair Interval Scheduling Algorithm under Absolute Setting (OFISAAS)

Input: A set of intervals $I = I^1 \cup I^2 \cup \ldots \cup I^k$, where $I^g = \{I_1^g, I_2^g, \ldots, I_{n_g}^g\}$ is the set of intervals of group g.

Output: A subset of intervals from I that are mutually non-overlapping.

Random Selection: Select a g uniformly at random from $[1, k]$.

Sorting: Sort intervals in I^g by their right endpoint.

Greedy Approach: Process intervals in the sorted order of I^g and accept each if and only if it does not intersect previously accepted intervals.

Theorem 23. OFISAAS *is a randomized algorithm for offline interval scheduling that achieves a fairness ratio of* $1/k$*, and its approximation ratio is* k*.*

Proof. We prove the claimed bounds for competitiveness and fairness as follows.

- **Competitiveness:** Let ALG denote the expected benefit of OFISAAS. Let OPT_g denote the benefit of an optimal schedule for I^g. Then we have

$$\text{ALG} = \frac{1}{k} \cdot \sum_{g=1}^{k} \text{OPT}_g. \tag{6.1}$$

  Let O_g be the number of intervals from group g in the optimal schedule of I. Then we have

$$\text{OPT} = \sum_{g=1}^{k} O_g \leq \sum_{g=1}^{k} \text{OPT}_g \leq k \cdot \text{ALG}$$

  The first equality holds due to the definition of O_g, the second inequality holds because we know O_g indicates a scheduling of I^g (which is feasible but not necessarily optimal), and the last inequality holds from 6.1.

- **Fairness:** OFISAAS selects every group g with a chance of $1/k$, and if a group g is selected, OFISAAS schedules members of g in an optimal way. Therefore, the expected number of intervals from any group g accepted by OFISAAS is $\frac{1}{k} \times \text{OPT}_j$. This establishes a fairness ratio of $1/k$ for OFISAAS.

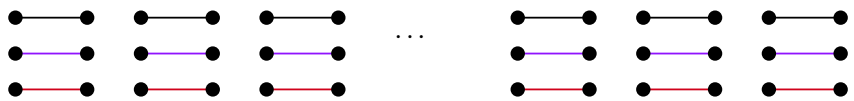
□

Further Improvements of OFISAAS We apply two augmentations to OFISAAS to further improve its typical performance (i.e., its performance in scenarios that go beyond worst-case scenarios). Our augmentations are twofold:

- First, if possible, we partition the input sequence into disjoint *blocks* $b_1, b_2, \ldots, b_m$, such that no interval in b_i intersects a block in $b_{i'}$ for $i \neq i'$. Then, we treat each block separately from the others.
- Second, instead of picking a single group, for each block, we select a random ordering of groups, which is a permutation $(\pi_q^1, \pi_q^2, \ldots, \pi_q^m)$ for block b_q. We then process the group sequentially according to the chosen permutation π_q^c. For each group g in order, we greedily accept intervals from group g based on their right endpoints.

We emphasize that the above augmentations do not improve the worst-case performance of OFISAAS. However, the first augmentation is expected to enhance the fairness of the algorithm in typical scenarios (beyond worst case), and the second is expected to improve its overall performance (i.e., the number of accepted intervals), as explained below.

Example 6.2.1. *Consider a setting where we have input* $(I_1^1, I_1^2, \ldots, I_1^k, I_1^2, I_2^2, \ldots, I_2^k, \ldots, I_m^1, I_m^2, \ldots I_m^k)$, *where for any* $i \in [1, m]$ *and* $j \in [1, k]$ *intervals of the form* I_i^j *are in the form* $[2i, 2i + 1)$ *and all belong to group* g_j. *In other words, we have disjoint blocks of intervals, where there is exactly one interval from each group in each block. See the figure below.*



Now, if we do not partition the input by blocks, our algorithm only selects intervals of the same group. The results are fair *in the sense that each group has the same chance of being selected, but it is not* "ex-post fair" *[34] in the sense that its solutions are never representative of all groups. In contrast, via block partitioning, we can ensure that each block is treated separately; thus, with some probability that intervals from each group are included in the final solutions.*

Example 6.2.2. *Consider a simple input* $(I_1, I_2, \ldots, I_m)$, *where the* $I_i = [i, i + 1)$ *(no two intervals intersect), and the groups of intervals change in a round-robin fashion*

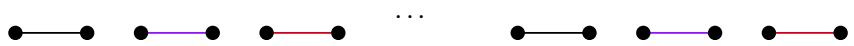


FIGURE 6.3: Round-Robin Grouping of Disjoint Intervals

$1, 2, \ldots, k, 1, \ldots, k$. *(See Figure 6.3). If the algorithm selects a group G_j uniformly at random and only accepts the intervals from such a group. The results are* fair *in the sense that each group has the same chance of being selected, but it is not* maximal *in the sense that many intervals remain unselected. Instead, draw a random permutation of the groups once at the beginning, and then process the intervals according to that order. This preserves the same expected fairness while producing a maximal solution which accepts all intervals.*

6.2.4 Online Interval Scheduling in the Absolute Setting

Next, we study online algorithms for the fairness ratio in the absolute setting.

