ALGORITHMS FOR TIMELY BIN PACKING, FAIR TCP ACKNOWLEDGEMENT, AND VARIANTS

AIDA AMINIAN

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

AUGUST 2025

Abstract

The Timely Bin Packing problem [1] is a variant of bin packing, which incorporates time constraints for packing items. This is related to the classic Dynamic TCP Acknowledgement [2] problem, which has been extensively studied and has real-world applications. This thesis studies new algorithms and settings for these two related problems.

For Timely Bin Packing, we present deterministic and randomized algorithms that improve the best-known competitive ratios. We further provide impossibility results for different settings and variants of the problem. We also study the problem under certain assumptions, such as having restricted sizes and integer arrival times.

For Dynamic TCP Acknowledgement, we study relaxed settings that include machine-learned predictions. We consider the $f_{\max}$ objective [2], which balances the max-latency of acknowledged packets (thus conveying a notion of max-min fairness [3]). For this setting, we study learning-augmented algorithms and present experimental results on both synthetic and real-world data.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Shahin Kamali, for his clear guidance and unwavering support throughout my graduate studies. His expertise and patience have greatly influenced my academic development, and I am grateful for the opportunity to learn from him. I also thank Dr. Jeff Edmonds for his constructive comments that improved both the writing and the results. I appreciate Dr. Mohamed Omar's suggestions, which helped refine the final version of this thesis. Finally, I would like to thank my family and friends for their encouragement and support throughout this journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Classic Bin Packing

Bin packing is one of the oldest and most fundamental problems in computer science. In the offline setting, the goal is to pack a multiset of items with sizes in $(0, 1]$ into a minimum number of bins of unit capacity, such that the total size of items in each bin is no more than 1. Note that the entire multiset is given to an offline algorithm. In the online setting, however, a *sequence* of items, each identified by a size in $(0, 1]$, is revealed one by one. Upon arrival of an item, it must be placed into a bin without any knowledge about forthcoming items, and decisions of the algorithms are irrevocable (unless some sort of deferral is allowed). As before, the goal is to pack items into a minimum number of bins of unit capacity.

Finding the optimal packing scheme is an NP-hard problem [4]. Nevertheless, it is known that almost optimal solutions can be achieved in the offline setting. In particular, if $\textsc{Opt}(S)$ indicates the number of bins in an optimal packing of a multiset S, there are algorithms that pack S in $\textsc{Opt}(S) + o(\textsc{Opt}(S))$ bins [5]. Note that this is better than a Polynomial-Time Approximation Scheme (PTAS). Whether one can pack items in $\textsc{Opt}(S) + c$ for some constant c is an open problem.

Greedy strategies are widely studied in online bin packing. In FirstFit, each arriving item is placed in the first available bin that can accommodate it; if no such bin exists, a new bin is opened. BestFit places the item in the fullest bin that can still accommodate it, prioritizing tighter packing. NextFit, in contrast, maintains only a single active bin at any time. If an incoming item does not fit in the current bin, that bin is closed permanently, and a new bin is opened. The Harmonic algorithm partitions items into K predefined size intervals: $(\frac{1}{2}, 1]$, $(\frac{1}{3}, \frac{1}{2}]$, $\ldots$, $(\frac{1}{K-1}, \frac{1}{K}]$, and $(0, \frac{1}{K}]$. Items within the same interval are packed separately using the NextFit strategy.

The (asymptotic) *competitive ratio* (CR) of an online bin packing algorithm is defined as the worst-case ratio between the number of bins used by the algorithm and the optimal offline solution, assuming that the cost of the optimal solution goes to infinity (See Definition 1.4.2). The competitive ratio of FirstFit and BestFit is 1.7 [6], while NextFit has a competitive ratio of 2 [4]. The competitive ratio of Harmonic asymptotically converges to 1.691 for sufficiently large values of K [7], while the best known algorithm Harmonic++, which uses similar ideas as Harmonic but with more intricate classes, improves it to 1.588 [8]. A lower bound of 1.54037 is established by Balogh et al. [9], proving that no online bin packing algorithm can achieve a better competitive ratio. We may conclude that the best attainable competitive ratio is in the range $(1.54037, 1.588]$. For more details about online bin packing, see the surveys by Coffman et al. [4], Coffman et al. [10], and Sgall [11] as well as the one by Christensen et al. [12] on online multidimensional bin packing.

Many variants of the online problem have been studied in the past decades, including Renting Servers in Clouds [13, 14, 15, 16, 17], Busy Time Scheduling [18, 19, 20, 21, 22, 23, 24], Dynamic Bin Packing [25, 26, 27], Virtual Machine Allocation [28], Server Allocation [29], Bin Packing with Delay Function [30, 31] and Bin Packing with predictions [9, 32]. See Chapter 2 for a review of the related work.

1.2 Timely Bin Packing

The sequential order of items in the online bin packing problem implies an arrival time. Recently, Khattar [1] has introduced a new variant of online bin packing, in which each item, in addition to its size and arrival time, has a deadline as well. We refer to the window between arrival time and deadline as the "availability time" of the item. In contrast to classic bin packing, where each item must be immediately packed upon its arrival, in the timely packing problem, every item must be packed into a bin at a time t which is within its availability interval. As a consequence, the availability intervals of all items in a bin must have at least a single point in time in common. To illustrate, suppose we have three items of size 1/3 each with availability intervals of $[1, 2]$, $[1.5, 5]$, and $[1.75, 4]$; these three intervals can be packed in a single bin B at any time $t \in [1.75, 2]$. Here, t is called the *delivery time* of B; this terminology roots in an application where we have perishable items in a warehouse that need to be packed into bins (trucks) that deliver them into stores. Note that every bin gets opened, packed, and delivered all at the same time, and only available items at that specific point in time can be placed into that bin.

As mentioned, the Timely Bin Packing problem can be utilized in logistic domains to efficiently organize orders into delivery trucks. Orders for perishable items arrive at the company's warehouse in an online manner (upon the customer's order placement). The orders must be dispatched before their respective deadlines, known as the perish time. Trucks, however, are limited in capacity, with the assumption that they all have the same maximum load. The objective is to minimize the number of trucks required for deliveries. Hence, orders can be likened to items, trucks to bins, and each order comes with an arrival time, deadline, and required capacity delivery via trucks.

The offline setting of the Timely Bin Packing problem is trivially NP-hard via a reduction from classic bin packing. In this thesis, we develop approximation algorithms for the Timely Bin Packing problem. We also study the online variant, which is more relevant in practical settings, improving the existing bounds on its competitive ratio in different settings. Through targeted case studies, we make progress toward narrowing the gap between the best-known upper and lower bounds. We also investigate previously unexplored areas, including the potential of randomization and advice, alongside the development of novel and superior algorithms.

1.3 Dynamic TCP Acknowledgement

Consider a variant of the bin packing problem where each item has both a size and an associated latency function. The longer an item waits before being packed, the higher the latency cost incurred by the algorithm. The objective is to minimize the total cost, defined as the number of bins used plus the cumulative latency cost of all items. This trade-off is abstracted in the Bin Packing with Delay Functions problem [30], which can be seen as a generalization of the Dynamic TCP Acknowledgement problem. The connection between Dynamic TCP Acknowledgement and bin packing, first explored in [33], is described in Chapter 3.

In a communication setting that uses the Transmission Control Protocol (TCP), a client receives a sequence of packets from a server and must send acknowledgements to inform the server of the current state of communication. This process presents a fundamental trade-off between two conflicting goals. On one hand, the server benefits from prompt feedback, which requires the client to send acknowledgements frequently. On the other hand, frequent acknowledgements incur significant communication overhead, making it desirable to reduce their frequency, which eventually increases the latency between a packet's arrival and its acknowledgement. Therefore, the objective function is often defined as the sum of the total acknowledgement and latency costs. More precisely, the cost of each acknowledgement is the weighted sum of a fixed acknowledgement cost plus the *sum* of the latencies of all acknowledged packets.

Max-Latency ($f_{\max}$) Setting. In time-sensitive systems, where fairness and responsiveness are critical, minimizing the sum of latency may result in unfair solutions where some packets have huge latencies while many packets are acknowledged promptly. To address this issue, in the *Max-Latency* setting of the problem [2], the cost of each acknowledgement is the weighted sum of the fixed acknowledgement cost and the *maximum* latency of any acknowledged packet. This cost formulation implies some sort of min-max fairness [3], which has been studied in various contexts previously [34, 35, 36, 37, 38, 39, 40, 41, 42]. Therefore, we use *Fair TCP Acknowledgement* to refer to this max-latency setting of the problem.

In this thesis, we consider the fair Dynamic TCP Acknowledgement problem under the maximum-latency objective in the presence of predictions. We present a family of algorithms with a trust parameter $\lambda \in (0, 1)$ and prove their competitive ratio as a function of λ and the prediction error. In the extreme cases, when $\lambda = 1$, the prediction is ignored and the algorithm becomes a purely online algorithm with no prediction, and when λ converges to 0, the algorithm closely follows the prediction (trusts it).

We also study the advice complexity of the problem and show that any algorithm with sublinear advice has a competitive ratio of at least 1.207. In particular, any algorithm that uses a prediction of sublinear size cannot be better than 1.207-competitive, even if all predictions are perfect.

1.4 Preliminaries

In this subsection, we provide an overview of the key definitions and concepts relevant to our work.

1.4.1 Online Algorithms and Competitive Ratio

We begin with some fundamental definitions related to online algorithms by [43], along with the competitive analysis framework formalized by [44].

Definition 1.4.1. *An* online algorithm *sequentially processes its input one by one, without knowledge of future input. At each step, the algorithm must make an immediate irrevocable decision based only on the information it has received up to that point.*

The performance of an online algorithm is typically evaluated using competitive analysis, which compares its outcome to the optimal offline algorithm that has complete knowledge of the entire input sequence in advance. Let the optimal solution Opt be the best possible offline solution with knowledge of future items. When clear from context, we use Opt to both refer to the optimal algorithm and the cost of the optimal solution. In particular, $\textsc{Opt}(\sigma)$ is the optimal cost (e.g., number of bins in an optimal packing) for an input σ.

Definition 1.4.2 ([44])**.** *The* asymptotic competitive ratio *of an algorithm A is the maximum ratio between the cost of A, denoted by $\textsc{Alg}$, and the cost of an optimal offline solution, denoted by $\textsc{Opt}$, for all sequences.*

$$CR = \max_{\sigma} \frac{\textsc{Alg}(\sigma)}{\textsc{Opt}(\sigma)} \;\; s.t. \;\; \textsc{Opt}(\sigma) \to \infty.$$

The *absolute competitive ratio* of an algorithm A is the maximum ratio between the number of bins opened by A and that of the optimal solution for any input, while the asymptotic competitive ratio is the maximum value of the same ratio restricted to inputs for which the optimal solution opens a sufficiently large number of bins. In this thesis, by competitive ratio, we mean asymptotic competitive ratio.

1.4.2 Randomized Algorithms

Randomization is a powerful tool for improving the performance of online algorithms under worst-case scenarios.

Definition 1.4.3. *A* randomized online algorithm *is a probabilistic function that maps the currently revealed prefix of the input and its internal random choices to an output decision. As before, the algorithm receives the input items one by one, and must make an immediate, irrevocable, and possibly randomized decision based on the parts of the input revealed so far.*

The performance of a randomized online algorithm is evaluated in terms of its expected cost for worst-case inputs generated by an *oblivious adversary*, which knows the definition (the code) of the algorithm but not its random choices. The expectation is taken over the distribution of its random choices for an input sequence. The competitive ratio of a randomized online algorithm is defined as the maximum ratio between its expected cost and the cost of an optimal offline algorithm over all input sequences. Namely,

$$\mathrm{CR} = \max_{\sigma} \frac{E[\textsc{Alg}(\sigma)]}{\textsc{Opt}(\sigma)} \;\; s.t. \;\; \textsc{Opt}(\sigma) \to \infty.$$

where $E[\textsc{Alg}(\sigma)]$ is the expected cost of $\textsc{Alg}$ for σ over all its random bits.

Randomization is particularly useful in adversarial scenarios, where deterministic algorithms may perform poorly, as it allows the algorithm to hedge against worst-case inputs by introducing unpredictability in its decisions. Yao's Minimax Lemma is a standard tool for proving lower bounds on the competitive ratio of randomized algorithms (e.g., in this thesis, we have used it in Theorems 4.2.2, 4.5.7, 4.5.10, 4.5.13). It asserts that the expected cost of any randomized algorithm against a fixed worst-case input is at least equal to the expected cost of the best deterministic algorithm against a distribution of random input sequences. Formally, it can be stated as follows:

Definition 1.4.4 (Yao's Minimax Principle [44]). *Let $\mathcal{A}$ be a finite set of deterministic algorithms and $\mathcal{X}$ a finite set of input instances. Let $c(a,x)$ be the cost of algorithm $a \in \mathcal{A}$ on input $x \in \mathcal{X}$. Then,*

$$\max_{x\in\mathcal{X}} \mathbb{E}[c(A,x)] \geq \min_{a\in\mathcal{A}} \mathbb{E}[c(a,X)],$$

where A is a randomized algorithm (a distribution over $\mathcal{A}$), and X is a distribution over inputs in $\mathcal{X}$.

Intuitively, if we want to prove a negative result concerning the competitive ratio of a randomized algorithm for its worst-case input, it suffices to establish the same result for the competitive ratio of a deterministic algorithm faced with random input sequences.

1.4.3 Advice

The advice model is an abstraction for relaxing the online constraint and allowing an online algorithm to receive partial information about the input sequence before the input is revealed. We briefly review the standard advice model [45], which can be thought of as a model in which online algorithms receive perfect (error-free) but succinct predictions about the input. This is a theoretical model that is useful to establish impossibility results for predictions of small sizes (e.g., in this thesis, we used it in Theorem 5.4.1 and Lemma 4.3.3).

Definition 1.4.5. *An* online algorithm with advice *is equipped with a finite sequence of bits, called advice bits, encoding partial information about the future input sequence. The advice bits are available before the input starts being revealed and are generated by a benevolent oracle that knows the entire input and has unbounded computational power. The algorithm is assumed to know the meaning of what advice encodes.*

The algorithm can use advice bits to make more informed decisions, potentially improving its competitive ratio compared to a purely online algorithm. The amount of advice required is measured in terms of the number of advice bits per input, and the efficiency of an advice-based algorithm is analyzed as a trade-off between the quality of solutions (e.g., competitive ratio) and the amount of advice needed (advice size). A comprehensive overview of the advice model is provided in the survey by Boyar et al. [45]. For example, in the DYNAMIC TCP ACKNOWLEDGEMENT problem, an advice tape of n bits can be used to solve the problem optimally. For each arrival a_j, the bit j of the tape indicates whether to acknowledge the pending packets at time j (1) or defer acknowledgement (0).

Reductions from the *string guessing* problem are a standard method for proving advice complexity lower bounds [46, 47]. In the *BINARY STRING GUESSING* problem, an algorithm is given a sequence of binary bits revealed one at a time and must guess each bit before it is revealed. The algorithm incurs a unit cost for each incorrect guess, and its total cost is the sum of these penalties. With a single bit of advice indicating whether the number of 0's exceeds the number of 1's in the entire sequence, the algorithm can correctly guess at least half of the bits by consistently predicting the majority bit. More importantly, one cannot guess a fraction $1/2 + \epsilon$ of bits correctly unless a linear $\omega(n)$ number of advice bits are available [48]. Therefore, this problem is often used as the starting point for proving impossibility results for advice (thus predictions) of sublinear size.

1.4.4 Algorithms Augmented with Prediction

Augmenting algorithms with machine-learned predictions has garnered significant attention in recent years. Predictions can help improve the time complexity of offline algorithms or the competitive ratio of online algorithms. Our main interest in this thesis is the latter.

Definition 1.4.6. *An* online algorithm with predictions *receives one or more predicted items, which can be available at the beginning or revealed over time. Each predicted item $\hat{a}_j$ is an estimate of the actual item a_j, for $j \in [n]$. The algorithm makes a decision at time t as a function of the predictions made up until time t alongside the observed input revealed up until time t.*

There are a few considerations when discussing algorithms with predictions:

- **Prediction Model:** We need to specify what information is being predicted. For example, for certain variants of the discrete bin packing problem, the frequency distribution of item sizes may be predicted in one shot at the beginning [32]. In the context of TCP acknowledgement, the arrival time of the next packet is predicted as the input is revealed; this type of prediction has been studied in [49, 50, 51, 52, 53, 54, 55, 56]. In practical settings, it is desirable to predict concepts that are Probably-Approximately Correct (PAC) learnable [57]. Informally speaking, that means the predictions can be made to be arbitrarily close to the actual concept with arbitrarily high probability, using samples of only a finite size.

- **Error Model:** The quality of the predictions is typically quantified by a *prediction error* measure, such as the number of incorrect prediction items or a distance metric between the prediction vector $\hat{a}$ and true predicted values a. In other words, an error model is a distance measure that quantifies the discrepancy between the predicted value and the intended value, thereby assessing the quality of the prediction. For example, if k predictions are made in a vector $\hat{p}$, while actual values are stored in a vector p, the error can be modelled as the ℓ_1 and ℓ_∞ between the two vectors. We note that there are different ways to model error, and the definition of error is problem-specific.

The performance of an online algorithm is evaluated in terms of its competitive ratio or regret.

Definition 1.4.7. Regret *measures the gap between the performance of an online algorithm and the best possible decision in hindsight.*

$$\textit{Regret} = \max_{\sigma}(\textsc{Alg}(\sigma) - \textsc{Opt}(\sigma)) \;\; \textit{s.t.} \;\; \textsc{Opt}(\sigma) \to \infty.$$

In the context of prediction, the competitive ratio and regret are typically expressed as a function of the prediction error; the higher the error, the worse the competitive ratio and regret. We use CR(η) and $\textit{Regret}(\eta)$ to denote the competitive ratio and regret as functions of the prediction error η, respectively.

Ideally, the goal is to design algorithms whose regret grows sublinearly with the input length, which is often referred to as the number of time steps (T); this ensures that the average regret per time step approaches zero as T increases. Note that in these scenarios, the competitive ratio converges to 1. In this sense, regret serves as an analysis measure that can be considered as the additive analogue of the competitive ratio, which is based on multiplicative comparisons.

Recent work by Purohit et al. [58] and Lykouris and Vassilvitskii [49] demonstrates how predictions can be incorporated into the design and analysis of online algorithms using two key performance measures. They introduced the notions of *consistency*, which quantifies the algorithm's performance under perfect predictions, and *robustness*, which captures its performance under worst-case prediction errors. These concepts have since been applied in a wide range of online algorithmic settings; see Section 2.5 and, for example, the survey by Mitzenmacher and Vassilvitskii [59].

Definition 1.4.8. *Consider a learning-augmented online algorithm A.*

- *A is said to be κ-*consistent *iff, for its competitive ratio, we have $CR(\eta) = \kappa$ when $\eta = 0$ (perfect prediction). That is, the cost of A when provided with perfect prediction is no* *more than κ times the cost of* Opt *for any input sequence and any perfect prediction about that sequence.*

- *A is said to be* γ-robust *iff for its competitive ratio, it holds that* $CR(\eta) \leq \gamma$*. That is, the cost of A is no more than* γ *times the cost of* Opt *for any input sequence and any prediction (with any large* η*) about that input.*

The notation $\{\kappa, \gamma\}$-*competitive* is commonly used to describe an algorithm's performance in terms of both consistency and robustness. There is a typical tension between consistency and robustness. For example, an algorithm that blindly trusts and follows predictions turns out to have good consistency (its competitive ratio is good when all predictions are correct), but in most cases, poor robustness (relying too much on predictions deteriorates performance in situations where the predictions are erroneous). The consistency-robustness framework enables the study of trade-offs between reliance on predictions and algorithm robustness to errors (For example, we have used it in Collorary 5.2.6).

Chapter 2

Literature Review

Both bin packing (and its variants) and Dynamic TCP Acknowledgement are among the most extensively studied in online algorithms, driving significant advancements in the field. This section presents a review of the main results and methodologies for these problems, with a focus on recent advancements that consider delay functions and deadlines in these problems' formulations.

2.1 Offline and Online Bin Packing

We begin by formally defining the offline bin packing problem.

Definition 2.1.1. *In the Offline Bin Packing problem, we are given a multiset of n items, with sizes $\{s_1, s_2, \ldots, s_n\}$, where $0 < s_i \leq 1$ for all i, and an unlimited number of bins, each with a fixed capacity of 1. The objective is to pack (partition) the items into the minimum number of bins so that the total size of the items assigned to any bin does not exceed its capacity of* 1.

The offline bin packing problem is known to be NP-hard (see, e.g. [60]). The hardness proof, which is based on a simple reduction from the Partition problem [60], also shows that the *absolute* approximation ratio of any polynomial-time algorithm is no less than $3/2$ (assuming P $\neq$ NP); that is, for inputs that may require a small number of bins, one cannot hope for an approximation factor better than 1.5. Recall that our main focus in this thesis is improving the asymptotic approximation factor (which we simply call "approximation factor") under the assumption that the cost of Opt goes to infinity

Definition 2.1.2. *In the Online Bin Packing problem, items with sizes* $0 < s_i \leq 1$ *arrive sequentially and must be packed into bins of unit capacity immediately upon arrival. Packing* *decisions are irrevocable and made without any knowledge of future items. The objective is to minimize the total number of bins used while ensuring that the total size of items in each bin does not exceed its capacity of 1.*

Next, we explore major advancements across various aspects of this problem.

2.1.1 Approximation Algorithms for Offline Bin Packing

The FirstFitDecreasing (FFD) algorithm begins by sorting items in decreasing order of their sizes (hence it is an offline algorithm), and then applies the FirstFit algorithm to the sorted list. It achieves an approximation ratio of 11/9 [6], which has been followed by a sequence of works refining the additive constant [61, 62, 63, 64]. More generally, applying any AnyFit strategy to a decreasingly sorted sequence yields an approximation ratio of at least 11/9. A variant of FFD, known as Modified FFD, improves this bound to 71/60 while preserving good average-case behavior [65].

The first polynomial time $(1+\epsilon)$-approximation algorithm for bin packing is proposed by de la Vega and Lueker [66] by applying linear grouping, a technique that reduces the number of distinct item sizes while maintaining an approximation guarantee. A major improvement came from Karmarkar and Karp [67], who achieved a guarantee of $\text{Opt} + \mathcal{O}(\log^2 \text{Opt})$ by introducing iterative rounding within a linear programming framework. This result was further refined by Rothvoß [5], who improved the bound to $\text{Opt} + \mathcal{O}(\log \text{Opt}. \log\log \text{Opt})$ using methods from discrepancy theory. Subsequently, Hoberg and Rothvoß [68] achieved an approximation of $\text{Opt}+\mathcal{O}(\log \text{Opt})$ by combining discrepancy theory with a novel two-stage packing approach, significantly narrowing the gap to optimality. It is unknown whether there is a polynomial algorithm that uses $\text{Opt} + \mathcal{O}(1)$ bins.

2.1.2 Online Algorithms with Advice

Note that online algorithms with advice can be thought of as semi-offline versions of the problem, and we first review them here. Boyar et al. [69] provided tight bounds of $\Theta(n \log \text{Opt})$ on the amount of advice required to achieve optimal packing in the online bin packing. They also proposed an algorithm that, with $\log n + o(\log n)$ bits of advice, achieves a competitive ratio of 3/2 for the general problem. They showed that another algorithm, with $2n + o(n)$ bits of advice, achieves a competitive ratio of $4/3 + \epsilon$. Additionally, they established an impossibility result via a reduction from the binary guessing problem, showing that linear-size advice is necessary to achieve a competitive ratio better than $9/8 \approx 1.125$. Angelopoulos et al. [70] further improved this lower bound to $7/6 \approx 1.166$ and introduced algorithms that utilize small-size advice to achieve competitive ratios surpassing those of purely online methods. Mikkelsen [71] later improved the lower bound to $4-2\sqrt{2} \approx 1.71$ by reducing weighted binary string guessing to bin packing. They showed that for online bin packing and various scheduling problems, a competitive ratio arbitrarily close to 1 can be achieved using only a constant number of bits of advice per request. Specifically, for the bin packing problem, they present an online algorithm that is $(1+\epsilon)$-competitive $(0 < \epsilon \leq 1/2)$ while requiring $\mathcal{O}\left(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\right)$ bits of advice per request.

2.1.3 Discrete Bin Packing and Online Algorithms with Frequency Prediction

An interesting and practical variant of bin packing is discrete bin packing, where item sizes belong to $\{1/k, 2/k, \ldots, 1\}$ for some integer k. Unfortunately, most negative results concerning competitive ratio can be extended to this setting, e.g., by having a large (but constant) k, an adversary can ensure that no algorithm can achieve a competitive ratio better than 1.54037 [9].

Angelopoulos et al. [32] examined algorithms for the discrete bin packing problem, with potentially erroneous predictions of item size frequencies. In other words, for each item a_i of size i/k, a prediction about the frequency of a_i in the input sequence is made. They designed and analyzed algorithms with parameters that allow balancing consistency (optimal performance with accurate predictions) and robustness (resilience under adversarial errors). They established the Pareto-efficiency of their approach (with respect to consistency and robustness) and demonstrated that the performance degradation of their algorithm is near-optimal as a function of prediction error.

2.1.4 Dynamic Bin Packing

Coffman et al. [25] introduced the *dynamic bin packing* problem, where each item, in addition to having a size, is also associated with an arrival and departure time. At any given moment, the total size of all "active" items within a bin must not exceed its capacity. The objective is to minimize the maximum number of bins ever used during the instance by the algorithm. There exists a lower bound of 8/3 [26] and an upper bound of 2.897 [25] for the competitive ratio of online dynamic bin packing algorithms. In another variant [27], the goal is to minimize the total active time of all bins used. The authors of [27] focused on the clairvoyant setting (where the departure times are known upon arrival) and proved a tight competitive ratio of $\Omega(\sqrt{\log \mu})$ for any online algorithm in this setting. We note that, in the problems studied in this thesis, items never depart from their bins (yet, they may have an arrival and a deadline).

The online *busy time scheduling* problem considers an unbounded number of machines, each equipped with g processors that can process jobs characterized by their processing time and deadline (it assumes all items have the same size of $1/g$). A machine consumes energy whenever at least one of its processors is actively running a task. The objective is to minimize the sum of all busy times of all machines. Flammini et al. [18] initiated the study of the problem in the *rigid* jobs setting, which means the jobs must be assigned immediately at the time they reveal (the deadline is equal to the arrival time plus the processing time). Even when all items are rigid, the busy time scheduling problem is NP-hard even when $g = 2$ [19]. There exists a 3-approximation algorithm for this setting [20]. In the online setting, no algorithm can have an absolute competitive ratio better than g [20]. Also, considering the maximum length of a job to be μ, there exists an online algorithm with a competitive ratio of at most $5 \log \mu$ [20]. In busy time scheduling when the items are *flexible* (not rigid), there exists an optimal polynomial-time offline algorithm for $g \to \infty$ [21]. Also, a lower bound of $(1 + \sqrt{5})/2$ for unbounded flexible jobs is proved in [22]. Chau and Li [23] has a survey on busy time problems. In the problems studied in this thesis, unlike busy time scheduling, the objective is to minimize the *number* (not busy time) of used servers (bins).

The *renting servers in the cloud* problem [13] is a generalization of the online busy time scheduling problem, with the key difference that jobs can have arbitrary sizes instead of a fixed size of $1/g$. Kamali and López-Ortiz [14] analyzed bin packing strategies for this setting, showing that the lower bound of μ extends to all online algorithms, where μ is the max/min interval length ratio of jobs. Masoori et al. [15] provided tight bounds on the problem's competitive ratio under parameterized constraints, including a ratio of 2 for cases with two servers and 3 for up to four servers, supported by experimental results. Li et al. [16] extended the problem to multi-dimensional settings, introducing monotone AnyFit algorithms with a tight competitive ratio of $\Theta(d\mu)$, where d is the dimensionality, and showed direct-sum properties for extending algorithms to higher dimensions. Masoori et al. [17] further analyzed NextFit and FirstFit for jobs of equal duration, proving a tight competitive ratio of 2 for NextFit and providing nearly tight bounds for FirstFit, including an upper bound of 2.565 for cases with two arrival times. Additionally, Davis et al. [72] developed resource allocation models that minimize reallocation costs in dynamic environments where items arrive and depart over time. Like Busy time Scheduling, Renting Servers in the Cloud differs from the problems studied in this thesis in the sense that it wants to minimize the total usage time (renting cost) rather than the number of used bins, as we aim to minimize.

2.2 Timely Bin Packing and Its Generalizations

Khattar [1] introduced Timely Bin Packing, where items are accompanied by a deadline and must be packed into a bin before their deadline (see Section 3.1 for a formal definition). They proved a general lower bound of 1.2 for the competitive ratio of any deterministic online algorithm, even if an item deadline is revealed upon its arrival. Moreover, they proposed the Pack-at-Deadline algorithm, a simple deterministic algorithm, and claimed an upper bound of 2 and established a lower bound of $1.\bar{6}$ for its competitive ratio. However, we demonstrate a flaw in their upper-bound analysis and show that the true analysis establishes a lower bound of 2.5 for the competitive ratio of Pack-at-Deadline (See Theorem 4.2.5 for the proof of the lower bound). We also differentiate between deadline-aware and deadline-oblivious algorithms, depending on the knowledge of items' deadlines, and introduce both algorithmic and impossibility results for various settings of the problem. We refer to Table 4.1, for a summary of known and new results for the generic problem, and Table 4.2 for our results concerning special settings of the problem (e.g., unit length for availability intervals of all items).

While the theoretical framework of Timely Bin Packing remains valuable for applications in logistics and resource scheduling where items have strict temporal constraints, this correction highlights the challenges in establishing tight bounds for this problem.

2.2.1 Bin Packing with Delay and Holding Costs

Bin Packing with Delay and Holding Costs was studied first by Ahlroth et al. [33], who compared the online bin packing and TCP acknowledgement problems, noting that the best-known asymptotic competitive ratio for online bin packing lies between 1.5403 and 1.5889, while TCP acknowledgement has a tight competitive ratio of 2. They introduced a generalization of online bin packing that includes (i) bin opening costs (starting a bin), (ii) item delay costs, capturing the cost for each item since its arrival up until its bin is closed (finalized and no new items placed there), and (iii) holding costs for open bins. Timely Bin Packing can be thought of as a special case where (i) opening cost for all bins is 1, (ii) delay cost for each item is 0 before its deadline and ∞ after the deadline, (iii) holding cost for all bins is 0. On the other hand, the Dynamic TCP Acknowledgement (Definition 3.2.1) with parameter ρ is a setting where all items have size 0 and (i) opening costs for all bins is a fixed value ρ, (ii) delay cost for all items is identical and a linear function of the time between their arrival and closing of their bins, and (iii) holding cost is 0.

Bin packing with delay and holding costs not only has practical applications but also unifies several previously studied online optimization problems. A general online algorithm is presented with a competitive ratio of 7 [33], alongside improvements for special cases, such as when delay or holding costs are zero or when item delay costs are proportional to their sizes. A variant of the bin packing problem with delay considers each item i to be associated with a positive, monotonically non-decreasing delay function $d_i(t)$. Let $t_i \geq 0$ denote the time elapsed from the arrival of item i until it is packed; the delay cost of item i is then $d_i(t_i)$. The objective is to minimize the total cost, defined as the number of bins used plus the sum of delay costs across all items. This problem can be thought of as a case of Bin Packing with Delay and Holding Costs, with the holding cost of all bins being 0.

Epstein [30] established a competitive ratio upper bound of 3.1551 for the online deterministic setting of this problem, improving upon the previous best bound of 3.951 by Azar et al. [31]. Given that Dynamic TCP Acknowledgement is a special case Bin Packing with Delay Functions with zero-size items and identical, linear delay functions for all items, all negative results concerning Dynamic TCP Acknowledgement also apply to Bin Packing with Delay Functions. In particular, there is no deterministic algorithm for Bin Packing with Delay Functions with a competitive ratio better than 2 because no such algorithm exists for Dynamic TCP Acknowledgement [30]. For results concerning Dynamic TCP Acknowledgement, see Section 2.3.

2.3 Dynamic TCP Acknowledgement

In the Dynamic TCP Acknowledgement problem, the input is an online sequence of *packets* arriving over time, and the algorithm must decide when to send acknowledgements. Each acknowledgement confirms receipt of all pending packets and incurs a fixed cost for a given parameter ρ. The objective is to minimize the total cost, which is the weighted sum of the number of acknowledgements and a *latency cost* experienced at each acknowledgement time. Two common formulations differ in how latency cost is measured: the $f_{\max}$ is the maximum delay of any pending packet at each acknowledgement point, while the f_{sum} objective is the total cumulative delay of all pending packets. Refer to Section 3.2 for a formal definition of the problem.

Next, we examine major advancements across various aspects of this problem.

2.3.1 Offline and Online Algorithms

Dooly et al. [2] introduced the Dynamic TCP Acknowledgement problem and provided $\mathcal{O}(n^2)$-time dynamic programming algorithms for solving the offline problem optimally under both objective functions, which was later optimized to take $\mathcal{O}(n)$ time [73]. They also presented 2-competitive deterministic algorithms for both f_{sum} and $f_{\max}$ objective functions, which is the best possible ratio for deterministic algorithms. Karlin et al. [74] proposed a randomized algorithm for the Dynamic TCP Acknowledgement problem with a competitive ratio of $e/(e-1)$. This bound was later shown to be optimal for randomized algorithms by Seiden [75], who introduced a general framework for establishing lower bounds on the competitive ratio of randomized online algorithms. Buchbinder et al. [76] later analyzed the problem within a primal-dual framework and designed another $e/(e-1)$-competitive (and thus optimal) randomized algorithm.

Albers and Bals [77] considered a slightly different variant than $f_{\max}$, where there is no latency cost associated with individual acknowledgement, and instead a single maximum latency, over all packets, is added to the sum of acknowledgement costs to form the total cost (See Example 3 for an illustration). For this variant, they developed a deterministic algorithm with a competitive ratio of $\pi^2/6 \approx 1.644$ and proved no deterministic algorithm can achieve a better competitive ratio. They also showed that no randomized algorithm can achieve a competitive ratio smaller than $3/(3-2/e) \approx 1.324$.

Frederiksen et al. [78] explored deterministic and randomized algorithms for $f_{\max}$ under the constraint that two successive acknowledgement times must be at least one unit time apart. They presented an optimal $(1+\sqrt{5})/2$-competitive deterministic algorithm and an optimal $(\sqrt{3}+1)/2$-competitive randomized algorithm. Additionally, they examined a class of algorithms that use only a limited number of random bits.

2.3.2 Online Algorithms with Lookahead

Dooly et al. [2] showed that in the case of the $f_{\max}$ objective function, it is enough to know the arrival time of the next packet to achieve an optimal solution. On the other hand, in the case of the f_{sum} function, the knowledge of the next k arrivals is not enough to have a better competitive ratio than 2 for any constant k.

A related setting is studied in [79], where the algorithm is given access to all packet arrivals within a fixed time window of length c, instead of a fixed number of future packets. For the $f_{\max}$ objective, an optimal algorithm is possible when $c \geq \rho/(1-\rho)$, and a $2-c(1-\rho)/\rho$-competitive algorithm can be designed for smaller values of c. It was also shown that no better competitive ratio can be achieved. For the f_{sum} objective, an algorithm with a competitive ratio of $1+\mathcal{O}(1/c)$ was presented, and a lower bound of $1+\Omega(1/c^2)$ was proven, showing that no 1-competitive algorithm is possible with constant lookahead.

2.3.3 Rent-or-Buy Algorithms and Relationship to Dynamic TCP Acknowledgement

Consider a rent-or-buy game, also known as the *ski-rental problem*, where a player (algorithm) goes skiing for an unknown number of n days. Every day, the skier must either rent equipment at a cost of 1 or buy it at a cost of $b > 1$; in the latter case, it does not need to make a decision for subsequent days. The objective is to minimize the total cost of the algorithm. For example, a break-even algorithm that rents for $b - 1$ days and buys at day b (if it ever gets to that day) is known to have a competitive ratio of 2 [44], which is the optimal ratio for deterministic algorithms. The best randomized algorithm is known to have a competitive ratio of $e/(e - 1)$ [44].

2.4 Min-Max Algorithmic Fairness

Min-max fairness is a widely used and intuitive notion of fairness, aiming to distribute workload as evenly as possible across all parties; see the surveys [3] and the framework in [34]. This notion of fairness involves minimizing the maximum load/cost (or symmetrically maximizing the minimum benefit) of agents/parties/objects to convey a notion of equity between them. For example, consider the problem of backbone routing, where routes and bandwidths are assigned to links to satisfy demands. In the min-max fair setting, an algorithm aims to minimize the load on the most congested link [35]. Reflecting similar fairness objectives in algorithmic settings, a variant of the online paging problem called min-max paging has been proposed, where the goal is to minimize the maximum number of faults experienced by any single page [36].

Paging is indeed a special case of the k-server problem on uniform metrics. The k-server problem involves efficiently moving k servers in a metric space to service a sequence of requests, aiming to minimize the total distance travelled by the servers on a given metric space [80]. A recent variant of the classic k-server problem, called Fair k-Food problem, aims to maximize the minimum reward earned by any delivery agent within a food-delivery network modelled as a metric space [42].

Our interest in Min-Max fairness arises from the f_{sum} objective in the TCP acknowledgement problem, which can be viewed as a fair variant of TCP acknowledgement aiming to balance the delay cost across all acknowledgements.

2.5 Learning-Augment Algorithms

In recent years, there has been an increasing interest in improving the performance of online algorithms via the notion of *prediction*. Here, it is assumed that the algorithm has access to a prediction, for instance, in the form of machine-learned information. Unlike the advice model, the prediction may be erroneous, and its error is quantified by an *error measure* η. The objective is to design algorithms whose competitive ratio degrades gently as a function of η. Several online optimization problems have been studied under the prediction model, various scheduling problems [58, 81, 82, 83, 84, 85, 86, 87, 88, 89], knapsack [90, 91, 92], caching [50, 93, 94, 95], matching problems [96, 97, 98], time series search [99], and various graph problems [100, 101, 102, 103, 104]. See also the survey by Mitzenmacher and Vassilvitskii [105] and the collection at [106]. Here, we focus on the related work on the Dynamic TCP Acknowledgement problem.

A parameter-learning algorithm for the f_{sum} objective is proposed in [107]. The algorithm has a parameter p that is being dynamically learned and updated by the algorithm. At each step, the algorithm selects the best waiting time before making the next acknowledgement, based on the parameter p and uses it for the next few arrivals, periodically updating the parameter based on recent observations. The authors showed the effectiveness of their approach through experimental analysis. Bamas et al. [108] assumed access to a complete predicted solution (that is, the exact acknowledgement times are given to the algorithm at the beginning) and designed a primal-dual algorithm that mimics the prediction. Their algorithm achieved a cost of $\min\{\frac{\lambda}{1-e^{-\lambda}}\textsc{Alg}, \frac{1}{1-e^{-\lambda}}\textsc{Opt}\}$ for any trust parameter $\lambda \in (0,1]$, where $\textsc{Alg}$ is the cost of an algorithm that sends acknowledgements precisely on the predicted timesteps and $\textsc{Opt}$ is the cost of the optimal offline solution on the true input. Clearly, this type of prediction is not available in practical settings (one can think of predictions appearing in an online manner as the sequence is being revealed, but a complete solution for the entire input is unlikely to be available in any practical scenario).

Im et al. [109] studied Dynamic TCP Acknowledgement under the f_{sum} measure, assuming predictions of the number of packets arriving at each time t. Let L denote the actual input sequence, where L_t denotes the number of packet arrivals at time t, and let $\hat{L}$ be the predicted sequence defined similarly. To quantify the error, two input sequences L' and L'' are defined, where, for each $t \in [n]$, we define $L'_t = \min\{L_t, \hat{L}_t\}$ and $L''_t = \max\{L_t, \hat{L}_t\}$. And they let η be a function of $\textsc{Opt}(L'') - \textsc{Opt}(L')$. They present an algorithm that does not require access to the full predicted solution and achieves a balanced trade-off between consistency and robustness. Their randomized algorithm achieves a competitive ratio of at most $\min\{(1+\epsilon)\textsc{Opt} + O(\frac{1}{\epsilon^2})\eta, (\frac{e}{e-1}+\epsilon)\textsc{Opt}\}$, while their deterministic algorithm achieves competitive ratio $\min\{(1+\epsilon)\textsc{Opt} + O(\frac{1}{\epsilon^2})\eta, (2+\epsilon)\textsc{Opt}\}$, for any $\epsilon > 0$. It should be noted that this work does not address the $f_{\max}$ objective.

The ski-rental problem is related to the Dynamic TCP Acknowledgement in the f_{sum} setting, where only one acknowledgement can be made (postponing an acknowledgement translates to a delay, which projects to renting the equipment in the ski-rental problem). Banerjee [110] studied learning-augmented algorithms for rent-or-buy problems, including Dynamic TCP Acknowledgement under the f_{sum} objective. Assuming predictions on the optimal acknowledgement times, the author proposed a randomized Hedge-based algorithm [111] that combines classical strategies with ML predictions. They showed that in order for the ML predictions to improve upon the competitive ratio of randomized algorithms (i.e., $e/(e-1)$), the prediction must be more accurate than a certain bound.

This thesis targets the $f_{\max}$ objective, which is more relevant in latency-sensitive settings. Our approach avoids probabilistic mixtures and expert-weighting schemes, instead providing a simple, deterministic algorithm with analytically tight guarantees.

Chapter 3

Problems Statement

This section formally defines the main problems of our study, highlighting their relationship.

3.1 Timely Bin Packing

We start with a formal definition of the TIMELY BIN PACKING problem, which is based on the definition of Khattar [1].

Definition 3.1.1. *The input to the* TIMELY BIN PACKING *problem is a multiset* $L = \{a_1, \dots, a_n\}$ *of* n *items each defined by a tuple* $a_i = (s_i, r_i, d_i)$ *where* $i \in [1, n]$*. Here,* $s_i \in (0, 1]$ *denotes the size of* a_i*, while* r_i *and* d_i *are called the* release time *and* deadline *of* a_i*, and we have* $r_i \leq d_i$*. Furthermore, the interval* $[r_i, d_i]$ *is called the availability interval of* a_i*. The objective is to pack items in* L *into a minimum number of bins under the following two constraints:*

- *The total sum of items placed in the same bin must not exceed* 1 *(capacity constraint).*
- *Each bin* B *must be "closed" (equivalently "delivered") at some time* $\tau(B)$ *such that the availability interval of all items placed in* B *contains* $\tau(B)$*. This implies that each item* i *is packed in a bin that is closed before the deadline* $d(i)$ *of* i*.*

In the offline setting, all items are available in advance.
In the online setting, items appear one by one in the order of their arrival time. Namely, at each given time t*, the sizes of items whose arrival time is* t *are revealed to the algorithm.*
In the "deadline-aware" setting, in addition to the size s_i *of such an item* i*, its deadline is also revealed at its arrival time* r_i*, while in the "deadline-oblivious" setting, the deadline is revealed only at time* d_i*.*

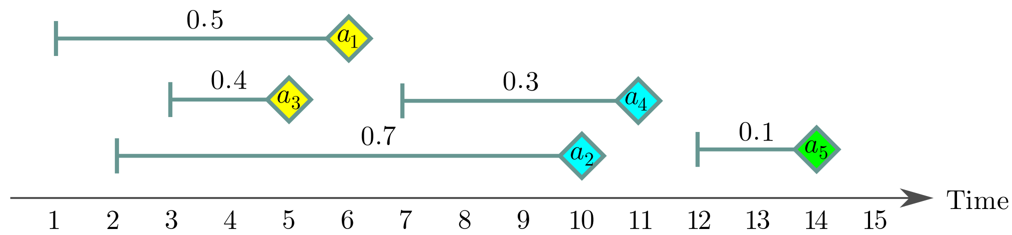


Figure 3.1: An example illustrating a feasible solution for the given TIMELY BIN PACKING instance. Items of the same color are packed into the same bins.

Example 1. *Consider the input* $L = \{a_1 = (s_i = 0.5, r_i = 1, d_i = 6), a_2 = (0.7, 2, 10), a_3 = (0.4, 3, 5), a_4 = (0.3, 7, 11), a_5 = (0.1, 12, 14)\}$ *of the* TIMELY BIN PACKING *problem. Figure 3.1 shows a valid solution of cost* 3 *for packing* L*, where items of the same color are packed into the same bin, and the size of each item is indicated above its associated time interval. The items* 0.4 *and* 0.5 *can be packed into a bin* B_1 *since their combined size is at most* 1*, and their intervals overlap at least a single point, e.g.,* $\tau(B_1) = 3$*. Note that* a_1 *and* a_2 *cannot be packed together since, despite sharing a time interval with item* 0.7*, their cumulative size exceeds 1. Moreover, while the total size of* a_1, a_3*, and* a_5 *(which is* $0.5 + 0.4 + 0.1 = 1$*) remains below the maximum bin capacity, the absence of any overlapping time in their availability intervals prohibits them from being packed together.*

Throughout the thesis, we refer to an item (s_i, r_i, d_i) as a **trigger item** iff we have $r_i = d_i$. In other words, a trigger item must be packed right after its arrival.

Definition 3.1.2. *[30] The input to the* BIN PACKING WITH DELAY FUNCTIONS *problem is a multiset of* n *items* $L = \{a_1, \ldots, a_n\}$*, where each item* a_i *is defined by a tuple* $(s_i, r_i, f_i(t))$*, where* $s_i \in (0, 1]$ *denotes the size of* a_i*,* r_i *denotes the release time of* a_i*, and* $f_i(t)$ *is a continuous, non-decreasing function of* t*, called "delay function". The objective is to pack items in* L *into a packing of minimum cost under the following constraints:*

- *The total sum of items placed in the same bin must not exceed* 1 *(capacity constraint).*
- *Each bin* B *must be "closed" (equivalently "delivered") at some time* $\tau(B)$ *such that the release time of all items packed in* B *is no more than* $\tau(B)$*.*

The cost of a bin B *is the sum of two components. The first component is a fixed value 1 (the default cost of* B*), while the second component is the total delay cost of all items in* B*, where the delay cost of each item* a_i *packed in* B *is* $f_i(\tau(B) - r_i)$*. The cost of a packing is the total cost of all bins, which one aims to minimize.*

Example 2. *Consider the input* $L = \{a_1 = (s_i = 0.5, r_i = 1, f_i(t) = 2t), a_2 = (0.7, 2, t^2), a_3 = (0.4, 3, 1.5t), a_4 = (0.3, 7, 3t)\}$ *of the* TIMELY BIN PACKING *problem. One* *can pack a_1 and a_2 at the same bin B_1, which is closed at time $\tau(B_1) = 2$; the delay costs of a_1 and a_3 would be 2 and 0, respectively, and the cost of B_1 would be $1 + 2 = 3$. Similarly, one can place a_2 and a_4 in the same bin B_2, which is closed at time $\tau(B_2) = 8$; the delay costs of a_2 and a_4 will be respectively $6^2 = 36$ and 3, respectively, and the cost of B_2 would be $36 + 3 + 1 = 40$.*

3.2 Dynamic TCP Acknowledgement

Next, we formally define the DYNAMIC TCP ACKNOWLEDGEMENT problem, based on the definition provided in [2].