Negative Result

Recall that no algorithm can achieve a fairness ratio better than $1/k$ (Theorem 19). The following theorem establishes an impossibility result for the trade-off between competitiveness and fairness in the online setting.

Theorem 24. *Consider the interval scheduling problem in the absolute setting with k groups. For any online algorithm* ALG*, at least one of the following holds:*

- ALG *is not fair, i.e., it has a fairness ratio of at most $1/k$; or*
- ALG *has a competitive ratio of at least $\Omega(\log T)$ (i.e., the ratio between the number of accepted intervals by* ALG *and that of* OPT *could be as large as k).*

Proof. Let ALG be a fair online algorithm with a fairness ratio of $1/k$. We show that the competitive ratio of ALG is at least $\log T$.

Consider an instance with intervals $\langle I^1, I^2, \ldots, I^{k-2}, I^{k-1}, \sigma^k \rangle$, where for each $j \in [1, k-1]$, the interval I^j belongs to group j and we have $I^j = [1, T)$, and σ^k is a sequence of intervals, all belonging to class k. The exact form of σ_i^k is defined stochastically, as we will clarify later.

Since ALG is fair, it must accept each of the first $k-1$ intervals with a chance of at least $1/k$. Therefore, the chance of not accepting any interval before σ_k arrives is at most $1 - (k-1) \times 1/k = 1/k$.

Consider the set of intervals $C_1^k \cup C_2^k \cup \ldots \cup C_{\log T}^k$. For each $i \in [1, \log T]$, define σ_i^k as the prefix sequence $\sigma_i^k = \langle C_1^k, C_2^k, \ldots, C_i^k \rangle$. We construct the set of intervals C_i by partitioning the interval $[1, T)$ into 2^{i-1} equal-length sub-intervals. Each interval in C_i has length $T/2^{i-1}$. Specifically, for each $x \in [1, 2^{i-1}]$, the interval I_x in class C_i is defined as $I_x = \left[\frac{(x-1)\cdot T}{2^{i-1}}, \frac{x \cdot T}{2^{i-1}}\right)$. Then, we will select σ_i^k with a probability of $\frac{T}{(T-1)2^i}$. The stochastic input generated for σ^k is similar to the one used in establishing a lower bound $\Omega(\log T)$ for online randomized algorithms [16]. For this input, the expected benefit of OPT for this stochastic input is greater than $\log T/2$. On the other hand, the expected benefit of ALG is $\frac{k-1}{k} \times 1$ (if one of the first $k-1$ intervals is accepted) plus at most $1/k \times 1 = 1/k$ (if it rejects all the first $k-1$ intervals). So, the expected benefit of ALG is 1. Therefore, the ratio between the benefit of OPT and ALG is at least $\frac{\log T/2}{2}$. Applying Yao's Minimax Principle (see 1), the competitive ratio cannot be better than $\frac{\log T/2}{2} = \Omega(\log T)$ □

Positive Result

Next, we provide a randomized, online algorithm for interval scheduling which achieves a fairness ratio of $\frac{1}{3k \log T}$. The algorithm is based on CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT and defined as follows.

Online Length-Classified Fair Interval Scheduling under Absolute Setting (OLCFISAS)

Input: An online Sequence $\langle I_1, I_2, \ldots, I_n \rangle$ of intervals, for some known T, where each interval I_i is an interval which belongs to some group $g \in G$.

Output: A subset of mutually non-overlapping intervals

Classification: Consider the following classification by length; we say an interval belongs to class c, if its length is in $[2^c, 2^{c+1})$. We have $c \in [0, \log T)$. Given that T bounds all lengths, the set $\mathcal{C}$ of all classes has $\log T$ classes in it.

Random Selection: Pick one value $c_\ell \in [0, \log T)$ uniformly at random in this range. Also, pick one group $g \in G$ uniformly at random among all groups in G.

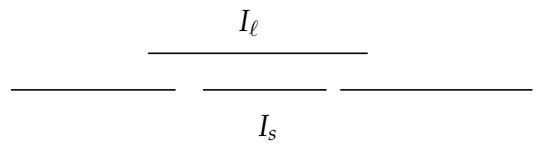
Online Process Suppose an interval I_j arrives. Accept I_j if and only if the following conditions are satisfied:

- I_j does not overlap with any previously accepted intervals;
- I_j belongs to class c_ℓ and group g.

Theorem 25 (Fairness and Competitiveness Guarantee)**.** *The* OLCFISAS *algorithm for fairness has a competitive ratio $O(k \log T)$ and a fairness ratio of $\frac{1}{3k \log T}$.*

Proof. We use ALG to refer to OLCFISAS. For a class $c \in \mathcal{C}$, let OPT_c denote the optimal scheduling of an input formed by intervals from class c. Let OPT^g denote the optimal scheduling of an input formed by all intervals within group g, and let OPT_c^g denote the optimal scheduling of an input formed by intervals of class c within group g.

We claim that if the algorithm selects a class c and group g, its benefit is at least $\text{OPT}_c^g/3$. This holds because the algorithm greedily accepts intervals of class c and group g, and accepting an interval of group g and class c can result in rejecting at most 3 intervals that follow it (because the length of the intervals belonging to class c differs by at most a factor of 2). In particular, any interval I_ℓ from class c has maximum length $2^{c+1} - 1$, it follows that at most one short interval I_s from OPT within class c can be contained within I_ℓ. Furthermore, the other two intervals from OPT within class c can intersect the endpoints of I_ℓ, with each covering one endpoint.