Definition 3.2.1. *The input to DYNAMIC TCP ACKNOWLEDGEMENT is a multiset of n packets, where each packet is identified solely by its arrival time a_i. In the offline setting, all packets are given in advance, while in the online setting, they appear one by one in the non-decreasing order of their arrival times. A solution to the problem defines a sequence of $\langle t_1, \ldots, t_k \rangle$ of acknowledge times ($t_j < t_{j+1}$), and packets whose arrival time is in $(t_{j-1}, t_j]$ are acknowledge at time t_j. The acknowledgement t_j has a cost associated with it, and the objective is to form an acknowledgement sequence of minimum total cost (cost over all acknowledgements).*

There are two cost formulations for the cost of acknowledgement t_j, as described below. In both settings, each acknowledgement incurs a fixed cost $\rho \in (0, 1)$, which we refer to as acknowledgement cost. *Moreover, each acknowledgement has a* latency cost *(also referred to as "delay cost") that differs across the two settings. The fixed parameter ρ indicates the importance of minimizing acknowledgements in comparison to minimizing extra latency.*

- *In the f_{max} setting, the cost of acknowlegdemnt t_j is $\rho + (1 - \rho) f_{max}(j)$, where $f_{max}(j)$ is the maximum delay of any packets acknowledged at t_j, that is, $f_{max}(j) = \max\limits_{i \text{ s.t. } t_{j-1} < a_i \le t_j} (t_j - a_i)$.*
- *In the f_{sum} setting, the cost of acknowlegdemnt t_j is $\rho + (1 - \rho) f_{sum}(j)$, where $f_{sum}(j)$ is the sum of delays of all packets acknowledged at t_j, that is, $f_{sum}(j) = \sum\limits_{i \text{ s.t. } t_{j-1} < a_i \le t_j} (t_j - a_i)$.*

The DYNAMIC TCP ACKNOWLEDGEMENT problem with f_{sum} objective is a special instance of the BIN PACKING WITH DELAY FUNCTIONS Problem, in which packets are items with identical and linear delay, and their sizes are minuscule in the sense that any number of items can be packed into a single bin (because any number of packets can be acknowledged at any given time). Specifically, determining when to send acknowledgements in TCP is similar to packing items within feasible time intervals (determined by network conditions) into bins (network resources) while optimizing efficiency and reliability.

Most previous work on the Dynamic TCP Acknowledgement problem has focused on minimizing the *total* delay (i.e., the f_{sum} objective), and the f_{max} objective has received comparatively less attention, despite its importance in ensuring fairness and responsiveness. The f_{max} objective captures the gap between the actual completion time of a packet and the earliest possible completion time. Controlling such delays is especially critical in time-sensitive systems [77].

Example 3. *Consider input $L = \langle a_1 = 1, a_2 = 2, a_3 = 5, a_4 = 7, a_5 = 10, a_6 = 14\rangle$ of the Dynamic TCP Acknowledgement problem. See Figure 3.2 for illustration. Consider a solution that sends acknowledgements at times $t_1 = 5, t_2 = 12$, and $t_3 = 15$. We investigate three different objectives for this solution as follows:*

- *f_{sum}: Upon acknowledging all the pending packets at time $t_1 = 5$, the $f_{sum}(1)$ is equal to the sum of costs incurred by any packet acknowledged at t_1, which is $(5-1)+(5-2)+(5-5) = 7$. Similarly, $f_{sum}(2) = (12-7)+(12-10) = 7$ and $f_{sum}(3) = 15-14 = 1$. Therefore, the cost of the proposed solution under this objective is $\rho \times 3 + (1-\rho) \times (7+7+1)$, which is 9 for the specified value of $\rho = 0.5$.*

- *f_{max}: Upon acknowledging all the pending packets at time $t_1 = 5$, the $f_{max}(1)$ is equal to the maximum cost incurred by any packet acknowledged at t_1, which is $5-1 = 4$. Similarly, $f_{max}(2) = 12-7 = 5$ and $f_{max}(3) = 15-14 = 1$. Therefore, the cost of the proposed solution under this objective is $\rho \times 3 + (1-\rho) \times (4+5+1)$, which is 6.5 for the specified value of $\rho = 0.5$.*

- *The third objective, which is only studied in the work of [77], concerns the maximum delay over all the packets of the sequence. Here, the latency cost of the proposed solution under this objective would be $\max\{(5-1), (5-2), (5-5), (12-7), (12-10), (15-14)\} = 5$. Therefore, the total cost would be $\rho \times 3 + (1-\rho) \times 5 = 4$.*

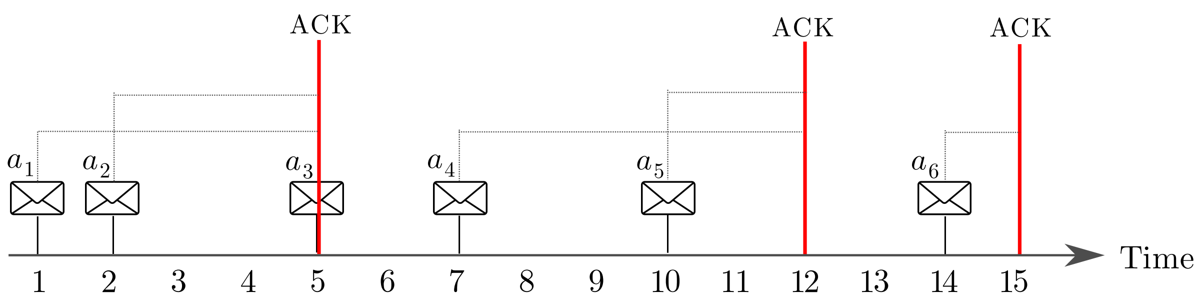


Figure 3.2: An example illustrating DYNAMIC TCP ACKNOWLEDGEMENT. The red line indicates the acknowledgement times for pending packets, while the dotted horizontal lines show the time that each packet incurs delay.

Chapter 4

Results for the Timely Bin Packing Problem

4.1 Overview & Summary of Results

In this chapter, we will present our results concerning the Timely Bin Packing problem. Table 4.1 presents a summary of our results. The chapter is organized as follows:

- In Section 4.2, we study the Timely Bin Packing problem in the generic setting, that is, without any assumption about the sizes or availability intervals of items. We will consider both deadline-aware and deadline-oblivious settings.
  - In Section 4.2.1, we provide impossibility results for the deadline-aware variant. In particular, we show that (i) no deterministic online algorithm can achieve a competitive ratio better than 1.5 (Theorem 4.2.1) and (ii) no randomized algorithm can achieve a competitive ratio better than $1.5 - 1/(2n - 4)$ for an input of length n (Theorem 4.2.2). Previously, only a weaker lower bound of 1.2 was known for deterministic algorithms [1].
  - In Section 4.2.2, we provide impossibility results for the deadline-oblivious setting. In particular, we provide (i) a simple and intuitive argument that shows no deterministic online algorithm can achieve a competitive ratio better than 2 (Theorem 4.2.3) and (ii) a slightly more complicated argument that extends the results to randomized algorithms; that is, no (randomized or deterministic) algorithm can achieve a competitive ratio better than 2 (Theorem 4.2.4). No general lower bound existed for the deadline-oblivious setting previously.
  - In Section 4.2.3, we revisit an algorithm proposed by Khattar [1], named Pack-at-

Deadline. This algorithm packs *all* available items every time any item reaches its deadline. This means that the algorithm is well-defined in both deadline-oblivious and deadline-aware settings. An upper bound of 2 for the competitive ratio of Pack-at-Deadline is claimed in [1]. We provide a lower bound of 2.5 for the competitive ratio of Pack-at-Deadline (Theorem 4.2.5), establishing that the claimed upper bound of 2 is incorrect.

  – In Section 4.2.4, we present a new algorithm for TIMELY BIN PACKING. This algorithm, named Single-Pack-at-Deadline (SPAD), works similarly to Pack-at-Deadline, except that, upon reaching the deadline of an item x, it only packs x and possibly other items in a single bin, but unlike Pack-at-Deadline, it does not require packing all items. Note that the algorithm is well-defined in the deadline-oblivious (and thus deadline-aware) settings. We prove the competitive ratio of SPAD is at most 3 (Theorem 4.2.6), and further present instances which show the competitive ratio of SPAD is at least 2 (Theorem 4.2.7). This set of results indicates a potential advantage for SPAD over Pack-at-Deadline. Note that these results hold for both deadline-aware and deadline-oblivious settings.

  – In Section 4.2.5, we consider a relaxed setting of the problem, which allows packing parts of an item into one bin and other parts into other bins. This variant, called *fractional timely bin packing*, is used as a medium to establish the presence of an algorithm, called Discrete-Greedy, for the deadline-aware setting of the problem. We prove that Discrete-Greedy has a competitive ratio of at most 2 (Theorem 4.2.10), establishing an advantage over the Pack-at-Deadline and SPAD in the deadline-aware setting.

- In Section 4.3, we consider a semi-online setting under the advice model, where an online algorithm has access to a bounded number of advice bits.

  – In Section 4.3, we show that advice of size $\mathcal{O}(n \log \text{OPT})$ is always sufficient and sometimes necessary to achieve optimality for the TIMELY BIN PACKING problem, where OPT specifies the cost of an optimal solution (Theorem 4.3.1).

  – In Section 4.3.1 we consider inputs with bounded "depth", i.e., instances in which the number of availability intervals that intersect at any given time t is bounded by a constant. We show that in this setting, a optimal algorithm exists that uses only $\mathcal{O}(n)$ bits of advice (Theorem 4.3.2).

  – In Section 4.3.2 we give a lower bound for the competitive ratio of any algorithm that uses sublinear advice. In particular, we use a reduction from BINARY STRING

GUESSING (see Section 1.4.3) to prove that any algorithm with sublinear advice has a competitive ratio of at least $1.5 - \epsilon$ for every $\epsilon > 0$ (Theorem 4.3.4). This result holds even for inputs of bounded depth and shows the optimality of our algorithms in Section 4.3.1 for these inputs.

- In Section 4.4 we focus on the offline TIMELY BIN PACKING problem, where the entire input sequence is known in advance.
  - In Section 4.4.1 we formally show that the offline TIMELY BIN PACKING problem is NP-hard (Observation 4.4.1) by a reduction from the classic offline bin packing.
  - In Section 4.4.2, we investigate the offline TIMELY BIN PACKING problem for instances where all item sizes are greater than 1/3. We show that in this setting, the problem can be solved optimally using a maximum matching algorithm (Theorem 4.4.1).
- In Section 4.5, we explore restricted variants of the online TIMELY BIN PACKING problem, where structural assumptions are made about the item sizes, availability intervals, or arrival times. Specifically, we establish the following results. See Table 4.2 for a summary.
  - In Section 4.5.1 we consider the case where all items have the same size (i.e., uniform size setting). We show that in the deadline-aware variant of this setting, the online constraint is not restrictive and, in particular, a simple greedy online algorithm achieves optimality (Observation 4.5.1). For the deadline-oblivious uniform size setting, on the other hand, we establish a lower bound of 2 for the competitive ratio of any (randomized or deterministic) algorithm (Theorem 4.5.1).
  - In Section 4.5.2, we explore a setting where item sizes are restricted to come from a sub-interval of $(0, 1]$ (this generalizes the results in Section 4.5.1 where the sub-interval is a single point). When item sizes come from $[1/4, 1]$, we present an algorithm with a competitive ratio of at most 2 (Theorem 4.5.2). Moreover, we prove that even when all item sizes are strictly less than $1/k$ for a large integer value k, no algorithm can achieve a competitive ratio better than 2 (Theorem 4.5.4). This means that, unlike the classic bin packing problem, where inputs formed entirely by small items are considered to be easy to pack, the TIMELY BIN PACKING problem is still challenging for these instances.
  - In Section 4.5.3, we consider *monotone instances* where sorting items by their arrival times results in the same ordering as sorting them by their deadlines (i.e.,

if one item arrives earlier than another, it must have an earlier deadline too). For this setting, we show that the Pack-at-Deadline algorithm achieves a competitive ratio of at most 2 (Theorem 4.5.5).

- In Section 4.5.4, we further relax the setting from monotone instances to the case where each item has a unit-length availability interval.
  * If there is no further assumption, we can prove a lower bound of 1.5 for the competitive ratio of any deterministic and randomized algorithm (Theorems 4.5.6 and 4.5.10).
  * We impose an additional restriction that arrivals occur only at integer times, meaning each item's availability interval is of the form $[x, x+1]$ for some integer x. Under this setting, we design a randomized algorithm with a competitive ratio of at most 1.5 (Theorem 4.5.8). Also, we establish a lower bound of 4/3 for the competitive ratio of deterministic algorithms (Theorem 4.5.9) and a lower bound of 1.2 for the competitive ratio of randomized algorithms (Theorem 4.5.10).
  * We consider the case where item sizes are limited to $\{\alpha, 1-\alpha\}$ and availability intervals are unit length. For this setting, we present a randomized algorithm with a competitive ratio of at most 1.5 (Theorem 4.5.11). We also establish a lower bound of 1.5 for the competitive ratio of any deterministic algorithm (Theorem 4.5.12) and a lower bound of 1.25 for the competitive ratio of any randomized algorithm (Theorem 4.5.13).

4.2 Results for Generic Online Timely Bin Packing

In this section, we first study the two variants of the TIMELY BIN PACKING problem: deadline-aware and deadline-oblivious (see Definition 3.1.1).

4.2.1 Impossibility Results for Deadline-Aware Variant

We present our negative results for the deadline-aware setting. We first show a simple lower bound of 1.5 for the competitive ratio of deterministic algorithms, followed by a slightly weaker lower bound of $1.5 - 1/(2n-4)$ for randomized algorithms, where n is the number of input items.

Theorem 4.2.1. *No deterministic deadline-aware algorithm can achieve a competitive ratio better than* 1.5 *in the online* TIMELY BIN PACKING *problem.*

Table 4.1: Summary of the contributions.

		Best Existing	Our Results
Online Deadline-Aware	Upper Bound	Claimed 2 [1]	2 (Theorem 4.2.10)
	Lower Bound	1.2 [1]	1.5 (Deterministic, Theorem 4.2.1; Randomized, Theorem 4.2.2)
Online Deadline-Oblivious	Upper Bound	Claimed 2 [1]	3 (Theorem 4.2.6)
	Lower Bound	1.2 [1]	2 (Deterministic, Theorem 4.2.3; Randomized, Theorem 4.2.4)
Advice	Upper Bound	–	$\Omega(n \log(\text{OPT})) \to 1$ (Observation 4.3.1)
	Lower Bound	–	$o(n) \to 1.5 - \epsilon$ (Theorem 4.3.4)
Offline	Upper Bound	–	2-approximation (Corollary 4.2.12)
	Lower Bound	NP-hard (Observation 4.4.1)	–

Proof. At timestep 0, the adversary releases 4 items $A = (s = 0.5, r = 0, d = 3), B = (0.3, 0, 1), C = (0.26, 0, 5)$ and $D = (0.26, 0, 5)$. See Figure 4.1 for an illustration.

The algorithm must pack the item B before its deadline, timestep 1. There are two cases to consider, depending on how an algorithm packs B with other times.

(i). If B gets packed with A, there would be no more space for the other items in the bins that they share. The adversary then sends another item $E = (0.5, 2, 3)$. The algorithm cannot pack the new item with both C and D in one bin. Thus, it needs at least 2 more bins to pack the items (3 bins in total). In contrast, the optimal solution packs A and E together in one bin, and B, C, and D in another. Thus, the competitive ratio would be 3/2 in this case.

(ii). Otherwise, if A does not get packed in the same bin as B, the other two items C and D can be in its bin. The adversary adds a new item $F = (0.01, 4, 5)$. As this new item has no intersection with A, the algorithm again needs 2 more bins (3 in total). On the other hand, the optimal solution could pack A and B together in one bin and F, C, and D in another. Therefore, the competitive ratio in this case is 3/2 as well. $\square$

We can achieve a slightly weaker lower bound of 1.25 for the competitive ratio of randomized algorithms via a slight modification of the same adversarial input as in the above proof. In this case, the adversary sends a new item (E or F) non-deterministically. However, we do not elaborate on this proof, as we will establish a better lower bound of 1.5 for the randomized case as we discuss in the next theorem.

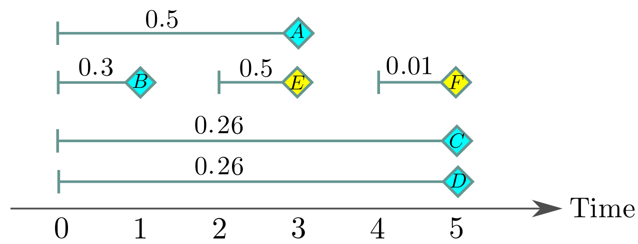


Figure 4.1: The instance first contains items A, B, C and D. Sending either E or F creates two possible instances with different optimal solutions. As the algorithm does not see which one of E and F is present at the deadline of B, it cannot perform optimally.

Theorem 4.2.2. *No randomized deadline-aware algorithm can achieve a competitive ratio better than* $1.5 - 1/(2n-4)$ *in the online* Timely Bin Packing *problem, where* n *is the number of input items.*

Proof. We use Yao's min-max principle (See Definition 1.4.4), which studies a deterministic algorithm faced with a randomized input.

We define a probability distribution over a family of input sequences as follows. Let $k = n-1$, and consider an instance with one item $c = (s_1 = 1/k, r_1 = 0, d_1 = 0)$ and $k-1$ other items $a_i = (1/k + i\epsilon, 0, 2i)$ for $i \in [k-1]$ and a constant $\epsilon \ll 1/k$. So far, the adversary has released k fixed items. See Figure 4.2 for an illustration of the instance.

The adversary then selects and send one final item b_{i^*} among k possible items of the form $b_i = (1 - (1/k + i\epsilon), 2i-1, 2i)$ for $i \in [k-1]$ uniformly at random.

This extra item b_{i^*} can be packed with only one item that has a size of $1/k + i^*\epsilon$: larger items cannot be packed with it as their sum would be larger than 1, and the availability intervals of smaller items does not intersect that of b_{i^*}.

The optimal algorithm, in this scenario, packs item c with all other items available at timeslot 0 except b_{i^*}, which will be packed with the last item that will arrive at time $2i-1$. Thus, we have Opt $= 2$. Any deterministic algorithm, on the other hand, must leave at least one of the items at timeslot 0 unpacked. With probability $(k-2)/(k-1)$, this item is not b_{i^*}, and the algorithm must use at least 3 bins. Therefore, the cost of the algorithm is at least $3.(k-2)/(k-1) + 2/(k-1) = 3 - 1/(k-1) = 3 - 1/(n-2)$. The competitive ratio would be at least $3/2 - 1/(2n-4) = 1.5 - 1/(2n-4)$. ☐

Observation 4.2.1. *The lower bound of* $1.5 - 1/(2n-4)$ *for the competitive ratio of randomized algorithms (Theorem 4.2.2) holds even when the item sizes are less than* c *for every constant* c.

Proof. We use a similar probability distribution of worst-case inputs as in the proof of Theorem 4.2.2. However, for a given constant c, we set k large enough so that all items

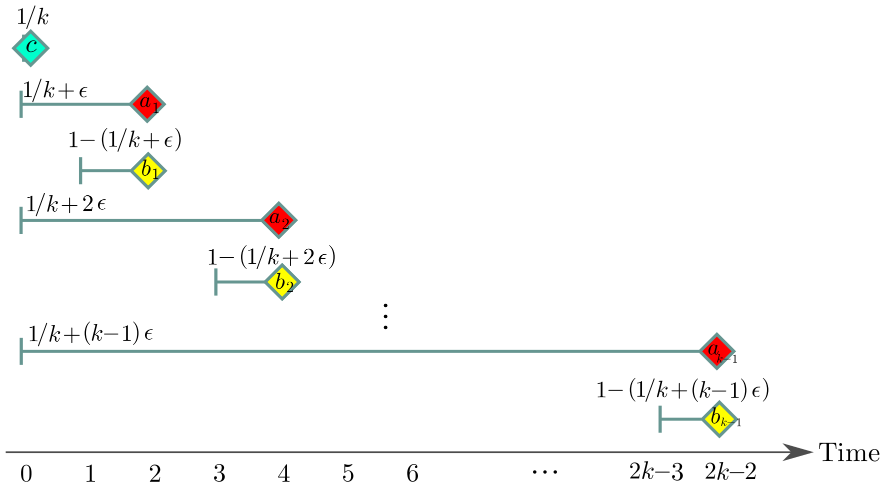


Figure 4.2: The blue and red items exist in the instance. The adversary randomly sends one of the yellow items with an equal probability. This new yellow item x with size $1-(1/k+x\epsilon)$ can only be packed with a red item of size $1/k+x\epsilon$.

except the last one (b_i's), have a size less than c. For the last one also, one can observe that by breaking down the last item into several items of sizes at most c, the argument above remains unchanged. In this way, the lower bound holds even when the item sizes are less than c for every constant c. □

4.2.2 Impossibility Results for Deadline-Oblivious Variant

In what follows, we first establish a lower bound of 2 for the competitive ratio of every deterministic algorithm and then extend this result to randomized algorithms via a slightly more complicated argument. The standing assumption in this section is that the algorithms are oblivious (or do not consider) the item deadlines.

Theorem 4.2.3. *No deterministic deadline-oblivious algorithm can achieve a competitive ratio better than 2 in the online* Timely Bin Packing *problem.*

Proof. In what follows, we will describe a worst-case σ. Refer to Figure 4.3 for an illustration. σ contains m trigger items: for $i\in[m]$, let $a_i=(s_i=1/2-(2i-1)\epsilon, r_i, d_i)$, where $r_i=d_i=2i-2$. In Figure 4.3, trigger items are blue items.

Moreover, we have two more types of items.

- For each $i\in[m]$, let $b_i=(1/4+i\epsilon, 2i-2, 2i-1)$ (red items in the figure).
- For each $i\in[m]$, let $c_i=(1/4+i\epsilon, 2i-2, 2i+1)$ (yellow items in the figure).

The optimal in each phase i, always packs the trigger a_i with b_i and c_{i-1} (if exists). Note that a_i, b_i, and c_i do not fit in the same bin, as both sequences of b_i and c_i sizes are increasing. It also needs an extra bin for the last item c_m. Thus, $\text{OPT} = m + 1$.

Since we assume deterministic deadline-oblivious algorithms A and the size of any b_i is equal to that of c_i (both are $1/4 + \epsilon$), it is safe to assume the algorithm cannot distinguish between b_i and c_i. Therefore, we can assume that A, for each phase i, packs the trigger a_i with c_i and also packs the b_i individually. This is because when we reach to a trigger, A must pack the trigger a_i with either the b_i or c_i, and as the algorithm cannot differentiate between these two items, whichever it chooses to not pack, the adversary considers the deadline of that item to be compatible with the first type (b_i). Therefore, A uses $2m$ bins. Hence, the competitive ratio is $\frac{2m}{m+1}$, which converges to 2 for large m. □

The above proof can be extended to randomized algorithms, except that we cannot assume that a_i is always packed with c_i (We can assume that a_i is packed with c_i with a probability of 0.5, which degrades the resulting lower bound). To resolve this issue, we can adjust the input in a way that a_i must be packed with c_i as before, except that c_i is accompanied by $k - 2$ other items of similar size and arrival time, and the algorithm cannot distinguish c_i among these $k - 1$ items.

Theorem 4.2.4. *No randomized deadline-oblivious algorithm can achieve a competitive ratio better than 2 in the online* TIMELY BIN PACKING *problem.*

Proof. Consider an instance with m trigger items as follows. For $i \in [m]$, trigger $a_i = (s_i = 1/k - ((k-1)i - 1)\epsilon, r_i, d_i)$, where $r_i = d_i = 2i - 2$.

Moreover, we have two more types of items as follows:

- For each $i \in [m]$, we have $k - 2$ identical items $b_i^j = (1/k + i\epsilon, 2i - 2, 2i - 1)$ for each $j \in [k - 2]$.
- For each $i \in [m]$, let $c_i = (1/k + i\epsilon, 2i - 2, 2i + 1)$.

The optimal algorithm, in each phase i, packs the trigger item a_i with all b_i^js and c_{i-1} (if exists). It also needs an extra bin for the last item c_m. Thus, $\text{OPT} = m + 1$.

Any randomized deadline-oblivious algorithm, on the other hand, packs the trigger a_i with c_i with probability at least $\frac{k-2}{k-1}$, and in this case, it has to pack some b_i^j individually. This is because, when we reach a trigger item a_i, the algorithm cannot pack a_i with all of the items in the phase, namely all b_i^j's and c_i. And because the algorithm cannot differentiate between these items b_i^j's and c_i, thus the algorithm leaves one of them unpacked with the same chance. Therefore, it uses $m(1 + \frac{k-2}{k-1}) \approx 2m$ bins on expectation. Hence, the competitive ratio tends to 2 if we tend m and then k to infinity. □

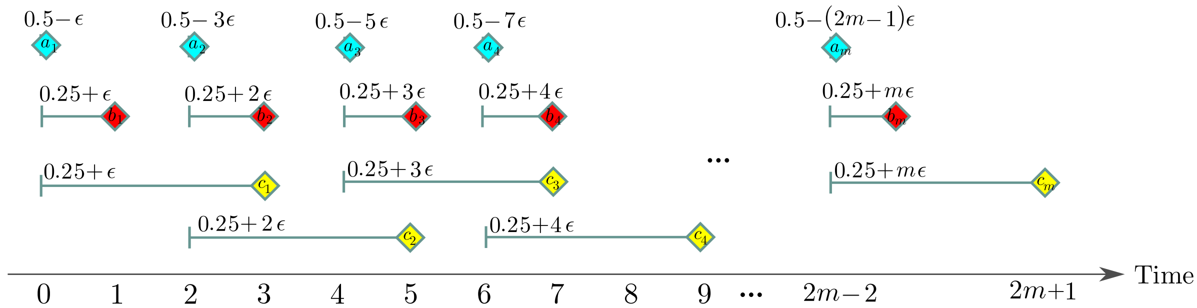


Figure 4.3: The items a_i's (trigger items), b_i's, and c_i's for $i \in [m]$ are shown in blue, red, and yellow, respectively.

4.2.3 Pack-at-Deadline Algorithm

Recall that the Pack-at-Deadline algorithm of [1] packs *all* items upon the deadline of any pending item (it does not look at the deadline upon arrival of items and hence is a deadline-oblivious algorithm). In this section, we establish a lower bound of 2.5 for the competitive ratio of the Pack-at-Deadline algorithm, which is an improvement over the lower bound 10/6 of [1]. It also proves that their upper bound of 2 is incorrect.

Theorem 4.2.5. *The Pack-at-Deadline algorithm has a competitive ratio of at least* 2.5 *in the online* Timely Bin Packing *problem.*

Proof. The idea is illustrated in Figure 4.4. The instance consists of the following components.

- Set P: for $i \in [m]$, let $p_i = (s_i = 0.5 + \epsilon, r_i = 0, d_i = 3m)$ (red items in the figure).
- Gadgets: There are $m/2$ gadgets, where each gadget $i \in [m/2]$ contains:
  - Trigger item: One item $u_1 = (0.5 - \epsilon, 3i + 1, d_i = 3i + 2)$ (pink color is the figure).
  - Filler items: Call $z = \frac{0.5+2\epsilon}{\epsilon}$. For $j \in [z]$ let $v_i^j = (\epsilon, 3i, 3m)$. Each set of v_i's contributes a total size of $0.5 + 2\epsilon$ (yellow color in the figure).
- Set Q: for $i \in [m/2]$, let $q_i = (\epsilon, 2m + 2i, 2m + 2i + 1)$ (blue color in the figure).
- Item A: A single item $a_1 = (\epsilon, 1, 2)$ (green color in the figure).

The Pack-at-Deadline algorithm, when it reaches the deadline of A, there are m items of size $0.5 + \epsilon$ available. The algorithm packs each of these items in a separate bin because the total size of any two of them is larger than 1. Thus, the algorithm uses m bins and the ϵ item can be packed in one of them.

In each gadget i, there is a trigger item with size $0.5 - \epsilon$ that reaches its deadline at $3i + 2$ and causes the algorithm to pack all the items of that gadget. As the total sum of the filler

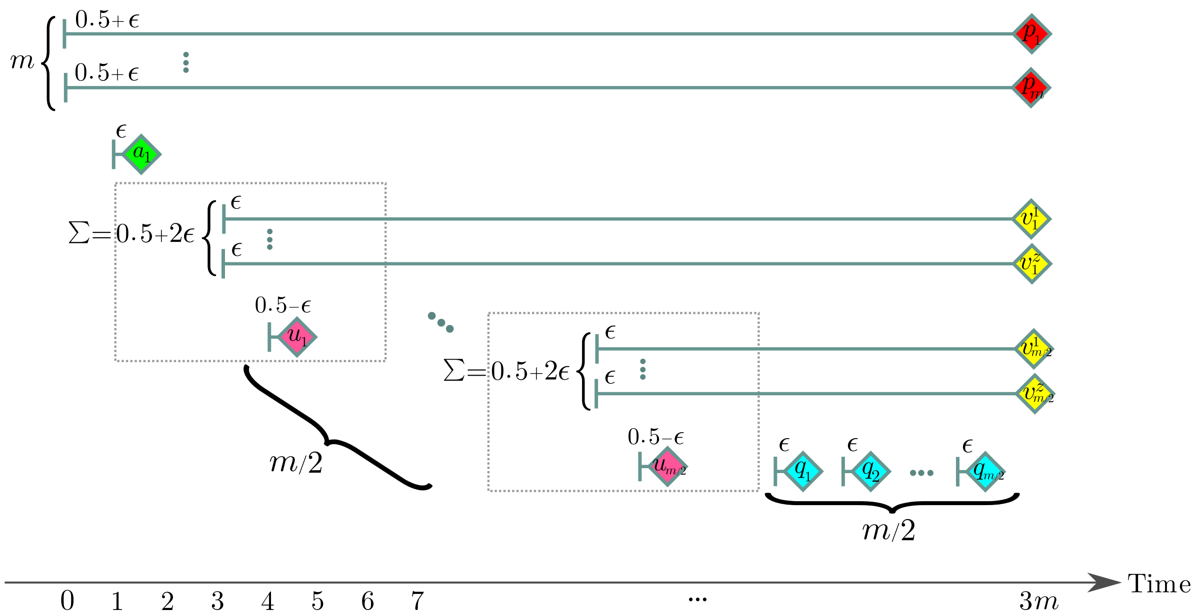


Figure 4.4: Here, P and Q items are shown with color red and blue, respectively. Gadgets are shown in a rectangle, each having one trigger item of size $0.5 - \epsilon$ and filler items of size ϵ with the total sum of $0.5 + 2\epsilon$. There is another ϵ item named A arriving at time 1.

items of each gadget is $0.5 + 2\epsilon$, all but one of them can be packed with the trigger in one bin, and the one remaining filler item will be packed in a separate bin. Thus, all the gadgets together need m bins to be packed.

The Q items do not intersect with any available items in their deadline, and each of them needs one bin, resulting in the usage of $m/2$ bins. Altogether, the algorithm uses $m + m + m/2 = 2.5m$ bins for this instance.

The optimal solution packs A in a single bin. Then it can pack each trigger item of gadgets at its deadline with one of the P items, and it uses $m/2$ bins for these items. Note that $m/2$ items of P will be remaining. Then, each of the Q items can be packed with the remaining P items and the filler items of one gadget, so that the bin becomes full. Four ϵ items will remain in each gadget that all can be packed in one bin at time $3m$. Therefore, the optimal solution needs $1 + m/2 + m/2 + 1 = m + 2$ bins to pack all items. We can conclude that the competitive ratio of Pack-at-Deadline is at least:

$$\lim_{m\to\infty} \frac{2.5m}{m+2} = 2.5.$$

□

The result above invalidates a claimed upper of 2 for the competitive ratio of Pack-at-Deadline, claimed in [1]. The main issue with the proof is Lemma 4 of [1], which attempts to lower bound the cost of the optimal solution.

4.2.4 Single-Pack-at-Deadline Algorithm

Our lower bound argument for the competitive ratio of Pack-at-Deadline (Theorem 4.2.5) reveals that packing all items upon a single item's deadline is suboptimal. A more natural approach is to pack only the item whose deadline has reached, along with other items, and try to pack that single bin as full as possible with the available items. We call this new algorithm Single-Pack-at-Deadline (SPAD). More precisely, suppose the deadline of an item a_i is at time t. SPAD forms a single bin B at time t and packs a_i in B. Moreover, it selects a subset of items from the set of available items whose total sum is as large as possible and yet at most $1 - s_i$. We note that SPAD works without knowledge of item deadlines and hence can be applied in both oblivious and deadline-aware settings.

Theorem 4.2.6. *The SPAD algorithm achieves a competitive ratio of at most* 3 *for the online* Timely Bin Packing *problem in deadline-oblivious (and thus also deadline-aware) settings.*

Proof. We generally use two types of lower bounds for the cost of the optimal solution. First, we know that the cost of the optimal solution is no smaller than the maximum number of items whose availability intervals are pairwise disjoint. This is because we need at least one bin for each of these items. Second, the optimal solution must be at least as large as the total size of all items because each bin can reduce the total size of the remaining items by at most 1.

Next, consider the bins of SPAD. Let *big bins* be the bins with a level of more than 1/2, and the *small bins* be the remaining bins of SPAD. Consider two small bins b and b'. Suppose they are triggered by the deadline of items x and y, respectively (recall that SPAD packs a bin only when it reaches an item's deadline). The availability intervals of x and y must be disjoint. This is because if one of them were available on the deadline of the other one, SPAD would have packed both of them together as both b and b' have a level of less than or equal to 1/2. Therefore, the items that triggered small bins are pairwise disjoint. As a result, based on our first lower bound, the number of small bins is at most Opt.

On the other hand, the number of big bins is at most twice the number of Opt because of our second lower bound for the optimal solution. Precisely, if S denotes the total size of items in the input, the cost of Opt is at least S, and the number of big bins is at most $2S$. We can conclude that, overall, SPAD uses at most Opt + 2Opt = 3Opt bins. □

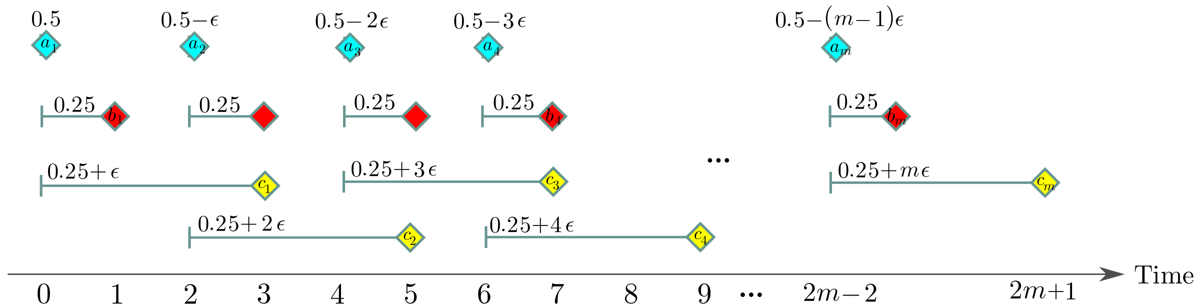


Figure 4.5: The items a_i's (trigger items), b_i's, and c_i's for $i \in [m]$ are shown in blue, red, and yellow, respectively.

Theorem 4.2.7. *The SPAD algorithm has a competitive ratio of at least 2 in the online* Timely Bin Packing *problem.*

Proof. In what follows, we will describe a worst-case σ. Refer to Figure 4.5 for an illustration. Consider an instance with m trigger items, for $i \in [m]$, $a_i = (s_i = 1/2 - (i-1)\epsilon, 2i-2, 2i-2)$. In Figure 4.5, trigger items are blue items.

Moreover, we have two more types of items.

- For each $i \in [m]$, let $b_i = (1/4, 2i-2, 2i-1)$ (red items in the figure).
- For each $i \in [m]$, let $c_i = (1/4 + i\epsilon, 2i-2, 2i+1)$ (yellow items in the figure).

The optimal solution in each phase i packs the trigger a_i with b_i and c_{i-1} (if it exists). It also needs an extra bin for the last item b_n. Thus, Opt $= m+1$. SPAD, on the other hand, in each phase packs the trigger a_i with c_i and also packs b_i individually. Therefore, it uses $2m$ bins. Hence, the competitive ratio is 2 if we tend n to infinity. □

As a first step toward proving a stronger upper bound, we consider a restricted setting where item sizes are less than $1/6$. Moreover, we assume items are grouped such that availability intervals from different groups do not overlap, while within each group, all intervals share a common time point.

Lemma 4.2.8. *Consider the deadline-oblivious setting of the* Timely Bin Packing *problem when the item sizes are less than* $1/6$*. Moreover, assume items are grouped such that availability intervals from different groups do not overlap, while within each group, all intervals share a common time point. Under these assumptions, the SPAD algorithm achieves a competitive ratio of at most* 1.5

Proof. Fix any input sequence as described, and let Alg denote the cost of SPAD for such input. Given that the availability intervals of items in different groups do not overlap, proving $\text{ALG} \leq 1.5\text{OPT}$ in every partition proves the upper bound of 1.5 for the whole instance.

We use a *weighting function* method, which is a standard tool in the analysis of bin packing algorithms [10]. First, for the cost of the algorithm for items of each group, we prove $\text{ALG} \leq 1.2\text{OPT}+1$. For that, for an item of size x, let the *weight* of x be $w(x) = 6x/5$. The total size of items in every bin of the optimal solution is at most 1, and their total weights sum to at most 6/5. On the other hand, every bin (except possibly the last one) in SPAD has a size of at least 5/6; otherwise, another item could fit into it. Thus, the weight of each bin in SPAD is at least $\frac{5}{6} \cdot \frac{6}{5} = 1$. So, if W denotes the total weight of items in a group, we can write $\text{ALG} \leq W + 1$ while $\text{OPT} \geq 5W/6$, which gives $\text{ALG} \leq 1.2\text{OPT} + 1$.

If $\text{OPT} > 3$, the above equation would give the upper bound of 1.5 for SPAD. Suppose $\text{OPT} \leq 3$. Order bins in the packing of OPT and SPAD by the earliest deadline among items packed in them. For example, the first bin would be the bin that contains the item with the earliest deadline. By the definition of SPAD, the total size of items in the first bin of SPAD is at least equal to that of the first bin of OPT.

There are three cases for the optimal solution. We describe each case individually.

(i). $\text{OPT} = \mathbf{1}$: In this case, as described above, SPAD has also exactly one bin.

(ii). $\text{OPT} = \mathbf{2}$: Because the total size of items in the first bin of SPAD is larger than or equal to that of the first bin in the optimal solution, the total size of the remaining items in SPAD is also less than the level of the second bin in the optimal solution. Thus, it is less than one and can be fit in a single bin. Therefore, SPAD also uses 2 bins in this case.

(iii). $\text{OPT} = \mathbf{3}$: We know that the average level of the bins other than the first bin in SPAD is greater than 1/2. This is because there cannot be two bins with levels of less than 1/2, as SPAD could have combined them into a single bin (i.e., the total size of the items in the bin with a size less than 1/2 must be larger than the empty part of the other bins). Furthermore, the total size of the items that are not in the first bin of the optimal solution is at most 2 because they are packed in two bins in the optimal solution. As a result, SPAD cannot use more than 3 bins after delivering its first bin because otherwise, the sum of the remaining items would be larger than $4/2 = 2$, which is a contradiction. Therefore, SPAD uses at most 4 bins.

To conclude, in all cases, we proved $\text{ALG} \leq 1.5\text{OPT}$ for each partition, which implies the upper bound of 1.5 for the whole instance. $\square$

4.2.5 Fractional Timely Bin Packing

In this section, we focus on the deadline-aware variant of the Timely Bin Packing problem and present an algorithm with a competitive ratio of at most 2.

First, consider a relaxed variant of the Timely Bin Packing problem, which we call the Fractional Timely Bin Packing, where an algorithm is allowed to break items into pieces and pack each piece into a different bin. One can think of items as liquids in this variant.

It is clear that every solution to the original problem is also a solution to the fractional variant of the problem. Thus, the optimal solution to the fractional variant of the problem provides a lower bound on the optimal solution to the discrete variant. That is why we elaborate on this variant in this section; this will become clear later.

Consider the greedy algorithm for the Fractional Timely Bin Packing problem, which works as follows.

Greedy Algorithm for Fractional Timely Bin Packing. When the deadline for at least one item arrives, open a bin B and pack the available items one by one in the increasing order of their deadline (i.e., the ones with earlier deadlines are packed first). This continues until either there are no more pending items or the next item does not fit into B. In the latter case, place a fraction of the last item in B such that B is fully packed (the total size of items in B becomes 1).

Lemma 4.2.9. *The Greedy algorithm is optimal for the online Fractional Timely Bin Packing problem.*

Proof. Let S be an optimal packing that differs from that of the Greedy algorithm. We will show that the packing of S can be converted to that of the Greedy algorithm without increasing its cost. It is safe to assume S only delivers bins at the deadline of some item (otherwise, it can postpone delivering bins without increasing its cost). Let t be the first deadline where the S and the packing of Greedy differ. That is, S delivers a bin B_o which differs from the bin B_g delivered by the Greedy algorithm at time t. That means there is a portion of an item y with the earliest deadline with size k that does not go to B_o, and some portions of other items with the same total size k and later deadlines are indeed placed in B_o. The portion of item y of size k then must be in other bins of S. Given that y has the first deadline among them, it is possible to re-position portions of y to the first bin and portions of other items from the first bin to the later bins. Therefore, this change does not increase the cost of the optimal solution. Repeating this argument will eventually convert

S to the packing of Greedy without increasing its cost. We can conclude that the Greedy algorithm's solution is optimal. $\square$

Next, we devise an online algorithm, named Discrete-Greedy, which solves the original Timely Bin Packing by simulating the Greedy algorithm for the fractional setting.

Discrete-Greedy Algorithm for Timely Bin Packing. Discrete-Greedy proceeds similarly to the Greedy algorithm for the fractional setting. Recall that, at any deadline t, the Greedy algorithm packs items in the increasing order of their deadlines in a bin B_1, except that the last item x packed in B_1 may be broken into two pieces such that B_1 is fully packed, while the remaining portion of x is placed in another bin B_2 (and the algorithm proceeds to pack B_2 similarly until all items with deadline t are packed). Now, given that in the original setting, we are not allowed to break x, Discrete-Greedy packs x into a separate bin B_1' and delivers (closes) that bin, and proceeds by packing B_2 as before.
Note that the algorithm is obviously aware of the item deadlines as it uses them to decide which items to pack at each given time.

Example 4. *The input is* $L = \{a_1 = (s_1 = 0.5, r_1 = 1, d_1 = 7), a_2 = (0.8, 2, 10), a_3 = (0.4, 3, 5), a_4 = (0.3, 8, 11), a_5 = (0.1, 13, 14)\}$. *Figure 4.7 shows an illustration of the input. At time t, Discrete-Greedy needs to pack* a_3. *For that, it sorts available items in the increasing order of their deadlines, that is, in order* (a_3, a_1, a_2). *Greedy opens a bin* B_1, *which contains* a_3, a_1, *and a fraction of size* 0.1 *of* a_2, *while deferring packing the remaining part of* a_2 *(with size 0.7). On the other hand, Discrete-Greedy packs two bins* B_1 *and* B_1' *at time t, one containing* a_3 *and* a_1, *and the other containing* a_2.

Theorem 4.2.10. *The Discrete-Greedy algorithm achieves a competitive ratio of at most* 2 *for the online deadline-aware Timely Bin Packing problem.*

Proof. In principle, Discrete-Greedy simulates Greedy for Fractional Timely Bin Packing, except that whenever Greedy delivers a fractional bin, Discrete-Greedy delivers at most two discrete bins. Therefore, the number of bins in the Discrete-Greedy packing is at most twice that of Greedy for Fractional Timely Bin Packing. In other words, each bin of Greedy that contains some fractional items projects to two bins in Fractional Timely Bin Packing. We further observe that the cost of Opt in the fractional setting, Opt_f, is no more than the cost of Opt in the original setting (any solution for the original setting is valid for the fractional setting too). We can conclude that, the cost of Discrete-Greedy for the original setting is at most twice the cost of Greedy for the factional setting, and thus by Lemma 4.2.9, twice the cost of the optimal fractional setting, and consequently at most twice the cost of optimal solution for the original setting.

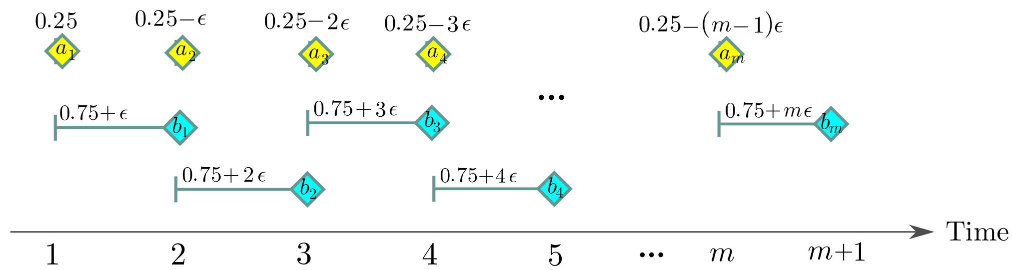


Figure 4.6: The trigger items are shown by color yellow and others are shown by color blue.

□

The following theorem shows that the upper bound in the above theorem is indeed tight. That is, the competitive ratio of Discrete-Greedy is not better than 2.

Theorem 4.2.11. *The competitive ratio of the Discrete-Greedy algorithm is at least* 2*.*

Proof. Consider an instance with m trigger items, where for every $i \in [m]$, $a_i = (s_i = 1/4 - (i-1)\epsilon, r_i = i, d_i = i)$. Moreover, there are m items, where for each $i \in [m]$, $b_i = (3/4 + i\epsilon, i, i+1)$. See Figure 4.6 for an illustration of the instance.

The optimal solution packs the first trigger in a separate bin, and each subsequent trigger a_i with b_{i-1}. It also needs an extra bin for the last item b_m. Thus, $\textsc{Opt} = m + 1$. The Discrete-Greedy solution packs the trigger a_i with item b_i, resulting in the use of two discrete bins at each step by Discrete-Greedy. Therefore, for the cost of the algorithm, $\textsc{Alg} = 2m$. As m tends to infinity, the lower bound for the competitive ratio tends to 2. □

Corollary 4.2.12. *A* 2*-approximation algorithm exists for the offline* Timely Bin Packing *problem.*

Proof. The Discrete-Greedy algorithm, which achieves a competitive ratio of 2 for the online Timely Bin Packing problem (Theorem 4.2.10), also serves as a 2-approximation algorithm for the offline setting. □

4.3 Semi-Online Timely Bin Packing

In this section, we study online Timely Bin Packing with advice. Recall that, under advice setting, a benevolent oracle encodes some information about the input sequence in the form of advice on an advice tape; this information is available to the online algorithm at the beginning, and the algorithm is assumed to know the meaning of advice bits. Unlike prediction setting, it is assumed that the advice bits are correct, i.e., they encode what they

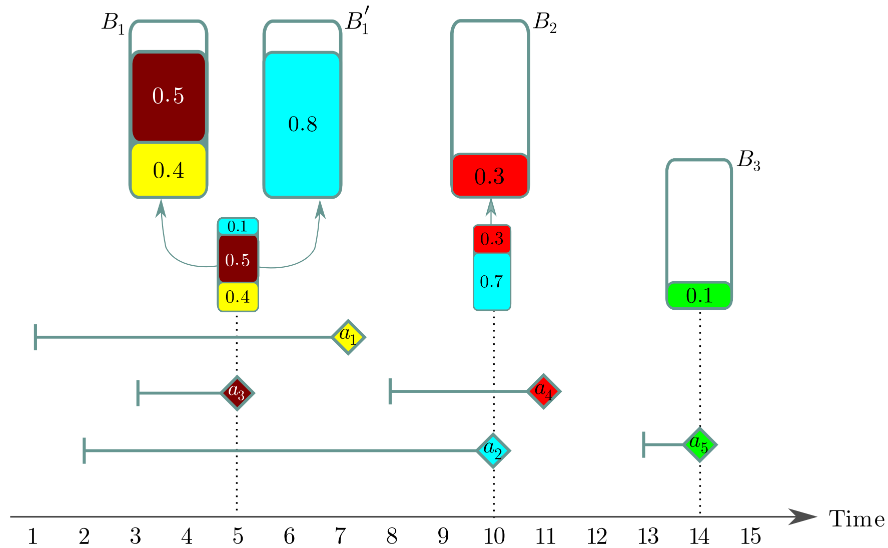


Figure 4.7: An illustration of Discrete-Greedy Algorithm.

are meant to. There is generally a correlation between the advice size and the performance: the larger the advice, the better the competitive ratio. See Section 1.4.3 for an overview.

For the complexity of the advice in the TIMELY BIN PACKING problem, we first present a trivial result that upper bounds the size of the advice required to achieve optimality. Let OPT be the cost of the optimal packing for an input. For each item x, it suffices to encode the bin where x is packed in the optimal solution. This means $\mathcal{O}(\log \text{OPT})$ bins per item, and thus we can conclude the following.