Thus, the benefit of applying the greedy approach to the intervals of class c within group g is at least $\text{OPT}_c^g/3$. Since there are a total of $\log T$ classes and k groups, and ALG selects one uniformly at random, the expected benefit of the algorithm for group g, denoted by ALG^g, satisfies the following:

$$E[\text{ALG}^g] = \frac{1}{k \log T} \sum_{c \in \mathcal{C}} \text{OPT}_c^g/3 \tag{6.2}$$

Let O_c^g denote the set of intervals selected by OPT^g within class c from group g. Therefore, we have: $\text{OPT}^g = \sum_{c \in \mathcal{C}} O_c^g$. Since O_c^g forms a valid (but possibly suboptimal) solution for the input restricted to group g with class c, we have $\text{OPT}_c^g \geq O_c^g$, where OPT_c^g is the optimal solution for group g restricted to class c.

Thus, summing up over all classes, for a fixed group g, we can write:

$$\sum_{c \in \mathcal{C}} \text{OPT}_c^g \geq \sum_{g \in G} O_c^g = \text{OPT}^g. \tag{6.3}$$

Combining Equation (6.2) and Equation (6.3), for the benefit of ALG for group g, we obtain:

$$E[\text{ALG}^g] = \frac{1}{3k \log T} \sum_{c \in \mathcal{C}} \text{OPT}_c^g \geq \frac{1}{3k \log T} \text{OPT}^g. \tag{6.4}$$

Similarly, let O^g be the set of intervals in the optimal solution from group g. So, we have $\text{OPT} = \sum_{g \in G} O^g$. Note that $O^g \leq \text{OPT}^g$, because O^g forms a valid but possibly suboptimal solution for the input intervals that belong to group g. Adding all groups together, for the benefit of OPT for group g, we can write:

$$\sum_{c \in \mathcal{C}} \text{OPT}^g \geq \sum_{g \in G} O^g = \text{OPT}. \tag{6.5}$$

Combining Equation (6.4) and Equation (6.5), we obtain:

$$E[\text{ALG}] = \sum_{g \in G} E[\text{ALG}^g] \geq \frac{1}{3k \log T} \sum_{g \in G} \text{OPT}^g \geq \frac{\text{OPT}}{3k \log T}.$$

Thus, CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT is $3k \log T$-competitive and $\frac{1}{3k \log T}$ fairness ratio. □

6.3 Fair Interval Scheduling in the Asymptotic Setting

In this section, we consider the asymptotic setting of the fair interval scheduling problem. We focus on the offline setting, and show there is a *deterministic* algorithm with a fairness ratio of $1/(3k)$ and a competitive ratio of $3k$. Interestingly, this is independent of T.

Let OPT_g denote the intervals in an optimal solution formed only by intervals of group g. Note OPT_g can be easily formed by applying the Greedy approach on intervals of group g (by sorting them in the non-decreasing order of their right-endpoint and accepting them in a greedy manner in that order).

Our algorithm works by partitioning optimal solutions into “blocks" and is thus referred to as the Block Partitioning Algorithm (BPA). In particular, it partitions OPT_g

into $3k$ blocks, which forms a set $\mathbf{B}_g$, such that each block in $\mathbf{B}_g$ contains $\text{OPT}_g/(3k)$ intervals from OPT_g (since OPT_g is asymptotically large, it is safe to assume it is divisible by $3k$). Define the *span* of the block as the difference between the maximum and minimum endpoints of intervals in that block. Moreover, define the *min-block-span* of group g as the minimum span across all blocks of group g.

Now, take the group $g_{\min}$ with the smallest min-block-span and let $B_{\min}$ be the block of $g_{\min}$ with the smallest span. BPA accepts all intervals of group $g_{\min}$ that being to class $B_{\min}$ and reject all other intervals of group $g_{\min}$. Also, it rejects all intervals in all blocks from other groups that intersect $B_{\min}$. This is repeated until a block from all groups is accepted.

To be more precise, the algorithm can be formally described as follows:

Block Partitioning Algorithm (BPA)

Input: A set of intervals $I = I^1 \cup I^2 \cup \ldots \cup I^k$, where $I^g = \{I_1^g, I_2^g, \ldots, I_{n_g}^g\}$ is the set of intervals of group $g \in G$ such that $\text{OPT}(I^g) \to \infty$.

Output: A subset of intervals from I that are mutually non-overlapping.

Optimal Formation: For any group $g \in G$:

- Form the optimal schedule OPT_g for J^g (using the greedy method)
- Partition OPT_g into a set $\mathbf{B}_g$ formed by $3k$ blocks such that each block contains $\text{OPT}_g/(3k)$

Block Choosing: Let $H \leftarrow G$ and repeat k times (where $k = |G|$):

- Let $g_{\min} \in H$ be the group with minimum min-block-span and $B_{\min}$ be the block of $\text{OPT}_{g_{\min}}$ with minimum span.
- Add intervals in $B_{\min}$ to Ac.
- Remove $g_{\min}$ from H
- For any $g' \neq g_{\min}$, remove all blocks that intersect $B_{\min}$ form $\mathbf{B}_{g'}$.