Observation 4.3.1. *An advice of size $\Omega(n \log \text{OPT})$ bits is sufficient to achieve an optimal solution for the TIMELY BIN PACKING problem.*

Indeed, the above upper bounds are tight. For that, we can reduce the online bin packing problem to the TIMELY BIN PACKING problem, where all items in the bin packing instance are translated to trigger items appearing at time 0 in an adversarial order that defines the bin packing instance (See Section 4.4.1 for details of such reduction). Meanwhile, in [69], it is proven that, when $\text{OPT} \in o(n)$, an advice of size $\Omega(n \log \text{OPT})$ is necessary to achieve optimality. Therefore, we can conclude the following.

Theorem 4.3.1. *Advice of size $\mathcal{O}(n \log \text{OPT})$ is always sufficient and sometimes necessary to achieve optimality for the TIMELY BIN PACKING problem.*

Roadmap. In what follows, we first extend the result of Theorem 4.3.1 to show that the problem is solvable optimally with $\mathcal{O}(n)$ bits of advice in the instances where, at each point in time, we have, at most, a constant number of items available. In other words, at most

k availability intervals include any time t, for some constant t. Second, we show that the above upper bound is tight. For that, we prove that no algorithm (neither deterministic nor randomized) with sublinear bits of advice can achieve a competitive ratio of better than $1.5 - \epsilon$, for every ϵ, which is the main result of this section.

4.3.1 Algorithm for Bounded Depth

In this section, we focus on a restricted setting of the problem where, at any time, at most k items are available for a constant k. We present an optimal solution encoding that uses $\mathcal{O}(n)$ bits of advice.

Theorem 4.3.2. *There exists a polynomial-time optimal algorithm for the online* Timely Bin Packing *problem using $\mathcal{O}(n)$ bits of advice.*

Proof. Fix an optimal solution S. As before, we may assume that S delivers bins only in item deadlines (otherwise, it can postpone packing bins to achieve this property).

For each item x in the input sequence, we write a bit on the advice tape that specifies whether x is the item with the earliest deadline in its bin in S. In other words, S delivers the bin of x at the deadline of x. We refer to this as the first part of the advice. Moreover, in the second part of the advice, for each bin B in S, delivered at time t, we encode k bits, one per each item available at time t, which specifies whether that item is packed in B. This is possible because of the assumption that at most k items are available at t.

Now, provided with the advice, an online algorithm can reproduce S as follows. Upon the deadline t of an item x, if the bit of x (in the first part of the advice) is set, we open a bin and check the k bits stored on the tape on the second part, to retrieve the available items that must be packed with x in the bin. The resulting bin is then delivered at time t. Therefore, a total of $\mathcal{O}(n+k|\textsc{Opt}|) = \mathcal{O}(n)$ advice bits (as k is constant and $|\textsc{Opt}| = \mathcal{O}(n)$) suffices to simulate the optimal packing. $\square$

4.3.2 Impossibility Result for Sublinear Advice

In this section, we start with a simple lower bound argument that shows any algorithm with sublinear advice cannot have a competitive ratio better than 1.25. Then, with a slightly more complicated argument, we improve the lower bound to $1.5 - \epsilon$, for any $\epsilon > 0$. As mentioned in Section 1.4.3, reductions from the Binary String Guessing problem, which asks for predicting the value of the next bit in a sequence of bits that is revealed online, are often used to establish lower bounds for the competitive ratio of various problems when the advice size is sublinear.

Here, we reduce the Binary String Guessing problem to the Timely Bin Packing problem to show a lower bound for the competitive ratio of Timely Bin Packing for *any* algorithm with sublinear advice cannot be better than 1.25-competitvie.

Lemma 4.3.3. *No algorithm with sublinear bits of advice can achieve a competitive ratio better than* 1.25 *in the deadline-aware (and thus also deadline-oblivious) online* Timely Bin Packing *problem.*

Proof. We describe a reduction from Binary String Guessing to Timely Bin Packing as follows. Let S be an online sequence of m bits that forms an input to the Binary String Guessing problem. We create an instance σ_S of Timely Bin Packing, formed by m "phases", as follows. The i'th phase in σ_S takes two possible forms, depending on the i'th bit in S. Regardless, the i'th phase starts with sending items $a_i = (s = 0.5, r = 0, d = 3), b_i = (0.3, 0, 1), c_i = (0.26, 0, 5)$ and $d_i = (0.26, 0, 5)$. Now, if $S[i] = 0$, another item $e_i = (0.5, 2, 3)$ is sent, and no more items arrive in the i'th phase. On the other hand, if $S[i] = 1$, another item $f_i = (0.01, 4, 5)$ is sent, and no more items arrive in the i'th phase. The availability intervals of items of different phases do not have an intersection. See Figure 4.8 for an illustration of the instance.

Any reasonable algorithm packs the items of each phase separately from the items of the next or previous phase. So, we can discuss each phase independently from the other phases. Intuitively speaking, upon the deadline of the item b_i at each phase i, the algorithm must "guess" whether the phase is of type "0", where the item e_i appears, or of type "1", where the item f_i appears.

Upon the deadline of b_i, there are two possible cases of packing. Whether the algorithm packs it with a_i or packs it with c_i and d_i in the same bin.

If $S_i = 0$, the optimal packing is $B_i^1 = (b_i, c_i, d_i), B_i^2 = (a_i, e_i)$ (2 bins in total). On the other hand, an incorrect guess will result in the possible packing of $B_i^1 = (b_i, a_i), B_i^2 = (c_i, d_i), B_i^3 = (e_i)$ (pack B_i^1 and at least 2 bins are needed for the rest of the items). If $S_i = 1$, the optimal packing is $B_i^1 = (b_i, a_i), B_i^2 = (c_i, d_i, f_i)$ (2 bins in total). On the other hand, the wrong guess will result in the possible packing of $B_i^1 = (b_i, c_i, d_i), B_i^2 = (a_i), B_i^3 = (f_i)$ (pack B_i^1 and at least 2 bins are needed for the rest of the items). In other words, at each phase i, if the algorithm could guess the bit b_i correctly, it would solve that instance optimally with 2 bins; otherwise, it would have to use at least 3 bins for that phase.

Recall that with $o(n)$ bits of advice, one cannot correctly guess a fraction $1/2 + \epsilon$ of bits in the instance S of the Binary String Guessing problem, for any $\epsilon > 0$ [47]. This bound is achieved by always predicting the majority bit, which can be identified using a single bit of advice. The cost of the optimal solution that always guesses correctly is at most $2m$, and

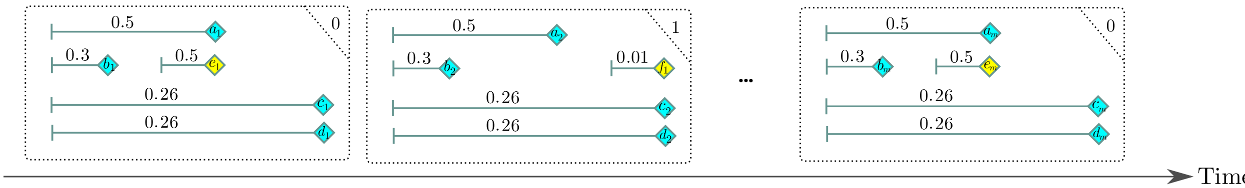


Figure 4.8: An instance σ_S of TIMELY BIN PACKING corresponding to a bit string $S = 01\dots0$. It contains m phases, each with two possible forms, corresponding to their bit in S.

the competitive ratio is at least

$$\frac{2m(\frac{1}{2}+\epsilon)+3m(\frac{1}{2}-\epsilon)}{2m} = 1.25 - \frac{\epsilon}{2}.$$

□

To further improve our results, we use the same technique as Lemma 4.3.3, but instead of *binary* STRING GUESSING, we use the general STRING GUESSING that its input is a sequence of n digits, where each digit can have q different values. Böckenhauer et al. [47] showed that for every constant q, the algorithm with sublinear bits of advice cannot guess more than $1/q+\epsilon$ ratio of the digits. We show a lower bound for the competitive ratio of TIMELY BIN PACKING for *any* algorithm with sublinear advice cannot be better than $(1.5-\epsilon)$-competitive, for every $\epsilon > 0$.

Theorem 4.3.4. *No algorithm with sublinear bits of advice can achieve a competitive ratio better than* $1.5-\epsilon$ *in the deadline-aware (and thus also deadline-oblivious) online* TIMELY BIN PACKING *problem for every* $\epsilon > 0$.

Proof. We describe a reduction from STRING GUESSING to TIMELY BIN PACKING as follows. Let S be an online sequence of n digits that forms an input to the STRING GUESSING problem. As before, we create an instance σ_S of TIMELY BIN PACKING, formed by n "phases", as follows. The i'th phase in σ_S takes k possible forms, depending on the i'th digit in S. Regardless, the i'th phase starts with sending items $c_i = (s = 1/k, r = 0, d = 0)$ and $k-1$ other items $a_i^j = (1/k + j\epsilon, 0, 2j)$ for $j \in [k-1]$ and a constant $\epsilon \ll 1/k$. Now, if $S[i] = j$, another item $b_i = (1-(1/k+j\epsilon), 2j-1, 2i)$ is sent, and no more items arrive in the i'th phase. The availability intervals of items of different phases do not have an intersection. See Figure 4.9 for an illustration of the instance.

At each phase i, the item b_i can be packed with only one item that has a size of $1/k+j\epsilon$: larger items cannot be packed with it as their sum would be larger than 1, and the availability intervals of smaller items does not intersect that of b_i. When $S[i]=j$, the optimal algorithm packs item c_i with all other items available at time 0, except b_i, and b_i with the item a_i^j

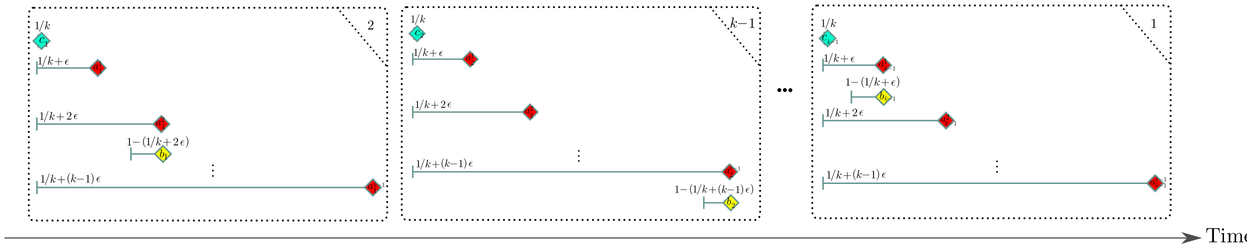


Figure 4.9: An instance σ_S of Timely Bin Packing corresponding to an input string $S = 2, k-1, \ldots, 1$. It contains m phases, each with $k-1$ possible forms, corresponding to their digit in S.

(which is of size $1/k + j\epsilon$). Thus, it uses 2 bins for this phase. Any algorithm, on the other hand, must leave at least one of the items at time 0 unpacked. If the algorithm "guesses" correctly, it uses 2 bins. However, if the algorithm guesses incorrectly, leaving any other $k-1$ available items (except b_i) unpacked, it uses 3 bins.

It is known that, with $o(n)$ bits, one cannot correctly guess a fraction $1/q + \epsilon$ of digits in the instance S of the String Guessing problem (where digits take q possible values), for any $\epsilon > 0$ [47]. Thus, for a constant k, with sublinear advice, the cost of every algorithm is at least $n(\frac{1}{k-1} \times 2 + \frac{k-2}{k-1} \times 3) = n(3 - \frac{1}{k-1})$. Given that the cost of the optimal solution is $2n$, the competitive ratio is at least

$$\frac{n(3 - \frac{1}{k-1})}{2n} = 1.5 - \frac{1}{2(k-1)}.$$

It can be arbitrarily close to 1.5 by increasing the value of k (yet, it must remain constant with respect to input length). □

4.4 Offline Timely Bin Packing

s In this section, we consider the offline setting of the problem via a simple reduction from the classic bin packing problem.

4.4.1 NP-hardness

We begin by proving the NP-hardness of the offline Timely Bin Packing problem.

Observation 4.4.1. *The offline Timely Bin Packing is NP-hard.*

Proof. We present a reduction from the classic bin packing problem to the Timely Bin Packing problem. Consider the decision problem that asks whether an instance

$S = \{s_1, \dots, s_n\}$ of the offline bin packing problem can be packed in k bins, where each item s_i has size in $(0, 1]$. Construct an instance $I = \{a_1, \dots, a_n\}$ of the offline TIMELY BIN PACKING problem by creating one trigger item a_i, setting its size equal to s_i, and its arrival and deadline as $r_i = d_i = 0$. Since all items are available from the start and share the same deadline, any two items can be packed together as long as they fit within a unit bin capacity. Therefore, any valid packing of I corresponds directly to a valid packing of S, and vice versa. Thus, the instance S of bin packing can be packed in k bins if and only if the instance I of TIMELY BIN PACKING can be packed in k bins. As the offline bin packing problem is NP-hard [60], so is the offline TIMELY BIN PACKING problem. □

We solve the problem optimally in a special case, where the item sizes are more than 1/3.

4.4.2 Polynomial-Time Algorithm for Restricted Size Items

In this section, we consider a setting of the problem in which all items have sizes greater than 1/3. We demonstrate that the TIMELY BIN PACKING problem can be optimally solved in polynomial time by reducing it to the maximum matching problem.

Theorem 4.4.1. *There exists a polynomial-time optimal algorithm for the offline TIMELY BIN PACKING problem when all item sizes are greater than* 1/3.

Proof. We reduce the TIMELY BIN PACKING problem to a graph matching problem, which asks for the maximum number of edges in a graph such that no two selected edges share a common endpoint. It is well-known that the matching problem can be solved in polynomial time via, for example, the blossom algorithm of Edmonds [112].

Given an instance I of the TIMELY BIN PACKING problem, we construct a graph G where each item in I corresponds to a node in G. We create an edge between two nodes if their corresponding items in I have overlapping available times. Since each item has a size greater than 1/3, at most two items can fit into a single bin (which will correspond to a match edge).

We then apply a maximum matching algorithm (such as the blossom algorithm) to the graph G, which runs in polynomial time. Each edge in the matching $a - b$ indicates that items a and b have to be packed together in the same bin in the instance I. Any unmatched vertices represent items that must be packed individually in separate bins. □

4.5 Case Studies for Online Timely Bin Packing

In this section, we study the Timely Bin Packing problem for restricted settings such as monotone availability intervals, uniform size, etc. An overview of the results of this section is presented in Table 4.2.

4.5.1 Instances with Items of Uniform Size

Consider the Timely Bin Packing problem when all items have the same size. For this variant of the problem, we establish a strong separation between the deadline-aware and deadline-oblivious settings. In particular, we first introduce an optimal algorithm for the deadline-aware setting, and then establish a lower bound of 2 for the competitive ratio of any deterministic algorithm.

Greedy-on-Deadline Algorithm (GoD). GoD is a simple, deterministic, deadline-aware algorithm that works as follows. Upon the deadline of an item a_i, sort all available items in the increasing order of their deadlines (that is, a_i will be the first item in such order) and pack a_i with as many other items as possible. The proof for optimality is direct from the fact that all items are identical, except that they may have different deadlines. Naturally, replacing an item with a further deadline with one that is more urgent in the bin that is being delivered does not increase the cost of the algorithm.

Observation 4.5.1. *The Greedy-on-Deadline algorithm is optimal for the deadline-aware online Timely Bin Packing problem with uniform-size items.*

The next lemma shows that, unlike the deadline-aware setting, in the deadline-oblivious setting, no deterministic algorithm can be better than 2-competitive. The proof is based on similar ideas explored in the proof of Theorem 4.2.3. Namely, in the inputs with items of similar sizes but different deadlines, a deadline-oblivious algorithm cannot distinguish "urgent" items from the ones with later deadlines.

Theorem 4.5.1. *No deterministic algorithm can achieve a competitive ratio better than* 2 *for the deadline-oblivious Timely Bin Packing with uniform size items.*

Proof. Consider an instance with all the items having a uniform size of $1/k$ with k being a large integer. Thus, a bin can hold exactly k items to be completely filled. There are three types of items as follows:

- A: there are t items, where t is a multiple of k, $a_i = (s_i = 1/k, r_i = 0, d_i = i\epsilon)$.

Table 4.2: Results of the case study under different settings of the TIMELY BIN PACKING problem.

		Deadline-Aware		Deadline-Oblivious	
		Deterministic	Randomized	Deterministic	Randomized
Uniform Size Items	Upper Bound	1 (Observation 4.5.1) Greedy-on-Deadline	1 (Observation 4.5.1) Greedy-on-Deadline	3 (Theorem 4.2.6) SPAD	3 (Theorem 4.2.6) SPAD
	Lower Bound	-	-	2 (Theorem 4.5.1)	-
Item Sizes $\geq 1/4$	Upper Bound	2 (Theorem 4.2.10) Discrete-Greedy	2 (Theorem 4.2.10) Discrete-Greedy	2 (Theorem 4.5.2) Group-and-Pack	2 (Theorem 4.5.2) Group-and-Pack
	Lower Bound	1.5 (Corollary 4.5.3)	-	2 (Theorem 4.2.3)	-
Item Sizes $< 1/k$	Upper Bound	2 (Theorem 4.2.10) Discrete-Greedy	2 (Theorem 4.2.10) Discrete-Greedy	3 (Theorem 4.2.6) SPAD	3 (Theorem 4.2.6) SPAD
	Lower Bound	-	-	2 (Theorem 4.5.4)	-
Monotone Availability Intervals	Upper Bound	2 (Theorem 4.5.5) Pack-at-Deadline	2 (Theorem 4.5.5) Pack-at-Deadline	NA	
	Lower Bound	-	-		
Unit Length Availability Intervals	Upper Bound	2 (Theorem 4.2.10) Discrete-Greedy	2 (Theorem 4.2.10) Discrete-Greedy		
	Lower Bound	1.5 (Theorem 4.5.6)	1.5 (Theorem 4.5.7)		
Unit Length Availability Intervals, Integer Arrival	Upper Bound	2 (Theorem 4.2.10) Discrete-Greedy	1.5 (Theorem 4.5.8) Random-Integer-Endpoint		
	Lower Bound	4/3 (Theorem 4.5.9)	1.2 (Theorem 4.5.10)		
Unit Length Availability Intervals, Sizes in $\{\alpha, 1-\alpha\}$	Upper Bound	2 (Theorem 4.2.10) Discrete-Greedy	1.5 (Theorem 4.5.11) Random-Endpoint		
	Lower Bound	1.5 (Theorem 4.5.12)	1.25 (theorem 4.5.13)		

- P: there are t trigger items, $p_i = (1/k, 1+i, 1+i)$.
- B: there are $t(k-1)$ items, $b_i = (1/k, 0, 2t)$.

The optimal algorithm packs all the A items together, using t/k bins, as each bin can hold k items and there are t items in total. For the trigger items, it combines each trigger with $k-1$ items from B, using t bins. Therefore, the cost of the optimal solution is $t/k + t$. As the algorithm is unaware of the item deadlines, we can assume that it packs each A item with $(k-1)B$ items, resulting in the use of t bins. Then, when the deadline for the trigger items arrives, no other items are available and they must be packed separately, using an additional t bins. Thus, the lower bound for the competitive ratio is $\frac{2t}{(t+tk)/k} = \frac{2}{1+1/k}$. As k tends to infinity, the lower bound converges to 2. □

Recall that SPAD has a competitive ratio of at most 3 in the general setting, and thus in the uniform-size setting. Finding an algorithm with a better competitive ratio remains a topic for future work.

4.5.2 Instances with Item Sizes Coming from Restricted Ranges

In this section, we discuss the Timely Bin Packing problem in two different settings, where items are either all large or all small. First, we consider instances where all items are of size at least $1/4$, and then we will discuss a setting in which all items are smaller than $1/k$ for some large integer k.

For the setting where all items are of size $1/4$ or more, we propose the following deadline-oblivious algorithm.

Group-and-Pack Algorithm. The algorithm packs bins only at item deadlines. Consider two groups of items: group x contains items of size larger than half, and group y contains other items. Upon the deadline of an item of group x, pack it into a single bin and deliver that bin. Upon the deadline of an item a_i of group y, if other items of group y are available, pack a_i with one or two such items. Otherwise, pack a_i in a separate bin and deliver B.

In what follows, we prove that Group-and-Pack has a competitive ratio of at most 2, which is better than the upper bound of 3 given by SPAD in the general setting.

Theorem 4.5.2. *The Group-and-Pack algorithm achieves a competitive ratio of at most* 2 *for the deadline-oblivious online* Timely Bin Packing *when item sizes are of size at least* $1/4$.

Proof. The optimal solution requires at least one bin for each item in group x, since no two such items can fit in the same bin. It can then use the remaining space in those bins to pack some of the y items. For the remaining y items, at most three can fit in a bin, as all items are larger than $1/4$ in size. Thus, the optimal solution needs at least $\frac{y-x}{3}$ additional bins for the remaining y items.

The Group-and-Pack algorithm assigns one bin to each x item and packs the y items separately, grouping two or three items per bin. If two y items are placed separately in a bin by the algorithm, they must not share an availability interval, since otherwise, the algorithm would pack them together. Therefore, the optimal solution must also use at least one bin for each such y item packed individually. Therefore, we have

$$\begin{aligned}
\textsc{Alg} &\leq x + \frac{y}{2} + \frac{\textsc{Opt}}{2} \\
\frac{2}{3}\textsc{Alg} &\leq \frac{2}{3}x + \frac{y}{2} + \frac{\textsc{Opt}}{3} = x + \frac{y-x}{3} + \frac{\textsc{Opt}}{3} \\
\textsc{Alg} &\leq 2\textsc{Opt}
\end{aligned}$$

Therefore, in this case, the upper bound for the competitive ratio is 2. □

Corollary 4.5.3. *No deterministic deadline-aware algorithm can achieve a competitive ratio better than* 1.5 *in the online* Timely Bin Packing *problem, even when item sizes are no less than* 1/4.

Proof. The proof is similar to the proof of Theorem 4.2.1, except the size of the item F, which is 0.25 rather than 0.01. As before, at timestep 0, the adversary releases 4 items $A = (s = 0.5, r = 0, d = 3), B = (0.3, 0, 1), C = (0.26, 0, 5)$ and $D = (0.26, 0, 5)$.

The algorithm must pack the item B before its deadline, timestep 1. There are two cases to consider, depending on how an algorithm packs B with other times.

(i). If B gets packed with A, there would be no more space for the other items in the bins that they share. The adversary then sends another item $E = (0.5, 2, 3)$. The algorithm cannot pack the new item with both C and D in one bin. Thus, it needs at least 2 more bins to pack the items (3 bins in total). In contrast, the optimal solution packs A and E together in one bin, and B, C, and D in another. Thus, the competitive ratio would be 3/2 in this case.

(ii). Otherwise, if A does not get packed in the same bin as B, the other two items C and D can be in its bin. The adversary adds a new item $F = (0.25, 4, 5)$. As this new item has no intersection with A, the algorithm again needs 2 more bins (3 in total). On the other hand, the optimal solution could pack A and B together in one bin and F, C, and D in another. Therefore, the competitive ratio in this case is 3/2 as well. □

Next, we will investigate a setting where all items are small. It is well known that in the classic bin packing, these instances constitute easy inputs to the problem; intuitively, regardless of the packing, all bins can be filled. In contrast, in what follows, we show that a deadline-oblivious algorithm cannot achieve a competitive ratio better than 2 even for these inputs in TIMELY BIN PACKING.

Theorem 4.5.4. *No deadline-oblivious deterministic algorithm can achieve a competitive ratio better than* 2 *when the item sizes are less than* $1/k$*.*

Proof. Assume $k = 2k'$, in the instance follows, all the item sizes are less than $1/k'$. Consider an instance with three sets of items

- Trigger items: One item $a_0 = (s_0 = 1/k, r_0 = 0, d_0 = 0)$ and for each $i \in [m-1]$ let $a_i = (1/k - ((i+1)(k-2)+1)\epsilon, 2i, 2i)$.
- For each $i \in \{0, \ldots, m-1\}$ there is a set of items b_i which includes $k-2$ items $b_i^j = (1/k + \epsilon, 2i, 2i+1)$ for $j \in \{k-2\}$.
- For each $i \in \{0, \ldots, m-1\}$, let $c_i = (1/k + \epsilon, 2i, 2i+2)$

The optimal solution packs each trigger time a_i with c_{i-1} (if it exists) along with all items of the set b_i. In other words, at each time $2i$ (the deadline of trigger item a_i), it leaves the item c_i unpacked. Thus, it uses a total of m bins.