Example 6.3.1. *To illustrated how* BPA *works, consider the example of Figure 6.4, where* $G = \{g_1, g_2, g_3\}$ *and there are* $k = 3$ *groups. The optimal schedule* OPT_g *for every group* g *is depicted. Note that the number of intervals is these optimal schedules is assumed to be asymptotically large. Given that* $k = 3$*, these schedules are partitioned into* $3k = 9$ *blocks. Among the blocks, the block* $B_{\min}$ *with the minimum span is highlighted (here it belongs to group* $g_{\min} = g_2$*). Note that, since* $B_{\min}$ *has minimum span, it intersects at most two blocks from any other group.* BPA *accepts all intervals of group* g_2 *from* $B_{\min}$ *and rejects all other intervals of group* g_2*. It also removes the blocks of the groups* g_1 *and* g_3 *that intersect* $B_{\min}$ *(the first block of* g_1 *and the first two blocks of* g_2*). The removed blocks are indicated with a cross.*

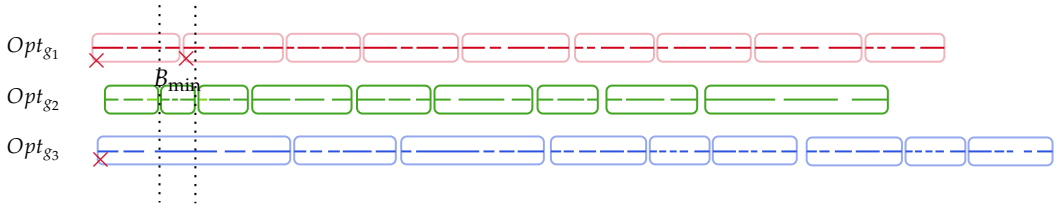


FIGURE 6.4: Example of BPA with $k = 3$

Theorem 26. BPA *is a deterministic algorithm that achieves an approximation ratio of at most* $3k$ *and a fairness ratio of* $1/(3k)$ *in the asymptotic setting of the offline interval scheduling problem.*

Proof. We use ALG to refer to BPA. We claim that the number of accepted intervals from any group g is at least $\text{OPT}_g/(3k)$. If this claim holds, the fairness ratio of ALG would be at most $1/(3k)$ (by definition of fairness ratio). For the approximation ratio, for the benefit of ALG, we can write $\text{ALG} = \sum_{g \in G} \text{OPT}_g/(3k) = \frac{1}{3k} \sum_{g \in G} \text{OPT}_g$. Moreover, if O_g denotes the number of accepted intervals from group g in an optimal solution (for the entire input), we have $\text{OPT} = \sum_g O_g \leq \sum_g \text{OPT}_g = 3k\text{ALG}$, which establishes the claimed upper bound for competitive ratio.

It remains to prove the claim. That is, the number of accepted intervals from group g is at least $\text{OPT}_g/(3k)$. For that, we show that at least k blocks remain at each stage in the algorithm for each group g, and thus, at some point it will be selected as $g_{\min}$ in which case one of its k remaining blocks will be selected as $B_{\min}$ (and thus the algorithm will accept $\text{OPT}_g/(3k)$ intervals in such a block). To see why at least k blocks remain at each stage for G, suppose we are in the i'th iteration of the algorithm, that is, previously $i-1$ groups have been selected as $g_{\min}$. Each of these groups has removed at most two blocks from $\mathbf{B}_g$. Since $i < k$ and $\mathbf{B}_g$ has initially $3k$ blocks in it, at the i'th iteration, there are $3k - 2(i-1) \geq k$ blocks in $\mathbf{B}_g$.

Therefore, every group g eventually gets a block, and we have $\text{ALG}_g \geq \text{OPT}_g/(3k)$. Summing over all groups, we obtain $\text{ALG} \geq \text{OPT}/(3k)$. □

Chapter 7

Discussion and Future Work

In this chapter, we will review our contributions in this thesis and discuss directions for future work.

7.1 Purely Online Rectangle Scheduling

For the purely online square and rectangle scheduling, no deterministic algorithm can have a competitive ratio that is sublinear in T (Proposition 10). Therefore, we studied randomized algorithms for the rectangle scheduling problem. In particular, we established a tight competitive ratio of $\Theta(\log^2 T)$ for rectangle scheduling (Theorem 11 and Theorem 12) .

We believe our ideas for using CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT in a 2D setting can extend to higher dimensions for scheduling hyper-cubes and hyper-rectangles in d dimensions. We conjecture that, for a constant d, the best competitive ratio for rectangle scheduling problem is $\Theta(\log^d T)$. Establishing this conjecture is a promising direction for future work:

Question 27. *What is the best attainable competitive ratio for d-dimensional rectangle scheduling?*

7.2 Rectangle Scheduling with Prediction

We considered learning-augmented algorithms that receive a set of rectangles that are predicted to appear in the input sequence. For an error measure γ, we showed that no algorithm can achieve a competitive ratio better than $\frac{1}{1-\gamma}$ (Theorem 13). We also showed that 2DTRUST, an algorithm that follows the prediction, has a competitive ratio of exactly $\frac{1}{1-2\gamma}$ (Theorem 14 and Theorem 15). We further showed how to robustify 2DTRUST by combining it with 2D-CLASSIFY-BY-LENGTH-AND-RANDOMLY-SELECT (Section 16).

Note that there is a gap between the competitive ratio $\frac{1}{1-2\gamma}$ of 2DTRUST and the lower bound $\frac{1}{1-\gamma}$ we have established. For interval scheduling, Boyar et al [20] close this gap with a modification of 2DTRUST, named TRUSTGREEDY. Indeed, TRUSTGREEDY can be extended to two dimensions as follows.

TRUSTGREEDYTWO Algorithm for Rectangle Scheduling with Prediction

Input. A predicted set of rectangles $\hat{R} = \{\hat{R}_1, \hat{R}_2, \ldots, \hat{R}_n\}$ and an online sequence $R = \langle R_1, R_2, \ldots, R_n \rangle$.

Output. A subset of rectangles that are mutually non-overlapping.

Planning. Form an optimal scheduled $\hat{R}^*$ as the optimal solution for scheduling $\hat{R}$

Online Process. For each arriving rectangle $r \in R$, do the following.

- If $r \in \hat{R}^*$, accept it.
- If $r \notin \hat{R}^*$, then:
  - Accept r if it does not intersect any previously accepted rectangle AND it intersects at most one rectangle r' from $\hat{R}^*$.
  - Otherwise, reject r.

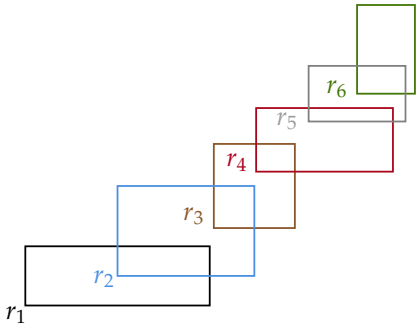
Unfortunately, the techniques used in [20] for interval scheduling cannot be used to prove a tight bound for higher dimensions. This is because their algorithm and analysis are heavily dependent on defining an ordering between intervals and a notion of "dominance" that cannot be extended to higher dimensions. We believe the above algorithm has a competitive ratio of $\frac{1}{1-\gamma}$ for special settings of RECTANGLE SCHEDULING PROBLEM, like when the offline set of rectangles resembles a "staircase". That is, we can define a dominating order for rectangles defined as follows.

Definition 28 (Dominating Order). *[2] A sequence of rectangles $\langle R_1, R_2, \ldots, R_n \rangle$ is said to be in* dominating order *if*

$$R_1 \prec R_2 \prec \cdots \prec R_n,$$

where $R_i \prec R_{i+1}$ indicates that R_i is dominated by R_{i+1}; that is, the top-right corner of R_i is coordinate-wise less than or equal to that of R_{i+1}.

For example, the following is a sequence $\langle R_1, R_2, R_3, R_4, R_5, R_6 \rangle$ in dominating order:



Conjecture 29. *The* TRUSTGREEDYTWO *has a competitive ratio of* $\frac{1}{1-\gamma}$ *for* RECTANGLE SCHEDULING PROBLEM *when the input has staircase shape (for inputs whose set of rectangles admits a dominating order).*

For the general setting, however, the problem remains open.

Question 30. *Is the competitive ratio of* TRUSTGREEDYTWO $\frac{1}{1-\gamma}$ *for general settings of the* RECTANGLE SCHEDULING PROBLEM *problem? If not, is there any algorithm with a competitive ratio of* $\frac{1}{1-\gamma}$*?*

7.3 Weighted Interval & Rectangle Scheduling

Throughout the thesis, we assume intervals or rectangles are unweighted and focus on maximizing the *number* of accepted intervals/rectangles. A natural extension of this work is to study the weighted setting of these problems.

In the online rectangle scheduling problem, rectangles appear online, each with a weight, and the objective is to accept a set of rectangles with the maximum total weight. We believe our results in Chapter 4 can be extended to the weighted setting.

Question 31. *What is the best attainable competitive ratio for the online rectangle (or square) scheduling problem in the weighted setting?*

Similarly, we may consider a weighted version of the interval scheduling problem in the fair setting. Again, instead of defining fairness based on the number of accepted intervals from each group, we aim to achieve a "fair" solution based on the total weight of intervals accepted from each group

Question 32. *What fairness guarantees can be provided for the* weighted *interval scheduling problem under offline/online and absolute/asymptotic settings with deterministic/randomized algorithms?*

.

7.4 Ex-Post Fairness for Interval Scheduling

Although the fairness ratio considers the expected share each group receives for a randomized algorithm, it does not ensure fairness in every group's final solution. For example, some of our randomized algorithms are based on selecting a group uniformly at random and accepting only intervals from that group. While such an algorithm is fair in *anticipation*, its final outcome (after a schedule is formed) would not seem fair ex post. We propose another fairness notion for interval scheduling, named POST FAIRNESS RATIO. The purpose of this notion is to ensure that, with high probability, *every* group receives a reasonable share of its potential optimal schedule.