Any deadline-oblivious deterministic algorithm cannot differentiate between the items of b_i and c_i. Thus, at the deadline of the trigger item a_i, it may leave one of the items of b_i unpacked. In other words, at each time $2i$ for $i \in \{0, \ldots, m-1\}$, it packs a_i, c_i and $k-3$ items from b_i and leaves one b_i^{j*} unpacked. It then has to pack b_i^{j*} in a separate bin at its deadline, as no other items are available. Thus, it uses at least $2m$ bins in total. Therefore, the competitive ratio of any deterministic algorithm is at least 2. □

4.5.3 Instance with Monotone Availability Intervals

Consider the *monotone* instances of the TIMELY BIN PACKING problem in which sorting items by their arrival times results in the same ordering as sorting them based on their deadlines. In other words, in monotone instances, if an item a_i arrives before item a_j, then the deadline of a_i must also be before a_j. In the following theorem, we prove that Pack-at-Deadline has a competitive ratio of at most 2 in these restricted input instances.

Theorem 4.5.5. *The Pack-at-Deadline algorithm achieves a competitive ratio of at most* 2 *in the online* TIMELY BIN PACKING *problem with monotone availability intervals.*

Proof. We know that every time the Pack-at-Deadline algorithm delivers a bin, there is an item, call it *starting item*, that has reached its deadline. Let the multiset of arrival and deadline times of the starting items be $\{(r_1, d_1), (r_2, d_2), \ldots, (r_k, d_k)\}$. We know that these availability intervals are disjoint; otherwise, if two intervals of items a_i and a_j share a common point in time, the item with the later deadline (say a_j) is available at the time when the other item (a_i) reaches its deadline. Since Pack-at-Deadline packs all available items by the deadline of a_i, it must have packed a_j at that time as well, contradicting the fact that a_j is a starting item. Using this observation, we can partition the entire timeline into $k+1$ parts as follows: $(-\infty, d_1], (d_1, d_2], (d_2, d_3], \cdots, (d_k, \infty)$. As it is a partition, every bin in the optimal solution must be in one of these *parts*.

Next, consider every bin in the optimal solution like B. Divide B into two bins, one containing all items whose availability interval intersects the previous part (if such a part exists). The other bin contains the remaining items in B. The items of the second bin are all available at the end time of the current part. This is because their starting time is within this part, and if they end before the end time of the part, Pack-at-Deadline would have delivered some bins earlier, which would lead to a contradiction.

In summary, we found a solution that delivers all available items at deadlines and uses at most twice bins as the optimal solution. Because Pack-at-Deadline is the best among all such algorithms (as it packs optimally at deadlines), its competitive ratio is at most 2. □

4.5.4 Instances with Availability Intervals of Unit Length

In this section, we consider a special setting where each item's availability interval is of the form $[x, x+1]$ for some $x \in \mathbb{R}$. Although this is an even more restricted setting than monotone instances, we can still establish lower bounds on the competitive ratio, suggesting that the online constraint remains quite restrictive even for these special instances.

First, we establish an impossibility result for the deterministic algorithms via a simple adversarial argument as follows.

Theorem 4.5.6. *No deterministic algorithm can achieve a competitive ratio better than* 1.5 *for the online* Timely Bin Packing *problem with unit length availability intervals (in both deadline-aware and deadline-oblivious settings).*

Proof. To prove the lower bound of 1.5 for the competitive ratio, consider an instance with items $A = (s_i = 1/3, 0, 1)$, $B = (1/2, 0.5, 1.5)$, and $C = (1/2 + \epsilon, 1, 2)$. The algorithm must pack the item A before its deadline, and there are two cases for doing so.

(i). Pack A with B: The adversary then sends an item $D = (1/2, 1.5, 2.5)$. The optimal solution packs A with C in one bin and B and D in another bin; that is OPT uses 2 bins. The algorithm, on the other hand, packs A and B into a single bin but then needs separate bins for each of C and D as they cannot fit into a unit bin. Therefore, the cost of the algorithm would be 3.

(ii). Pack A with C: The adversary then sends an item $E = (\epsilon, 2, 3)$. The optimal solution packs the A with B in one bin and C with E in another bin; therefore, OPT uses 2 bins. The algorithm, on the other hand, packs A with C, and because the availability intervals of B and E do not intersect, it must use two extra bins to pack them separately. Therefore, the cost of the algorithm is at least 3

To conclude, in both cases, the algorithm uses 1.5 times as many bins as the optimal algorithm. This establishes a lower bound of 1.5 for the competitive ratio. □

Next, we extend the above negative result to randomized algorithms.

Theorem 4.5.7. *No randomized algorithm can achieve a competitive ratio better than* 1.5 *in online* TIMELY BIN PACKING *with unit length availability intervals.*

Proof. The proof idea is similar to that of the proof of Theorem 4.2.2. We will use Yao's min-max principle (see Definition 1.4.4). For that, we consider a deterministic algorithm faced with a probability distribution over input sequences. All sequences start with one item $a_1 = (s_1 = 1/k, r_1 = 0, d_1 = 1)$, for a value of k that will be defined k later. This is followed by $k-1$ other items $b_i = (1/k + i\epsilon, i\epsilon, 1 + i\epsilon)$ for $i \in [k-1]$. The input then continues with one of the items $c_i = (1 - (1/k + i\epsilon), 1 + i\epsilon, 2 + i\epsilon)$ for some $i \in [k-1]$ selected uniformly at random. This extra item can be packed with only one item that has a size of $1/k + i\epsilon$. Larger items cannot be packed with it as the summation would be larger than one, and smaller items do not have any common time with its availability interval.

The optimal algorithm, in this scenario, packs the first item alongside all other items available at timeslot 1 except item b_i, which will be packed with the last item that will arrive at time $1 + i\epsilon$. Thus, it uses 2 bins. The algorithm, on the other hand, must leave at least one of the items at timeslot 1 unpacked. With probability $(k-2)/(k-1)$, this item is the wrong item, and the algorithm must use at least 3 bins. Therefore, the cost of the algorithm is at least $3.(k-2)/(k-1) + 2/(k-1)$. If we tend k to infinity, the cost would be 3, and the competitive ratio would be at least $3/2 = 1.5$. □

Instances with Integer Arrival Times

Consider the instances where arrival times are integers. Given that we are concerned with items of unit-length availability intervals, the deadlines are integer time as well; that is, the availability interval are in the form $[x, x+1]$ where $x \in \mathbb{Z}$. In what follows, we first present a randomized algorithm with a competitive ratio of at most 1.5 for this setting of the problem (Theorem 4.5.8) and then prove lower bounds of 1.5 (Theorem 4.5.9) and 4/3 (Theorem 4.5.10) for the competitive ratio of deterministic and randomized algorithms, respectively.

Random-Integer-Endpoint Algorithm. Consider two online strategies, called E and O. The E strategy forms an optimal packing of all pending items at even time steps (it does not pack any item in odd time steps). Similarly, strategy O optimally packs items on odd time steps (and does not pack any item in even time steps). Given that every item finds one endpoint of its availability interval in an even integer and the other in an odd integer, both E and O strategies are well-defined. Our randomized algorithm, Random-Integer-Endpoint, is a barely random algorithm that flips a fair coin at the beginning to select either O or E with equal chance and follows the selected algorithm (it uses 1 random bit).

We proceed to prove an upper bound for the competitive ratio of Random-Integer-Endpoint.

Theorem 4.5.8. *The Random-Integer-Endpoint algorithm achieves a competitive ratio of at most* 1.5 *for the online* Timely Bin Packing *for instances with unit length availability intervals and integer arrival times.*

Proof. We know that each item finds one endpoint of its availability interval on an odd time and the other on an even time. Consider an instance I consisting of items $a_i = (s_i, r_i, r_i + 1)$ for $i \in [n]$. Construct a new instance I_E by creating, for each $a_i \in I$, a corresponding trigger item $a'_i = (s_i, t, t)$, where $t = r_i$ iff r_i is even and $t = r_i + 1$ otherwise. Similarly, construct a new instance I_O the same way by creating, for each $a_i \in I$, a corresponding trigger item $a''_i = (s_i, t, t)$, where $t = r_i$ iff r_i is odd and $t = r_i + 1$ otherwise. Finding the optimal solution for both I_O and I_E is straightforward, as each item must be packed exactly at its release time. We claim that $\text{Opt}(I_E) + \text{Opt}(I_O) \leq 3\text{Opt}(I)$. Given that the expected cost of Random-Integer-Endpoint is $(\text{Opt}(I_O) + \text{Opt}(I_E))/2$, we can conclude that the expected cost of Random-Integer-Endpoint is at most $3\text{Opt}(I)/2$, which completes the proof.

It remains to prove the claim. That is, to show that, for every instance I, $\text{Opt}(I_E) + \text{Opt}(I_O) \leq 3\text{Opt}(I)$. We process the m bins in an optimal solution of I, time by time, and retrieve two solutions for I_E and I_O such that the total number of bins in these two solutions is $3m$. Let B be any bin in the optimal solution of I and consider a time step t. W.l.o.g., suppose t is even. In the solution for I_E, we include the same bin B, while in the solution for I_O, we include at most two bins delivered at time $t-1$ and $t+1$. Repeating this for all time steps, we can pack both I_E and I_O within at most $3m$ bins. $\square$

Next, we establish our impossibility results.

Theorem 4.5.9. *No deterministic algorithm can achieve a competitive ratio better than* 4/3 *for the online* TIMELY BIN PACKING *problem with unit length availability intervals and integer arrival times.*

Proof. Consider an instance starting with four items $A = (1/3, 0, 1)$, $B = (1/2, 1, 2)$, $C = (1/4 + \epsilon, 1, 2)$, and $D = (1/4 + \epsilon, 1, 2)$. The algorithm must pack item A before its deadline at timestep 1. There are two possibilities for packing A, depending on the other items packed with it in its bin.

(i). Pack A with B: There would be no more space for the other items. The adversary then adds another item $E = (1/2, 2, 3)$. The algorithm cannot pack the new item with C and D in one bin. Thus, it needs at least 2 more bins (3 bins in total) to pack the items. The optimal solution, on the other hand, could pack items B and E together, as well as items A, C, and D together. Thus, the ratio between the cost of the algorithm and OPT is 3/2 in this case.

(ii). Pack A with C and D: The adversary adds two new items $F = G = (3/4 - \epsilon, 3, 4)$. As these new items cannot fit into the same bin as the item B, the algorithm needs 3 more bins (4 in total). However, the optimal solution needs one bin to pack A and B together at timestep 1, and two other bins to pack C with F and D with G at timestep 2 (3 bins in total). Therefore, the ratio between the cost of the algorithm and OPT in this case is 4/3.

The above idea gives a competitive ratio of at least 4/3 for the problem. $\square$

Next, we provide a slightly weaker lower bound for the competitive ratio of randomized algorithms, using a similar proof that uses a stochastic input.

Theorem 4.5.10. *No randomized algorithm can achieve a competitive ratio better than* 1.2 *for the online* TIMELY BIN PACKING *problem with unit length availability intervals and integer arrival times.*

Proof. Using Yao's principle (see Definition 1.4.4), we consider a deterministic algorithm ALG faced with a stochastic input defined as follows. Consider instances starting with of four items $A = (1/3, 0, 1)$, $B = (1/2, 1, 2)$, $C = (1/4 + \epsilon, 1, 2)$, and $D = (1/4 + \epsilon, 1, 2)$. The algorithm must pack item A before its deadline at timestep 1. The input continues randomly as follows. At timestep 2, the adversary releases two items $E = F = (3/4 - \epsilon, 2, 3)$ with probability p, or alternatively, releases one item $G = (1/2, 2, 3)$ with probability $1 - p$ for $p = 0.6$. There are two possible cases for the deterministic algorithm.

(i). Case 1: Suppose ALG packs A and B in separate bins. In this case, the algorithm uses at least 1 bin to pack A, C, and D. Given that with a probability of p, two new items E and F are added at timestep 2, and these new items cannot be packed with each other or with B, the algorithm needs 3 more bins (or 4 in total) with a probability of p. In this case, the optimal solution packs A with B and C with E and D with F, opening 3 bins. With a probability of $1 - p$, the algorithm uses 2 bins as the optimal does. The worst-case ratio in this case is $(p)(\frac{4}{3}) + (1-p)(\frac{2}{2}) = 1 + \frac{p}{3} = 1.2$.

(ii). Case 2: Suppose ALG packs A with B in the same bin. In this case, there would be no more space for the other items in the same bin. Given that with a probability of $1-p$, a new item G is added at timestep 2. The algorithm cannot pack the new item with C and D in one bin. Thus, it needs at least 2 more bins (3 bins in total) to pack the items in this case. The optimal solution, on the other hand, could pack B and G in one bin, and A, C, and D in a second bin. With a probability of p, the algorithm uses 3 bins as the optimal does. The worst-case ratio in this case is $(p)(\frac{3}{3}) + (1-p)(\frac{3}{2}) = \frac{3}{2} - \frac{p}{2} = 1.2$.

Altogether, regardless of the choices of the algorithm, the ratio between the expected cost of the algorithm and that of OPT is at least 1.2. □

Instances with Restricted Sizes

In this section, we explore instances of TIMELY BIN PACKING with unit availability interval where item sizes are restricted to the set $\{\alpha, 1 - \alpha\}$.

Random-Endpoint Algorithm. Choose either Strategy 1 or Strategy 2 uniformly at random. In Strategy 1, all items of size $1 - \alpha$ are restricted to be packed only at their arrival time (similar to trigger items). In Strategy 2, all items of size $1 - \alpha$ must be packed only at their deadline. In both strategies, all items (both α and $1 - \alpha$, subject to their respective time restrictions) are packed greedily: when an item reaches its deadline, sort the available items in increasing order of their deadlines and insert them one by one into the current bin until it no longer has capacity.

Theorem 4.5.11. *The Random-Endpoint algorithm achieves a competitive ratio of at most* 1.5 *for the online* TIMELY BIN PACKING *problem with unit length availability intervals and sizes in* $\{\alpha, 1-\alpha\}$.

Proof. We refer to an item as a *small* (respectively *large*) if it is of size α (respectively $1-\alpha$). As all items have unit availability interval, there exists one optimal solution in which any of the large items is packed either at its starting time or on its deadline.

Consider an input instance $I = \langle a_1, \ldots, a_n \rangle$, where $a_i = (s_i, r_i, d_i)$. We create two new instances $I' = \langle a'_1, \ldots, a'_n \rangle$ and $I'' = \langle a''_i, \ldots, a''_n \rangle$ as follows. In I', we set $a'_i = (1-\alpha, r_i, r_i)$ if $s_i = 1-\alpha$ and $a'_i = (\alpha, r_i, d_i)$ otherwise. Similarly, in I'', we set $a''_i = (1-\alpha, d_i, d_i)$ if $s_i = 1-\alpha$ and $a''_i = (\alpha, r_i, d_i)$ otherwise.

The optimal solution for each of the instances I' and I'' would be greedy based on the deadlines. The optimal solution packs any large item with a small item with the earliest deadline, as the large items are trigger items. Moreover, it packs small items when they reach their deadlines in a greedy manner and in the increasing order of their deadlines. The Random-Endpoint algorithm implicitly picks one of the instances randomly and delivers a packing of I' and I'' based on the random choice. Thus, the expected cost of the algorithm is $(\text{OPT}(I') + \text{OPT}(I''))/2$. Next, we show that for every instance I, $\text{OPT}(I') + \text{OPT}(I'') \leq 3\text{OPT}(I)$. Fix an optimal solution, and let OPT_α denote the multiset of bins in that solution that only contain small items, and let $\text{OPT}_{1-\alpha}$ denote the multiset of remaining bins (those containing one large and possibly one small item). We form a packing of I' and a packing of I'' that, together, use $3\text{OPT}(I)$ bins. First, bins in OPT_α are copied in both solutions for I' and I''. Moreover, any bin in $\text{OPT}_{1-\alpha}$ is projected to at most three bins across the two packings for I' and I''.

The large item and the small item paired in any bin B of $\text{OPT}_{1-\alpha}$ have either an intersection on the arrival time of the large item, or on its deadline, or in both. If they intersect in both, we again copy B in both packings for I' and I''. Next, assume the pair of items in B intersect on one endpoint of the large item, say the deadline. In the solution for I'', we copy B (it is still valid for that instance), while in the solution for I', we create two bins, one containing the large item and a separate one containing the small item. This is because the small and large items do not intersect in I'. The case when the pair intersects at the starting time of the large item is handled symmetrically. In summary, we have $\text{OPT}(I') + \text{OPT}(I'') \leq 2\text{OPT}_\alpha + 3\text{OPT}_{1-\alpha} \leq 3\text{OPT}$ which completes the proof. ☐ Next, we establish our impossibility results for this setting.

Theorem 4.5.12. *No deterministic algorithm can achieve a competitive ratio better than* 1.5 *for the online* TIMELY BIN PACKING *with unit length availability intervals and item sizes* *in* $\{\alpha, 1-\alpha\}$.

Proof. Consider an instance where the adversary releases 3 items $A = (\alpha, 0, 1)$, $B = (\alpha, 0.5, 1.5)$, and $C = (1-\alpha, 1, 2)$. The algorithm must pack item A before its deadline, timestep 1. There are two cases in which other items will also be packed with A:

(i). Pack A with B: The adversary then sends another item $D = (1-\alpha, 1.5, 2.5)$. The algorithm cannot fit the new item with C in one bin; thus, it needs at least two more bins (3 bins in total, $(A, B), (C), (D)$) to pack the items. The optimal solution, on the other hand, could pack A with C and B with D. Thus, the competitive ratio would be $3/2$ in this case.

(ii). Pack A and B separately: The adversary adds a new item $E = (\alpha, 2, 3)$. As E does not have any intersection with B, the algorithm needs 2 more bins to pack these items (at least 3 bins for A, B, and D, in total). However, the optimal solution needs one bin to pack A and B together, and one other bin to pack C with E at timestep 2 (2 bins in total). Therefore, the competitive ratio in this case is $3/2$.

The above idea gives a competitive ratio of 1.5 for the problem. $\square$

Theorem 4.5.13. *No randomized algorithm can achieve a competitive ratio better than* 1.25 *for the online* TIMELY BIN PACKING *with unit length availability intervals and item sizes in* $\{\alpha, 1-\alpha\}$.

Proof. Using Yao's principle (see Definition 1.4.4), we consider a deterministic algorithm ALG faced with a stochastic input defined as follows. The input states with 3 items $A = (\alpha, 0, 1)$, $B = (\alpha, 0.5, 1.5)$, and $C = (1-\alpha, 1, 2)$. The algorithm must pack item A before its deadline, timestep 1. The adversary releases an item $D = (1-\alpha, 1+\epsilon, 2+\epsilon)$ with probability p, or $E = (\alpha, 2, 3)$ with probability $1-p$, for $p = 0.5$, as explained below. There are two cases in which other items will also be packed with A.

(i). Case 1: Suppose ALG packs item A with B: There would be no more space for the other items in the bin of (A, B). Since with a probability of p, item $D = (1-\alpha, 1+\epsilon, 2+\epsilon)$ is added, and C and D cannot be packed in one bin, ALG needs at least 2 more bins (3 bins in total) to pack the items. The optimal solution, on the other hand, could pack items A with C and B with D. With a probability of $1-p$, the algorithm uses 2 bins as the optimal does. The worst-case ratio in this case is $(p)(\frac{2}{2}) + (1-p)(\frac{3}{2}) = \frac{3}{2} - \frac{p}{2}$.

(ii). Case 2: Suppose ALG packs item A item with item C: Since with a probability of $1-p$, a new item $E = (\alpha, 2, 3)$ is added, and E does not have any time intersection with item B, the algorithm needs 2 more bins (3 in total) to pack all items. However, the optimal solution combines A and B together and C with E, resulting in the use of two bins. With a probability of p, the algorithm uses 2 bins as the optimal does. The worst case ratio in case 2 is $(p)(\frac{3}{2}) + (1-p)(\frac{2}{2}) = 1 + \frac{p}{2}$.

Altogether, regardless of the choices of the algorithm, the ratio between the expected cost of the algorithm and that of OPT is at least 1.25. $\square$

Chapter 5

Results for the Dynamic TCP Acknowledgement Problem

5.1 Overview & Summary of Results

In this chapter, we present our results on the Dynamic TCP Acknowledgement problem under the $f_{\max}$ objective, focusing on settings where the algorithm has access to predictions or advice. Our main contributions are summarized below:

- In Section 5.2 we explore the Dynamic TCP Acknowledgement problem under the $f_{\max}$ objective when the algorithm receives predicted arrival times of the next packet. We introduce a parameterized algorithm that utilizes predictions to enhance performance while remaining robust in the presence of errors.
  - In Section 5.2.1 we define our prediction model and its relevance for learning-augmented algorithms. The prediction specifies the arrival time of the next packet $\hat{a}_{j+1}$ after seeing each packet a_j. In particular, at each given time t, the online algorithm is given a prediction that specifies when the next packet is expected to arrive.
  - In Section 5.2.2 we describe our error model as the ℓ_∞ distance between the vector of predicted arrival times and actual arrival times. That is, we define the prediction error η as the maximum difference between predicted and actual arrival times: $\eta = \max_j |\hat{a}_j - a_j|$. We also explain why the ℓ_∞ error is a natural choice, drawing motivation from related areas.
  - In Section 5.2.3 we investigate the Trust algorithm, which performs optimally when predictions are perfect (Lemma 5.2.1). We show that while this algorithm is

1-competitive under perfect predictions, its performance can degrade significantly if the predictions are inaccurate (Lemma 5.2.3). This motivates the development of algorithms that are both robust and consistent.

  - In Section 5.2.4 we present a family of algorithms with a trust parameter $\lambda \in (0, 1)$ ($\text{Augmented}_{\text{Alarm}}(\lambda)$), which controls how much the algorithm trusts the prediction. Specifically, when λ converges to 0, the algorithm almost fully trusts the prediction, and when $\lambda = 1$, the algorithm ignores the prediction and acts like a purely online algorithm with no prediction. We prove upper and lower bounds for the competitive ratio $\text{CR}(\lambda, \eta)$ of any member of this family as functions of η (Theorems 5.2.4 and 5.2.5). In particular, when $\eta = 0$, we get a consistency of $1+\lambda$ and when η converges to infinity, we get the robustness $1+\frac{1}{\lambda}$ (Corollary 5.2.6).

- In Section 5.3 we evaluate the practical performance of the algorithms for the Dynamic TCP Acknowledgement problem under the $f_{\max}$ objective.

  - In Section 5.3.1, we present the real-world MAWI dataset [113] used in our experiments, along with the distribution used to generate synthetic packet arrivals. We also explain how the arrival sequences are created and how predictions with bounded error η are added.
  - In section 5.3.2, we discuss implementation details, including the number of trials, the range of prediction errors, and the value of ρ (acknowledgement cost).
  - In Section 5.2.4, we present the algorithms used in the experiments and describe their characteristics.
  - In Section 5.3.4, we describe the evaluation metrics and explain how the performance of the algorithms is measured.
  - In Section 5.3.5, we analyze the experimental results and highlight the performance trends observed across different settings.

- In Section 5.4 we study the advice complexity of the problem and show that any algorithm with sublinear advice has a competitive ratio of at least 1.1716 (Theorem 5.4.1). In particular, any algorithm that uses a prediction of sublinear size cannot be better than 1.1716-competitive, even if all predictions are perfect. We note that any reasonable prediction that is PAC learnable is likely to have a size that is sublinear to the input length.

5.2 Algorithm with Prediction

In this section, we first describe our prediction model, error model, and application of a Trust algorithm that blindly follows the prediction and thus is not robust. Subsequently, we present a new algorithm designed to allow a trade-off between consistency and robustness.

5.2.1 Prediction Model

We consider a setting where, upon observing each packet arrival a_j, the algorithm receives a predicted arrival time $\hat{a}_{j+1}$ for the next packet. In other words, upon arrival of a_j, the algorithm makes a query about the next (unknown) arrival time a_{j+1} and for that it receives $\hat{a}_{j+1}$. We further assume that a prediction is made at time 0 about the arrival time of the first item.