Definition 33 ((δ, β)-ex-post fairness)**.** *An algorithm is said to be (δ, β)-ex-post-fair if and only if for any output solution S it generates for σ and any confidence parameter $\delta > 0$, with a chance of at least $1 - \delta$, it holds that $\min_{g \in G} \frac{S_g(\sigma)}{\text{OPT}_g(\sigma)} \geq \beta$, where $S_g(\sigma)$ is the number of intervals of σ present in S from group $g \in G$.*

In other words, δ is a confidence parameter: with a confidence of $1 - \delta$, we can guarantee that each group receives a share at least β of its optimal solution. When δ is arbitrarily close to 0, we say that the algorithm has an ex-post fairness ratio of β.

Ex-post fairness is most relevant in the asymptotic setting, where there is a guarantee on the minimum number of overlapping intervals from each group. Note that for the offline, asymptotic setting, we can achieve fairness via deterministic algorithms (a stronger notion of fairness), as described in Section 6. In the online setting, we believe that a randomized algorithm that partitions the horizon $[1, T]$ into equal-length windows and assigns each window to a group selected uniformly at random may achieve ex-post fairness. The algorithm is described below:

Window Partitioning Algorithm (WPA) with parameter w

Input: An Online Sequence of intervals $\sigma = \langle I_1, I_2, \ldots, I_n \rangle$, where each interval I_i belongs to a group $g(i) \in G$. We use I^g to denote the set of intervals of group $g \in G$ in σ and assume $\text{OPT}(I^g) \to \infty$.

Output: A subset of intervals from σ that are mutually non-overlapping.

Partition: Partition the horizon $[1, T]$ into T/w windows of equal length w; that is, the j'th window, w_j, covers $[jw + 1, (j + 1)w)$ for $j \in [0, T/w)$.

Random Selection For each window w_j, choose a group $g_j \in [1, k]$ uniformly at random and also a class $c_j \in [1, \log w]$ uniformly at random.

Online Process Upon the arrival of the i-th interval I_i, let $g(i)$ denote the group of I_i. Define the class $c(i)$ such that the length of I_i satisfies $\ell_i \in [2^c, 2^{c+1})$ for some $c \in [1, \log T]$. Accept I_i if and only if the following hold:

- I_i is entirely inside a window w_j
- I_i belongs to group g_j, i.e., $g(i) = g_j$.
- I_i belongs to class c_j, i.e., $c(i) = c_j$.

Question 34. *What ex-post fairness ratio and competitive ratio can be achieved by any randomized online algorithm? What are these ratios for the* WPA *algorithm?*

7.5 Fair Rectangle/Square Scheduling

We initiated the study of fairness in designing interval scheduling algorithms. It is natural to extend this to rectangular and square scheduling, when rectangles each have a group, and we want to maximize the fairness ratio (defined analogously to interval scheduling). Again, we believe many of our algorithmic and impossibility results for fair interval scheduling naturally extend to rectangle and square scheduling.

Question 35. *What fairness guarantees can be provided for the online rectangle (or Square) scheduling problem under offline/online and absolute/asymptotic settings with deterministic/randomized algorithms?*

Bibliography

[1] Mohsen Abbasi, Aditya Bhaskara, and Suresh Venkatasubramanian. "Fair clustering via equitable group representations". In: *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 2021, pp. 504–514.

[2] Rishi Advani and Abolfazl Asudeh. "Online Maximum Independent Set of Hyperrectangles". In: *arXiv preprint arXiv:2307.13261* (2023).

[3] Antonios Antoniadis, Hajo Broersma, and Yang Meng. "Online graph coloring with predictions". In: *International Symposium on Combinatorial Optimization*. Springer. 2024, pp. 289–302.

[4] Antonios Antoniadis et al. "Secretary and Online Matching Problems with Machine Learned Advice". In: *Proc. NeurIPS*. 2020.

[5] Makis Arsenis and Robert Kleinberg. "Individual fairness in prophet inequalities". In: *arXiv preprint arXiv:2205.10302* (2022).

[6] Baruch Awerbuch et al. "Competitive Non-Preemptive Call Control." In: *SODA*. Vol. 94. Citeseer. 1994, pp. 312–320.

[7] Xiaohui Bei et al. "Cake Cutting: Envy and Truth." In: *IJCAI*. 2017, pp. 3625–3631.

[8] Nawal Benabbou et al. "Finding fair and efficient allocations when valuations don't add up". In: *International Symposium on Algorithmic Game Theory*. Springer. 2020, pp. 32–46.

[9] Ziyad Benomar and Vianney Perchet. "Non-clairvoyant scheduling with partial predictions". In: *arXiv preprint arXiv:2405.01013* (2024).

[10] Ziyad Benomar et al. “Pareto-Optimality, Smoothness, and Stochasticity in Learning-Augmented One-Max-Search”. In: *arXiv preprint arXiv:2502.05720* (2025).

[11] Jon Louis Bentley et al. “A locally adaptive data compression scheme”. In: *Communications of the ACM* 29.4 (1986), pp. 320–330.

[12] Suman Bera et al. “Fair algorithms for clustering”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[13] Arghya Bhattacharya and Rathish Das. “Machine learning advised ski rental problem with a discount”. In: *International Conference and Workshops on Algorithms and Computation*. Springer. 2022, pp. 213–224.

[14] Hans-Joachim Böckenhauer et al. “Online Simple Knapsack with Reservation Costs”. In: *38th International Symposium on Theoretical Aspects of Computer Science (STACS)*. Vol. 187. 2021, 16:1–16:18.

[15] Hans-Joachim Böckenhauer et al. “The string guessing problem as a method to prove lower bounds on the advice complexity”. In: *Theoretical Computer Science* 554 (2014), pp. 95–108.

[16] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

[17] Allan Borodin and Christodoulos Karavasilis. “Any-order online interval selection”. In: *International Workshop on Approximation and Online Algorithms*. Springer. 2023, pp. 175–189.

[18] Allan Borodin and Christodoulos Karavasilis. “Random-Order Interval Selection”. In: *arXiv preprint arXiv:2407.20941* (2024).

[19] Joan Boyar et al. “Online algorithms with advice: A survey”. In: *ACM Computing Surveys* 50.2 (2017), pp. 1–34.

[20] Joan Boyar et al. “Online interval scheduling with predictions”. In: *Algorithms and Data Structures Symposium*. Springer. 2023, pp. 193–207.

[21] Steven J Brams and Alan D Taylor. "An envy-free cake division protocol". In: *The American Mathematical Monthly* 102.1 (1995), pp. 9–18.

[22] Niv Buchbinder, Kamal Jain, and Mohit Singh. "Secretary problems via linear programming". In: *Mathematics of Operations Research* 39.1 (2014), pp. 190–206.

[23] Parinya Chalermsook and Julia Chuzhoy. "Maximum independent set of rectangles". In: *Proceedings of the twentieth annual ACM-SIAM symposium on discrete algorithms*. SIAM. 2009, pp. 892–901.

[24] Anshuman Chhabra, Karina Masalkovaitė, and Prasant Mohapatra. "An overview of fairness in clustering". In: *IEEE Access* 9 (2021), pp. 130698–130720.

[25] Marek Chrobak et al. "Online scheduling of equal-length jobs: Randomization and restarts help". In: *SIAM Journal on Computing* 36.6 (2007), pp. 1709–1728.

[26] Edward G. Coffman et al. "Bin Packing Approximation Algorithms: Survey and Classification". In: *Handbook of Combinatorial Optimization*. Springer, 2013, pp. 455–531.

[27] John Dickerson et al. "The computational rise and fall of fairness". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 28. 1. 2014.

[28] Sharmila Duppala et al. "Proportionally Fair Matching via Randomized Rounding". In: *CoRR* abs/2412.11238 (2024).

[29] Christoph Dürr, Łukasz Jeż, and Kim Thang Nguyen. "Online scheduling of bounded length jobs to maximize throughput". In: *Approximation and Online Algorithms: 7th International Workshop, WAOA 2009, Copenhagen Denmark, September 10-11, 2009. Revised Papers 7*. Springer. 2010, pp. 116–127.

[30] Yuval Emek et al. "Online computation with advice". In: *Theoretical Computer Science* 412.24 (2011), pp. 2642–2656.

[31] Ulrich Faigle and Willem M Nawijn. "Note on scheduling intervals on-line". In: *Discrete Applied Mathematics* 58.1 (1995), pp. 13–17.

[32] Thomas S Ferguson. "Who solved the secretary problem?" In: *Statistical science* 4.3 (1989), pp. 282–289.

[33] Till Fluschnik et al. “Fair knapsack”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 1941–1948.

[34] Rupert Freeman, Nisarg Shah, and Rohit Vaish. “Best of both worlds: Ex-ante and ex-post fairness in resource allocation”. In: *Proceedings of the 21st ACM Conference on Economics and Computation*. 2020, pp. 21–22.

[35] Kaito Fujii and Yuichi Yoshida. “The secretary problem with predictions”. In: *Mathematics of operations research* 49.2 (2024), pp. 1241–1262.

[36] Stanley PY Fung. “Online preemptive scheduling with immediate decision or notification and penalties”. In: *Computing and Combinatorics: 16th Annual International Conference, COCOON 2010, Nha Trang, Vietnam, July 19-21, 2010. Proceedings 16*. Springer. 2010, pp. 389–398.

[37] Stanley PY Fung, Chung Keung Poon, and Feifeng Zheng. “Improved randomized online scheduling of intervals and jobs”. In: *Theory of Computing Systems* 55 (2014), pp. 202–228.

[38] Waldo Gálvez et al. “A 3-approximation algorithm for maximum independent set of rectangles”. In: *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2022, pp. 894–905.

[39] David García-Soriano and Francesco Bonchi. “Maxmin-fair ranking: individual fairness under group-fairness constraints”. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 436–446.

[40] Mohit Garg, Debajyoti Kar, and Arindam Khan. “Random-Order Online Independent Set of Intervals and Hyperrectangles”. In: *32nd Annual European Symposium on Algorithms (ESA 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2024.

[41] Sruthi Gorantla, Amit Deshpande, and Anand Louis. “Ranking for individual and group fairness simultaneously”. In: *arXiv preprint arXiv:2010.06986* (2020).

[42] Hadi Hosseini et al. “Class fairness in online matching”. In: *Artificial Intelligence* 335 (2024), p. 104177.

[43] Christodoulos Karavasilis. “Interval Selection with Binary Predictions”. In: *arXiv preprint arXiv:2502.10314* (2025).

[44] Anna R Karlin et al. “Competitive snoopy caching”. In: *Algorithmica* 3.1 (1988), pp. 79–119.

[45] Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. “An optimal algorithm for on-line bipartite matching”. In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 1990, pp. 352–358.