The predictions are provided by an external oracle and are potentially learned from past data. The algorithm may use these predictions to make informed decisions about the next time to make acknowledgements. This form of next-event prediction is also common in other online settings, such as paging or caching, where the prediction estimates when a requested item will appear again [49, 50, 51, 52, 53, 54, 55, 56]. Such predictions are natural and can be interpreted as a form of lookahead.

5.2.2 Error Model

We evaluate prediction quality using an error parameter η, defined as the maximum absolute difference between predicted and actual arrival times:

$$\eta = \max_{j \in [n]} |\hat{a}_j - a_j|.$$

This corresponds to the ℓ_∞ distance between the prediction vector and actual packet arrival vector, and captures the worst-case deviation across all predictions. The ℓ_∞ norm is widely used in control theory for designing robust controllers in systems such as aircraft, robotics, and power grids, where limiting maximum error is crucial for safety and performance [114]. It also plays a role in image processing tasks, including quality assessment, bounding color deviations, and compression methods where controlling worst-case artifacts is essential [115].

It should be pointed out that, in previous work studying the TCP acknowledgement problem under the f_{sum} objective, the error is defined as the ℓ_1 distance between the prediction and actual arrival-time vectors. This is because, unlike ours, in that problem, the objective is cumulative over delays of all packets, and thus all delays contribute to the objective.

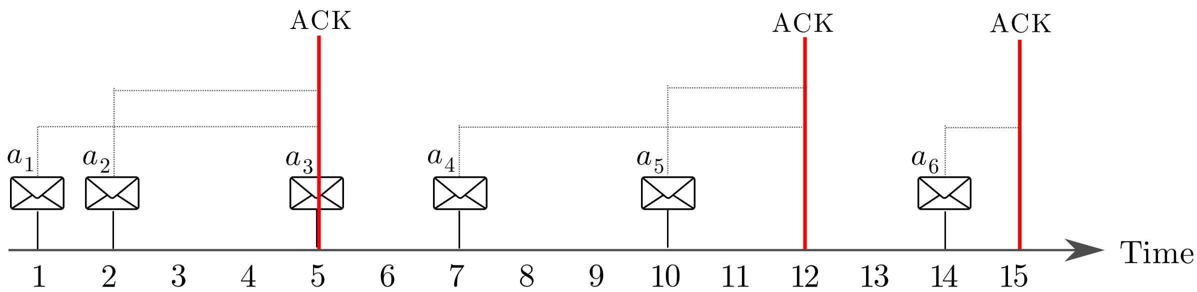


Figure 5.1: An example illustrating Dynamic TCP Acknowledgement. The red line indicates the acknowledgement times for pending packets, while the dotted horizontal lines show the time that each packet incurs delay.

Example 5. *Consider $\rho = 0.5$ and an input formed as $L = \langle a_1 = 1, a_2 = 2, a_3 = 5, a_4 = 7, a_5 = 10, a_6 = 14 \rangle$ of the Dynamic TCP Acknowledgement problem. See Figure 5.1 for illustration. Suppose the prediction vector is $\langle \hat{a}_1 = 1, \hat{a}_2 = 4, \hat{a}_3 = 5, \hat{a}_4 = 6, \hat{a}_5 = 13, \hat{a}_6 = 22 \rangle$. Thus, for the error measures, we have $\ell_\infty = \max\{0, 2, 0, 1, 3, 8\} = 8$.*

5.2.3 Trust: A Simple Consistent Algorithm

The Alarm family of algorithms, introduced by Dooly et al. [2], operates as follows. Upon observing a packet at time a_j, the algorithm sets an alarm for time $a_j + d_j$, where the choice of d_j depends on the specific strategy. To make an acknowledgement, the algorithm waits until the next arrival, a_{j+1}, or the time $a_j + d_j$ (when the alarm "triggers"), whichever comes first. If a_{j+1} arrives before the alarm triggers, it resets the alarm for $a_{j+1} + d_{j+1}$. If the alarm triggers first, it sends an acknowledgement for all pending arrivals at the time of the trigger.

The special case of the Alarm algorithm, where the alarm d_j is set to $\frac{\rho}{1-\rho}$, will result in a 2-competitive algorithm, which is indeed the classic algorithm of [2], and we refer to it as Classic-Deterministic. Recall that no deterministic algorithm can be better than 2-competitive, and thus this variant of Alarm is the optimal deterministic algorithm for the $f_{\max}$ version of Dynamic TCP Acknowledgement.

Dooly et al. [2] also studied the Alarm algorithm in the presence of *lookahead*, which can be interpreted as perfectly predicting the next arrival time. In particular, they show that, with access to the next arrival time a_{j+1}, a member of the Alarm family of algorithms achieves optimality.

In what follows, we study the lookahead algorithm of [2] in the prediction setting. We refer to this algorithm as Trust, which can be described as follows. Recall that the algorithm has access to the next arrival time $\hat{a}_{j+1}$ right after the arrival of packet a_j. Provided with this information, Trust works as follows:

- If $\hat{a}_{j+1} - a_j > d_j$ for $d_j = \frac{\rho}{1-\rho}$, immediately send an acknowledgement at time a_j;
- Otherwise, when $\hat{a}_{j+1} - a_j \leq d_j$ for $d_j = \frac{\rho}{1-\rho}$, don't send an acknowledgement and defer the decision to the next arrival time.

Lemma 5.2.1 ([2]). *The Trust algorithm is optimal (1-competitive) when the predictions are perfect, i.e., when, for all j, we have $\hat{a}_j = a_j$.*

Intuitively, knowing the next arrival helps the algorithm make an optimal decision of either acknowledging immediately or waiting for the next arrival.

Next, we show that if a possibly inaccurate prediction of the next arrival, $\hat{a}_{j+1}$, is given, Trust lacks robustness. To demonstrate this, we first show that Trust has an unbounded competitive ratio for a simple input consisting of two packets.

Lemma 5.2.2. *The Trust algorithm has an unbounded competitive ratio with inaccurate prediction of the next arrival time $\hat{a}_{j+1}$ upon observing each actual arrival a_j.*

Proof. Consider an input formed by two packets $\langle a_1 = 0, a_2 = L \rangle$, where L is an arbitrarily large value. Suppose the prediction at time 0 specifies $\hat{a}_2 = \epsilon$ for sufficiently small ϵ. Therefore, Trust plans to wait until a_2 comes before making the first acknowledgement, resulting in unbounded delay and cost for Trust. However, an optimal algorithm can make two acknowledgements at times a_1 and a_2 for a cost of 2ρ. $\square$

A closer look at the worst-case input of the above shows that Trust can see the prediction is incorrect and yet continues to follow it (by waiting for a_2 to arrive before making an acknowledgement). For such inputs, it is reasonable for the algorithm to give up the prediction once it observes that they are incorrect. The following lemma shows that even this adapted version of Trust cannot achieve a good competitive ratio. In other words, there are worst-case inputs such that Trust never realizes that the predictions were incorrect, and still has an arbitrarily large competitive ratio.

Lemma 5.2.3. *An Adaptive Trust algorithm that ignores predictions when it observe they are incorrect has a competitive ratio of at least $\frac{n\rho}{\rho+n\epsilon(1-\rho)}$ with inaccurate prediction of the next arrival time $\hat{a}_{j+1}$ upon observing each actual arrival a_j.*

Proof. Intuitively, a worst-case scenario occurs when all packets arrive within a short window, implying that making a single acknowledgement is the optimal strategy. For these inputs, if the prediction incorrectly specifies that the next request is sufficiently far from now, Trust makes an acknowledgement upon arrival of any item, resulting in a very high, sub-optimal acknowledgement cost. Next, we formalize these intuitions.

Consider an instance with n arrivals spaced ϵ apart, i.e., $a_{j+1} - a_j = \epsilon$ for all $j \in [n-1]$. At each arrival, the prediction overestimates the next arrival time such that $\hat{a}_{j+1} - a_j > \frac{\rho}{1-\rho}$, leading Trust to send an acknowledgement immediately after each arrival, resulting in an acknowledgement cost of ρn for n items. In contrast, the optimal solution sends a single acknowledgement immediately after the last arrival, resulting in a cost of ρ (for a single acknowledgement) plus $n\epsilon(1-\rho)$ for delays (the max delay time for the single acknowledgement made is $n\epsilon$). The resulting lower bound for the competitive ratio of Trust is thus $\frac{n\rho}{\rho + n\epsilon(1-\rho)}$, which becomes unbounded with sufficiently large n and small ϵ. □

To address the robustness issue identified above, we present a modified algorithm that achieves both consistency and robustness.

5.2.4 Augmented$_{\textbf{Alarm}}(\lambda)$: A Robust and Consistent Algorithm

In this section, we demonstrate that alternative members of the Alarm family of algorithms (See Section 5.2.3 for the description) allow us to achieve both consistency and robustness. We present an algorithm Augmented$_{\text{Alarm}}(\lambda)$, a member of the Alarm family of algorithms, which has a parameter λ, and provides a trade-off between consistency and robustness for the Dynamic TCP Acknowledgement problem under the $f_{\max}$ model.

The algorithm takes a trust parameter $\lambda \in (0, 1)$ to interpolate between fully trusting the predictions and ignoring them entirely. Smaller values of λ yield better performance under accurate predictions, while larger values improve robustness to errors. In particular, the setting $\lambda = 1$ is similar to the Classic-Deterministic algorithm with no prediction, while when λ converges to 0, the algorithm acts similarly to Trust.

The algorithm is defined as follows. Upon arrival of each packet, the prediction of the next arrival falls into one of two cases:

- **Short Alarm:** If the prediction suggests acknowledging immediately, that is, if $\hat{a}_{j+1} - a_j > \frac{\rho}{1-\rho}$, the algorithm sets an alarm for $a_j + \lambda(\frac{\rho}{1-\rho})$. This implies waiting for a time of $\lambda(\frac{\rho}{1-\rho})$ before following the suggestion made by the prediction. In particular, when λ is arbitrarily close to 0, the acknowledgement is done right away (as Trust does) and when $\lambda = 1$, the alarm is set for $a_j + \frac{\rho}{1-\rho}$, as the Classic-Deterministic algorithm.
- **Long Alarm:** If the prediction suggests deferring decision to the next arrival time, that is, if $\hat{a}_{j+1} - a_j \leq \frac{\rho}{1-\rho}$, the algorithm sets its alarm for $a_j + (\frac{1}{\lambda})(\frac{\rho}{1-\rho})$. In other words, it waits for a time $(\frac{1}{\lambda})(\frac{\rho}{1-\rho})$ and if the next packet has not arrived yet, then it makes an acknowledgement. In particular, when λ approaches 0, it sets the alarm to

a large value (in other words, it waits until the next packet arrives) and when $\lambda = 1$, again it sets its alarm for $a_j + \frac{\rho}{1-\rho}$, as the Classic-Deterministic algorithm.

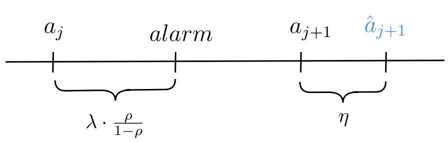
Next, we analyze the competitive ratio of the algorithm as a function of both λ and η. We will demonstrate that adjusting λ provides a trade-off between the algorithm's robustness and consistency.

Theorem 5.2.4. *The competitive ratio of the Augmented$_{Alarm}(\lambda)$ algorithm as a function of the prediction error η can be bounded as follows:*

$$CR(\eta, \lambda) \leq \begin{cases} \max\{1 + \eta(\frac{\rho}{1-\rho}), \frac{(1+\lambda)\rho}{\rho - \eta(1-\rho)}\}, & \textit{if} \ \ \eta \leq (\frac{1}{\lambda} - 1)\frac{\rho}{1-\rho} \\ 1 + \frac{1}{\lambda}, & \textit{if} \ \ \eta > (\frac{1}{\lambda} - 1)\frac{\rho}{1-\rho}. \end{cases}$$

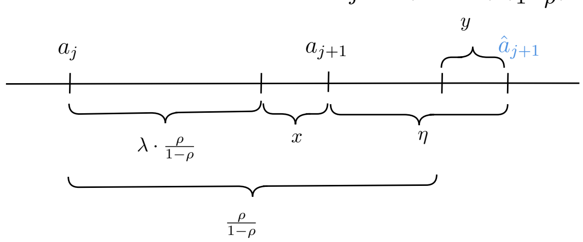
Proof. We consider two scenarios.

- **Short Alarm:** In this case, we have $\hat{a}_{j+1} - a_j > \frac{\rho}{1-\rho}$. That is, the next arrival is predicted to be relatively late, and Augmented$_{\text{Alarm}}(\lambda)$ sets the alarm to $a_j + \lambda(\frac{\rho}{1-\rho})$; this is relatively a short alarm (wait) time before the next acknowledgement.
  - **Good Case:** Suppose $a_{j+1} - a_j > \frac{\rho}{1-\rho}$. So the prediction is consistent with the real arrival time (good). Note that η_j can take any non-negative value (It can be 0 or it can be arbitrarily large when a_{j+1} and $\hat{a}_{j+1}$ take very different values, as long as they are larger than $a_j + \frac{\rho}{1-\rho}$). In this case, given that $\hat{a}_{j+1} - a_j > \frac{\rho}{1-\rho}$, $\textsc{Opt}$ acknowledges at time a_j and has cost ρ. On the other hand, the alarm is $a_j + \lambda(\frac{\rho}{1-\rho}) \leq a_{j+1}$. Therefore, the algorithm acknowledges before the next arrival, and its cost for this event is $\textsc{Alg}_j = \rho + \lambda(\frac{\rho}{1-\rho}) \cdot (1-\rho) = \rho(1+\lambda)$. To summarize, in the Short/Good case, we have $\textsc{Alg}_j \leq (1+\lambda)\textsc{Opt}_j$ and $\eta_j \in [0, \infty)$.



  - **Bad Case:** Suppose $a_{j+1} - a_j \leq \frac{\rho}{1-\rho}$. So the prediction is inconsistent with the real arrival time (bad). For this event, optimal does not acknowledge before a_{j+1} and incurs a delay of $(1-\rho) \cdot (a_{j+1} - a_j)$. Note that η_j could be arbitrary large (when $\hat{a}_{j+1}$ is arbitrarily large). We consider two cases:
    * Suppose $a_{j+1} - a_j \leq \lambda\frac{\rho}{1-\rho}$: Since the alarm is at $a_j + \lambda(\frac{\rho}{1-\rho}) \geq a_{j+1}$, the next arrival occurs before the alarm is triggered. So, the algorithm incurs a delay of $(1-\rho) \cdot (a_{j+1} - a_j)$, and we have $\textsc{Alg}_j = \textsc{Opt}_j = (1-\rho) \cdot (a_{j+1} - a_j)$. The value of η_j is at least $\hat{a}_{j+1} - a_{j+1} \geq (a_j + \frac{\rho}{1-\rho}) - (a_j + \lambda\frac{\rho}{1-\rho}) = (1-\lambda)\frac{\rho}{1-\rho}$.

* Suppose $a_{j+1} - a_j > \lambda\frac{\rho}{1-\rho}$, that is, $a_{j+1} = a_j + \lambda\frac{\rho}{1-\rho} + x$ for some $x \geq 0$. The alarm is $a_j + \lambda\frac{\rho}{1-\rho} \leq a_{j+1}$. The cost of the algorithm is thus $\textsc{Alg}_j = \rho + (1-\rho)\lambda\frac{\rho}{1-\rho} = (1+\lambda)\rho$. On the other hand, $\textsc{Opt}_j = (1-\rho)(a_{j+1} - a_j) = (1-\rho)(\lambda\frac{\rho}{1-\rho} + x) = \lambda\rho + x(1-\rho)$. Moreover, $\eta_j = \hat{a}_{j+1} - a_{j+1} = (a_j + \frac{\rho}{1-\rho} + y) - (a_j + \lambda\frac{\rho}{1-\rho} + x) = \frac{\rho}{1-\rho}(1-\lambda) + y - x$ for some $y \geq 0$; that is $x = \frac{\rho}{1-\rho}(1-\lambda) + y - \eta_j$. So, the cost of optimal is $\textsc{Opt}_j = \lambda\rho + x(1-\rho) = \lambda\rho + \rho(1-\lambda) - (\eta_j - y)(1-\rho)$. Let $\eta'_j = \eta_j - y$; then $\eta'_j = \eta_j - y \in [0, (1-\lambda)(\frac{\rho}{1-\rho})]$. So $\textsc{Opt}_j = \lambda\rho + \rho(1-\lambda) - \eta'_j(1-\rho)$. Thus, $\frac{\textsc{Alg}_j}{\textsc{Opt}_j} \leq \frac{(1+\lambda)\rho}{\rho - \eta'_j(1-\rho)} \leq \frac{(1+\lambda)\rho}{\rho - \eta_j(1-\rho)} \leq \frac{(1+\lambda)\rho}{\rho - \eta(1-\rho)}$. The competitive ratio is maximized when $\eta'_j = (1-\lambda)(\frac{\rho}{1-\rho})$, in which case it becomes $1 + \frac{1}{\lambda}$.



- **Long Alarm:** In this case, we have $\hat{a}_{j+1} - a_j \leq \frac{\rho}{1-\rho}$. That is, the next arrival is predicted to be relatively early. In this case, $\text{Augmented}_{\text{Alarm}}(\lambda)$ sets the alarm to $a_j + \frac{1}{\lambda}(\frac{\rho}{1-\rho})$; this is relatively a long alarm (wait) time before the next acknowledgement.

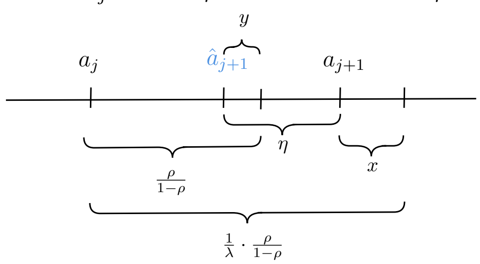
  – **Good Case:** In this case, we have $a_{j+1} - a_j \leq \frac{\rho}{1-\rho}$. So the prediction is relatively good. In particular, we can write $\eta_j = |a_{j+1} - \hat{a}_{j+1}| \in (0, \frac{\rho}{1-\rho}]$. Since $a_{j+1} - a_j \leq \frac{\rho}{1-\rho}$, $\textsc{Opt}$ does not acknowledge and its cost for this event is $a_{j+1} - a_j$. Given that the alarm is $a_j + \frac{1}{\lambda}(\frac{\rho}{1-\rho}) > a_j + \frac{\rho}{1-\rho} = a_{j+1}$, the next arrival happens before the alarm. Therefore, the algorithm does not acknowledge before a_{j+1}, and its cost for this event is $a_{j+1} - a_j$. To summarize, in the Long/Good case, we have $\textsc{Alg}_j = \textsc{Opt}_j = a_{j+1} - a_j$ and $\eta_j \in (0, \frac{\rho}{1-\rho}]$.

  – **Bad Case:** In this case, we have $a_{j+1} - a_j > \frac{\rho}{1-\rho}$. So the prediction is bad. The value of η_j could be arbitrarily large (when a_{j+1} is arbitrarily large). In this case, given that $a_{j+1} - a_j > \frac{\rho}{1-\rho}$, $\textsc{Opt}$ acknowledges immediately after a_j and incurs a cost of ρ.

    We consider two cases:

    * Suppose $a_{j+1} - a_j \leq \frac{1}{\lambda}(\frac{\rho}{1-\rho})$. Suppose $a_{j+1} - a_j = \frac{1}{\lambda}(\frac{\rho}{1-\rho}) - x$ for some $x \geq 0$. Given that $\text{Augmented}_{\text{Alarm}}(\lambda)$ sets its alarm for $a_j + \frac{1}{\lambda}(\frac{\rho}{1-\rho}) \leq a_{j+1}$, the next arrival happens before the alarm is triggered. So, $\textsc{Alg}_j = (1-\rho)(\frac{1}{\lambda}(\frac{\rho}{1-\rho}) - x) = \frac{\rho}{\lambda} - (1-\rho)x$. The value of η is $a_{j+1} - \hat{a}_{j+1} = (a_j + \frac{1}{\lambda}(\frac{\rho}{1-\rho}) - x) - (a_j + \frac{\rho}{1-\rho} - y) = (\frac{1}{\lambda} - 1)\frac{\rho}{1-\rho} - x + y$ for some $y \geq 0$; that is $x = (\frac{1}{\lambda} - 1)\frac{\rho}{1-\rho} - \eta_j + y$. So, the

cost of the algorithm is $\text{ALG}_j = \frac{\rho}{\lambda} - (1-\rho)((\frac{1}{\lambda}-1)\frac{\rho}{1-\rho} - \eta_j + y) = \rho + (\eta_j - y)(1-\rho)$. Let $\eta'_j = \eta_j - y$, then $\eta'_j \in [0, (\frac{1}{\lambda}-1)(\frac{\rho}{1-\rho})]$. So, $\text{ALG}_j = \rho + \eta'_j(1-\rho)$, and $\frac{\text{ALG}_j}{\text{OPT}_j} = \frac{\rho+\eta'_j(1-\rho)}{\rho} = 1 + \eta'_j(\frac{\rho}{1-\rho}) \leq 1 + \eta_j(\frac{\rho}{1-\rho}) \leq 1 + \eta(\frac{\rho}{1-\rho})$.



* Suppose $a_{j+1} - a_j > \frac{1}{\lambda}(\frac{\rho}{1-\rho})$. Since $\text{Augmented}_{\text{Alarm}}(\lambda)$ sets its alarm for $\frac{1}{\lambda}(\frac{\rho}{1-\rho}) \geq a_{j+1} - a_j$, the alarm triggers before the next arrival. Therefore, the algorithm incurs a cost of $\text{ALG}_j = \rho + (1-\rho)\frac{1}{\lambda}\frac{\rho}{1-\rho} = (1+\frac{1}{\lambda})\rho$. So, in this case we have $\text{ALG}_j \leq (\frac{1}{\lambda}+1)\text{OPT}_j$. Note that the value of η_j is at least $(\frac{1}{\lambda}-1)\frac{\rho}{1-\rho}$.

□

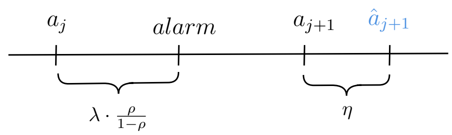
Next, we establish a lower bound for the competitive ratio of $\text{Augmented}_{\text{Alarm}}(\lambda)$.

Theorem 5.2.5. *The competitive ratio of the $\text{Augmented}_{\text{Alarm}}(\lambda)$ algorithm as a function of the prediction error η is at least*

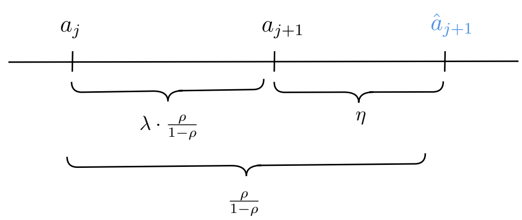
$$CR(\eta, \lambda) \geq \begin{cases} 1+\lambda, & \text{if } \eta \leq (1-\lambda)\frac{\rho}{1-\rho} \\ 1+\frac{1}{\lambda}, & \text{if } \eta > (1-\lambda)\frac{\rho}{1-\rho}. \end{cases}$$

Proof. We consider three worst-case inputs, each establishing the desired lower bound for the competitive ratio of $\text{Augmented}_{\text{Alarm}}(\lambda)$ for different values of η.