[46] Dmitriy Katz, Baruch Schieber, and Hadas Shachnai. “Brief announcement: Flexible resource allocation for clouds and all-optical networks”. In: *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*. 2016, pp. 225–226.

[47] Sanjeev Khanna, Shan Muthukrishnan, and Mike Paterson. “On approximating rectangle tiling and packing”. In: *SODA*. Vol. 98. 1998, pp. 384–393.

[48] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[49] Koji M Kobayashi. “Online interval scheduling to maximize total satisfaction”. In: *Theoretical Computer Science* 806 (2020), pp. 673–688.

[50] Rohan Kodialam. “Optimal Algorithms for Ski Rental with Soft Machine-Learned Predictions”. In: *ArXiv* (2019). arXiv:1903.00092 [cs.DS].

[51] Antoon WJ Kolen et al. “Interval scheduling: A survey”. In: *Naval Research Logistics (NRL)* 54.5 (2007), pp. 530–543.

[52] T. Lavastida et al. “Using Predicted Weights for Ad Delivery”. In: *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA)*. 2021, pp. 21–31.

[53] Adam Lechowicz et al. “Time fairness in online knapsack problems”. In: *arXiv preprint arXiv:2305.13293* (2023).

[54] Bo Li, Minming Li, and Ruilong Zhang. “Fair allocation with interval scheduling constraints”. In: *arXiv preprint arXiv:2107.11648* (2021).

[55] Bo Li, Chenhao Wang, and Ruilong Zhang. "A note on the online interval scheduling secretary problem". In: *Operations Research Letters* 50.1 (2022), pp. 72–75.

[56] Xiaoxiao Liang et al. "Online scheduling on a single machine with one restart for all jobs to minimize the weighted makespan". In: *AIMS Mathematics* 9.1 (2024), pp. 2518–2529.

[57] Richard J Lipton and Andrew Tomkins. "Online Interval Scheduling." In: *SODA*. Vol. 94. 1994, pp. 302–311.

[58] Joseph Mitchell. "Approximating maximum independent set for rectangles in the plane". In: *SIAM Journal on Computing* (2024), FOCS21–299.

[59] Hiroyuki Miyazawa and Thomas Erlebach. "An improved randomized on-line algorithm for a weighted interval selection problem". In: *Journal of Scheduling* 7.4 (2004), pp. 293–311.

[60] Dritan Nace and Michal Pióro. "Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial". In: *IEEE Communications surveys & tutorials* 10.4 (2009), pp. 5–17.

[61] Pooria Namyar et al. "Solving {Max-Min} Fair Resource Allocations Quickly on Large Graphs". In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2024, pp. 1937–1958.

[62] Joseph Naor, Hadas Shachnai, and Tami Tamir. "Real-time scheduling with a budget". In: *Algorithmica* 47 (2007), pp. 343–364.

[63] Deval Patel, Arindam Khan, and Anand Louis. "Group fairness for knapsack problems". In: *arXiv preprint arXiv:2006.07832* (2020).

[64] Benjamin Plaut and Tim Roughgarden. "Almost envy-freeness with general valuations". In: *SIAM Journal on Discrete Mathematics* 34.2 (2020), pp. 1039–1068.

[65] Bozidar Radunovic and Jean-Yves Le Boudec. "A unified framework for max-min and min-max fairness with applications". In: *IEEE/ACM Transactions on networking* 15.5 (2007), pp. 1073–1083.

[66] Vinod Raman and Ambuj Tewari. “Online Classification with Predictions”. In: *arXiv preprint arXiv:2405.14066* (2024).

[67] Yongho Shin et al. “Improved learning-augmented algorithms for the multi-option ski rental problem via best-possible competitive analysis”. In: *International Conference on Machine Learning (ICML)*. 2023, pp. 31539–31561.

[68] Daniel D Sleator and Robert E Tarjan. “Amortized efficiency of list update and paging rules”. In: *Communications of the ACM* 28.2 (1985), pp. 202–208.

[69] Unknown. “Report of the Washington Meeting, September 6-18, 1947”. In: *Econometrica* 16.1 (1948), pp. 33–111. ISSN: 00129682, 14680262. (Visited on 12/16/2024).

[70] Leslie G Valiant. “A theory of the learnable”. In: *Communications of the ACM* 27.11 (1984), pp. 1134–1142.

[71] Shufan Wang and Jian Li. “Online Algorithms for Multi-shop Ski Rental with Machine Learned Predictions”. In: *19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 2035–2037.

[72] Gerhard J Woeginger. “On-line scheduling of jobs with fixed start and end times”. In: *Theoretical Computer Science* 130.1 (1994), pp. 5–16.

[73] Meike Zehlike, Ke Yang, and Julia Stoyanovich. “Fairness in ranking: A survey”. In: *arXiv preprint arXiv:2103.14000* (2021).

[74] Yunhong Zhou, Deeparnab Chakrabarty, and Rajan M. Lukose. “Budget Constrained Bidding in Keyword Auctions and Online Knapsack Problems”. In: *Internet and Network Economics, 4th International Workshop, WINE 2008, Shanghai, China, December 17-20, 2008. Proceedings*. Vol. 5385. Lecture Notes in Computer Science. Springer, 2008, pp. 566–576.