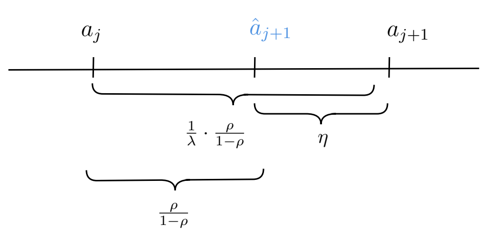
- Let η be any arbitrary value of at most $(1-\lambda)\frac{\rho}{1-\rho}$. Consider the input arrival times are $\sigma_1 = \langle a_1, \ldots, a_n \rangle$, where $a_j = L \times j$ for sufficiently large values of L. Given that all packets are far from each other, the optimal algorithm thus makes n acknowledgements upon the arrival of any item, which implies $\text{OPT}(\sigma_1) = n\rho$. Suppose the prediction for any a_j is $\hat{a}_j = a_j + \eta$, which implies the error value is η. Given that $\hat{a}_{j+1} - a_j = L + \eta > \frac{\rho}{1-\rho}$, the algorithm waits for a time of $\lambda(\frac{\rho}{1-\rho})$ before making an acknowledgement (we are always in Short Alarm case). The cost of $\text{Augmented}_{\text{Alarm}}(\lambda)$ for each packet is thus $\rho + \lambda(\frac{\rho}{1-\rho}) \times (1-\rho) = \rho(1+\lambda)$. Note that the algorithm makes n acknowledgements as OPT does, and thus the ratio between the cost of $\text{Augmented}_{\text{Alarm}}(\lambda)$ and OPT for σ_1 is $1+\lambda$.



- Let η be any arbitrary value larger than $(1-\lambda)\frac{\rho}{1-\rho}$, that is $\eta = (\frac{1}{\lambda} - 1)\frac{\rho}{1-\rho} + \delta$ for some positive δ. Consider the input arrival times are $\sigma_2 = \langle a_1, \ldots, a_n \rangle$, where $a_j = (\frac{\rho}{1-\rho} + \epsilon - \eta) \times j$ for sufficiently small values of ϵ. One solution for this input is to acknowledge only at the end of the instance; thus $\text{OPT}(\sigma_2) \leq \lambda\rho n + \rho$. Now, suppose the prediction specifies that, for any $j \in [n]$, $\hat{a}_j = a_j + \eta$; thus, the prediction error is exactly η. Now, given that, $\hat{a}_{j+1} - a_j = a_{j+1} + \eta - a_j = \frac{\rho}{\rho-1} + \epsilon > \frac{\rho}{\rho-1}$. Therefore, we are in the Short Alarm case, and the algorithm sets the alarm for $a_j + \lambda\frac{\rho}{1-\rho}$. Moreover, given that $\eta = (1-\lambda)\frac{\rho}{1-\rho} + \delta$, we have $a_{j+1} = a_j + \frac{\rho}{1-\rho} + \epsilon - \eta > a_j + \lambda\frac{\rho}{1-\rho}$. We can conclude that, the algorithm makes an acknowledgement at time $a_j + \lambda\frac{\rho}{1-\rho}$, and its cost for each acknowledgement would be $\rho + (1-\rho)\lambda\frac{\rho}{1-\rho} = (1+\lambda)\rho$. This sums up to $(1+\lambda)n\rho$ for all n acknowledgements, and thus the ratio between the cost of $\text{Augmented}_{\text{Alarm}}(\lambda)$ and OPT for σ_2 is $1 + \frac{1}{\lambda}$ for arbitrarily large n.



- Let η be any arbitrary value larger than $(\frac{1}{\lambda} - 1)\frac{\rho}{1-\rho}$, that is $\eta = (\frac{1}{\lambda} - 1)\frac{\rho}{1-\rho} + \delta$ for some positive δ. Consider the input arrival times are $\sigma_3 = \langle a_1, \ldots, a_n \rangle$, where $a_j = (\frac{\rho}{1-\rho} - \epsilon + \eta) \times j$ for sufficiently small values of ϵ (say $\epsilon = \delta/n^2$). One solution for this input is to acknowledge every packet separately; thus $\text{OPT}(\sigma_3) \leq \rho n$. Now, suppose the prediction specifies that, for any $j \in [n]$, $\hat{a}_j = a_j - \eta$; thus, the prediction error is exactly η. Now, given that, $\hat{a}_{j+1} - a_j = a_{j+1} - \eta - a_j = \frac{\rho}{\rho-1} - \epsilon < \frac{\rho}{\rho-1}$. Therefore, we are in the Long Alarm case, and the algorithm sets the alarm for $a_j + \frac{1}{\lambda}\frac{\rho}{1-\rho}$. Moreover, given that $\eta = (\frac{1}{\lambda} - 1)\frac{\rho}{1-\rho} + \delta$, we have $a_{j+1} = a_j + \frac{\rho}{1-\rho} - \epsilon + \eta > a_j + \frac{1}{\lambda}\frac{\rho}{1-\rho}$. We can conclude that, the algorithm makes an acknowledgement at time $a_j + \frac{1}{\lambda}\frac{\rho}{1-\rho}$, and its cost for each acknowledgement would be $\rho + (1-\rho)\frac{1}{\lambda}\frac{\rho}{1-\rho} = (1 + \frac{1}{\lambda})\rho$.



This sums up to $(1 + \frac{1}{\lambda})n\rho$ for all n acknowledgements, and thus the ratio between the cost of $\text{Augmented}_{\text{Alarm}}(\lambda)$ and $\textsc{Opt}$ for σ_3 is $1 + \frac{1}{\lambda}$. $\square$

Corollary 5.2.6. *The $\text{Augmented}_{\text{Alarm}}(\lambda)$ algorithm is $\{1 + \lambda, 1 + \frac{1}{\lambda}\}$-competitive.*

Proof. According to Theorems 5.2.4 and 5.2.5, when the prediction is accurate, meaning η is zero, the competitive ratio is $\lambda + 1$. In contrast, as the prediction error increases and the prediction becomes untrusted, the competitive ratio degrades to $\frac{1}{\lambda}$. $\square$

5.3 Experiments

In this section, we evaluate the performance of the studied algorithms under various input sequences and different levels of prediction error, using both synthetic and real-world data. We aim to validate the theoretical guarantees from earlier sections and compare the algorithms' practical performance with their theoretical bounds.

5.3.1 Datasets

Synthetic data. The distribution of TCP packet arrival times is traditionally assumed to follow some Poisson distribution. That is, the packet arrival times are generated by taking the cumulative sum of inter-arrival times (time difference between arrival times of consecutive packets), which is an exponential distribution, resulting in a Poisson model. The scale parameter, a value such as 0.3, defines the mean of the inter-arrival times generated. Later, it was shown that this model is not always representative of reality [116]. In practice, there is often a significant probability of a huge number of packets arriving within a short time interval. To better capture these characteristics, heavy-tailed distributions such as the Pareto distribution are frequently used in more recent studies such as [108]. Pareto distribution is a distribution that has a power-law property, that is, a few smaller values (inter-arrival time) are very frequent, while a huge number of large values are relatively scarce. This motivates our use of similar distributions for modelling random packet arrival instances in our experiments. We use a Pareto distribution in our experiments, with shape parameter $\alpha = 3$ and scale 0.5, which controls the arrival density and tail behavior. Precisely, the shape parameter controls the tail behavior; a larger α indicates a heavier tail. The scale parameter represents the statistical dispersion of the distribution.

Real-world data. The MAWI Working Group Traffic Archive [113] is a long-standing repository of real-world Internet traffic traces collected from a trans-Pacific backbone link between Japan and the United States. Each trace consists of 15 minutes of packet-level data captured daily, with anonymized IP headers. In our experiments, we use traces from MAWI to evaluate the performance of our algorithms on real-world packet arrival sequences. Since the MAWI dataset has fine-grained timestamps, arrival times are rescaled to ensure a mean inter-arrival time of 0.2.

5.3.2 Implementation Details

The parameter ρ is set to 0.5, making the cost of delaying a packet by one time unit equal to the cost of sending an acknowledgement. To apply prediction error, each arrival time is perturbed by additive noise drawn uniformly at random from the interval $[-\eta, \eta]$, where η represents the prediction error level and ranges between 0 and 1.5.

5.3.3 Algorithms

The performance of the $\text{Augmented}_{\text{Alarm}}(\lambda)$ algorithm is evaluated for different values of λ in $\{0.05, 0.1, 0.2, 0.4, 0.6, 0.8\}$. For comparison, we also include the Trust algorithm (λ converges to 0) and the Classic-Deterministic algorithm ($\lambda = 1$) as baselines.

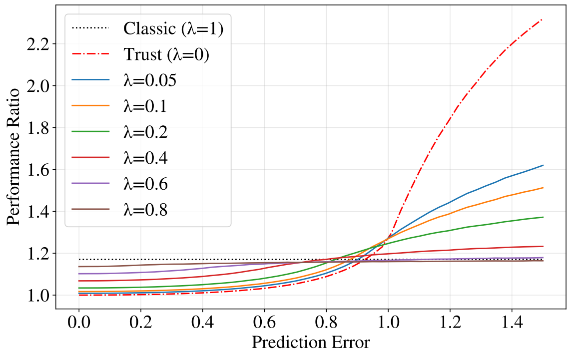
5.3.4 Metrics

All experiments are repeated over 1000 independent trials consisting of 1000 arrivals, and the *average performance ratio* across trials is reported. For that, we compute the average value of the ratio between the cost of $\textsc{Alg}$ and that of $\textsc{Opt}$ over the 1000 trials.

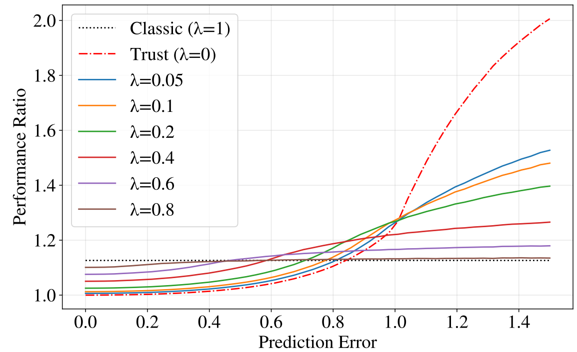
5.3.5 Experimental Analysis

The plots in Figure 5.2 compare the average performance ratio of different algorithms across varying levels of prediction error. When prediction is perfect ($\eta = 0$), the optimal strategy is to follow the prediction without adjustment. In this case, the Trust algorithm (λ converges to 0) achieves a performance ratio of 1, and smaller values of λ consistently yield better performance. For larger prediction errors, smaller values of λ result in a sharper increase in the performance ratio. As shown in Figure 5.2, the experimental results are stable across all distributions and align well with our theoretical results.

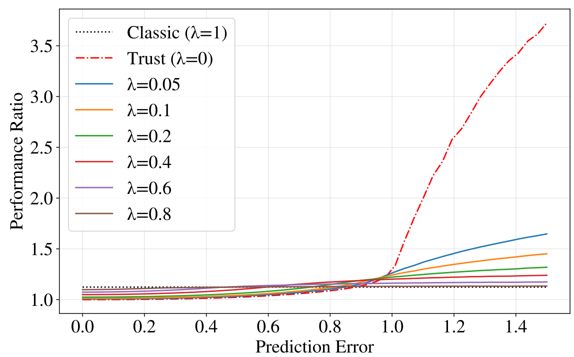
Figure 5.2a shows the performance on Pareto-distributed arrivals, where the heavy-tailed distribution creates challenging scenarios with both short bursts and long quiet periods. The



(a) Pareto-distributed arrivals.



(b) Poisson-distributed arrivals.



(c) MAWI dataset arrivals.

Figure 5.2: Performance ratio of the evaluated algorithms as a function of prediction error η, across three arrival models: (a) synthetic sequences generated using a Pareto distribution, (b) synthetic sequences from a Poisson distribution, and (c) real-world sequences extracted from the MAWI traffic trace dataset.

results demonstrate that algorithms with $\lambda \geq 0.4$ maintain relatively stable performance, with the Trust algorithm showing explosive growth in performance ratio as prediction errors increase beyond $\eta = 0.6$.

Figure 5.2b presents results for Poisson-distributed arrivals, which exhibit smoother performance curves due to the memoryless property of the inter-arrival times (i.e., the distribution of the next arrival times does not depend on how many trials have already passed). This property allows smaller λ values to maintain good performance over a wider range of prediction errors. There is a more gradual increase in average performance ratio for λ values larger than 0.6.

Figure 5.2c reveals the most complex behavior on real-world MAWI data, showing significantly higher performance ratios overall and more pronounced sensitivity to prediction errors, especially in the Trust algorithm. When evaluated on this dataset, the Trust algorithm reaches performance ratios above 3.0, highlighting the challenges of real-world network traffic patterns.

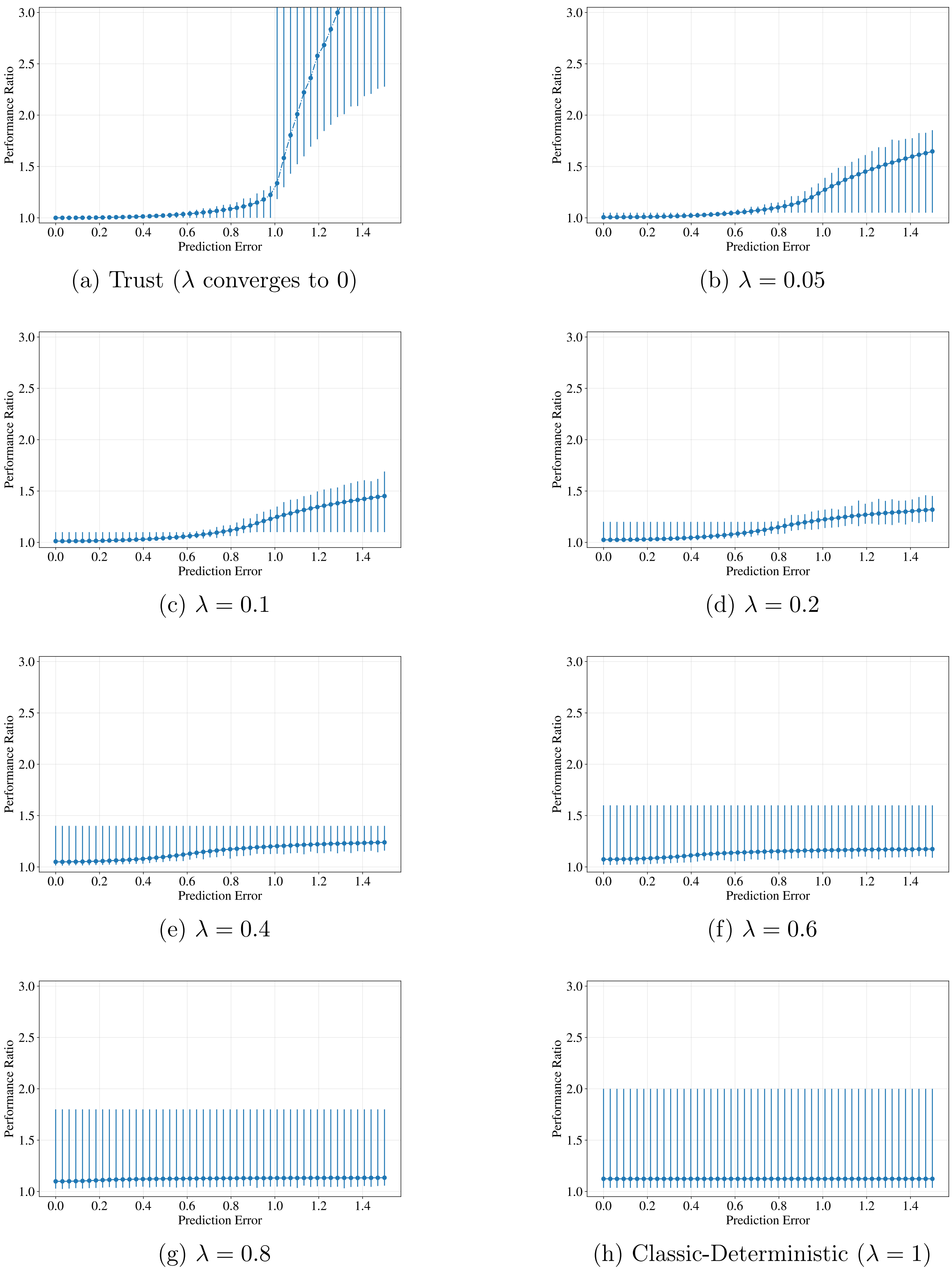
Figure 5.3 presents error bars for each algorithm, showing the maximum, minimum, and average performance ratios, over the 1000 trials, as a function of prediction error. The Trust algorithm (Figure 5.3a) shows not only poor average performance but also high variance, particularly at higher error levels. Small λ values (Figures 5.3b,5.3c) demonstrate increasing variance as prediction errors grow, suggesting less reliable performance under uncertainty. However, moderate λ values show more consistent performance with smaller error bars, confirming their robustness properties (Figure 5.3d,5.3e). High λ values maintain the most stable performance with minimal variance across all error levels (Figure 5.3g). The Classic-Deterministic algorithm maintains the same performance ratio across all error levels, as it entirely ignores the prediction. Its maximum performance ratio is 2, consistent with the theoretical upper bound established in [2].

Figure 5.4 illustrates the Trust algorithm's performance on MAWI data at its original scale. It shows how the performance ratio can exceed 140 under high prediction errors. This extreme performance degradation highlights the importance of robustness in real-world applications, where prediction quality cannot be guaranteed. The results validate the theoretical analysis (Lemmas 5.2.2 and 5.2.3), showing that algorithms relying solely on predictions can have an unbounded performance ratio when those predictions are inaccurate.

5.4 Algorithm with Advice

Recall that prediction and advice are closely related. Indeed, advice can be thought of as a form of perfect (error-free) prediction for which the main concern is the *size* of advice (rather than its quality and predictability, which are the concerns in the prediction setting). As mentioned in Section 1.4.3, reductions from the Binary String Guessing problem, which asks for predicting the value of the next bit in a sequence of bits that is revealed online, are often used to establish lower bounds for the competitive ratio of various problems when the advice size is sublinear.

Here, we reduce the Binary String Guessing problem to the Dynamic TCP Acknowledgement problem to show a lower bound for the competitive ratio of TCP acknowledgement algorithms (under $f_{\max}$ setting) to show *any* algorithm with sublinear advice cannot be better than 1.1716-competitive. This means that predictions of sublinear size, even if they are perfect (they are advice), cannot help in achieving almost-optimal solutions. This includes many families of PAC-learnable predictions, e.g., predictions concerning statistical information about the input sequence (e.g., average, mean, and standard deviation



(a) Trust (λ converges to 0)

(b) $\lambda = 0.05$

(c) $\lambda = 0.1$

(d) $\lambda = 0.2$

(e) $\lambda = 0.4$

(f) $\lambda = 0.6$

(g) $\lambda = 0.8$

(h) Classic-Deterministic ($\lambda = 1$)

Figure 5.3: Performance ratios of the evaluated algorithms across varying levels of prediction error. Arrival patterns are derived from the MAWI dataset (error bars are shown).

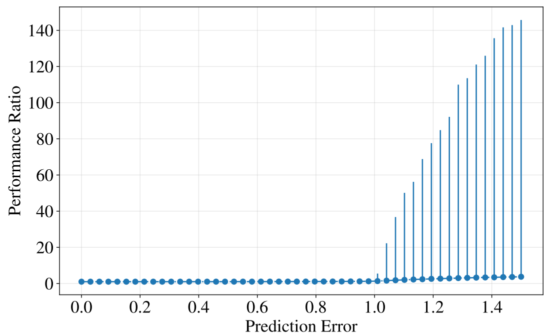


Figure 5.4: Performance of the Trust algorithm on the MAWI dataset in its original scale.

of inter-arrival times).

Theorem 5.4.1. *In the* Dynamic TCP Acknowledgement *problem under the* f_{max} *objective, no algorithm with* $o(n)$ *bits of advice can achieve a competitive ratio better than* 1.1716.

Proof. We describe a reduction from Binary String Guessing to Dynamic TCP Acknowledgement as follows. Let s be an online sequence of m bits that forms an input to the Binary String Guessing problem. We create an instance σ_S of TCP acknowledgement, formed by m "phases", as follows. The i'th phase in σ_S takes two possible forms, depending on the i'th bit in S. Regardless, the i'th phase starts with a packet that arrives at time $i \cdot C$, for a sufficiently large value of C (e.g., $C = m \times \rho$). Now, if $S[i] = 1$, the phase ends, and no more packets come in the i'th phase. On the other hand, if $S[i] = 0$, a second packet appears at time $i \cdot C + x$, where $x < \frac{\rho}{1-\rho}$ is a small value that we will determine later.

Intuitively, the long distance between two phases implies that a reasonable algorithm does not defer acknowledging packets at phase i to the next phase. More importantly, upon arrival of the first packet (at time $i \cdot C$), the algorithm must "guess" whether the phase of type "1", where no more packet comes and the acknowledgement should be made immediately, or of type "0", where another packet comes soon and it is best to wait for an extra x time.

The algorithm's decisions for acknowledging packets at phase i translate to making binary guesses for the value of the ith bit. In particular, if the algorithm sends an acknowledgement before $i \cdot C + x$, the i'th bit in S is guessed as "0", and otherwise "1".

Depending on the guess made by the algorithm and the actual i'th bit in S, we get four possibilities for the cost of the algorithm, which are summarized as follows:

bit	cost of correct guess	cost of wrong guess
0	$(1-\rho)x+\rho$	$\geq 2\rho$
1	ρ	$\geq (1-\rho)x+\rho$

The above costs can be described as follows. In a phase of type "0", if the algorithm guesses correctly, it waits for x time after the first arrival and sends a single acknowledgement immediately after the second arrival. However, if it guesses incorrectly, it sends an acknowledgement for the first arrival and, since the next phase begins after a gap of at least $\frac{\rho}{1-\rho}$, it must send a second acknowledgement for the second arrival. Thus, it pays at least 2ρ.

In a phase of type "1", if the algorithm guesses correctly, it sends an acknowledgement immediately upon seeing the arrival. However, if it guesses incorrectly, it waits for x time expecting a second arrival. When no additional arrival occurs, it realizes the guess was incorrect and, in the best case, sends an acknowledgement immediately after the wait. Thus, it incurs a cost of at least $(1-\rho)x+\rho$.

We choose $x = \frac{(\sqrt{2}-1)\rho}{1-\rho}$. Substituting this value into the previously derived cost expressions, the worst-case ratio between the cost of an incorrect guess and its associated correct guess becomes at least $\sqrt{2}$ in for both types of phases (precisely, we will have $\frac{2\rho}{(1-\rho)x+\rho} = \frac{(1-\rho)x+\rho}{\rho} = \sqrt{2}$). Note that in phases of type "0" (i.e., two arrivals), both the correct and incorrect guess costs take larger values compared to phases of type "1" (i.e., a single arrival). That is, we have $2\rho > (1-\rho)x+\rho$ and $(1-\rho)x+\rho > \rho$. Since the ratio between incorrect and correct guess costs is the same for both phase types, the best scenario for the algorithm in this setting occurs when the algorithm incorrectly guesses phases of type "1" as "0".

Recall that with $o(n)$ bits of advice, one cannot correctly guess a fraction $1/2+\epsilon$ of bits in the instance S of the Binary String Guessing problem, for any $\epsilon > 0$ [47]. This bound is achieved by always predicting the majority bit, which can be identified using a single bit of advice. The cost of the optimal solution that always guesses correctly is at most

$$\textsc{Opt} \leq \frac{n}{2}((1-\rho)x+\rho) + \frac{n}{2}(\rho) = \frac{\sqrt{2}+1}{2}n\rho.$$

And the cost of any algorithm is at least

$$\textsc{Alg} \geq n((1-\rho)x+\rho) = \sqrt{2}n\rho.$$

Therefore, the competitive ratio is at least

$$\mathrm{CR} \geq \frac{\sqrt{2}n\rho}{\frac{\sqrt{2}+1}{2}n\rho} = \frac{2\sqrt{2}}{\sqrt{2}+1} \approx 1.1715.$$

This establishes a lower bound on the competitive ratio of any algorithm with sublinear advice and thus completes the proof. $\square$

Chapter 6

Concluding Remarks

In this thesis, we investigated the Timely Bin Packing and Dynamic TCP Acknowledgement problems under various settings and assumptions. In what follows, we review our results and discuss some open problems for future work.

Timely Bin Packing & variants. A summary of our contributions for the Timely Bin Packing problem is provided in Table 4.1. For the offline (and deadline-aware online) settings, we have designed the Discrete-Greedy, which has an approximation factor of at most 2. Designing algorithms with lower approximation ratios can be a direction for future research. For the online setting, although we tightened the gap between the lower and upper bounds of the competitive ratio (for various settings), there are remaining gaps to be addressed. In particular, for the deadline-aware setting, the best competitive ratio is an unknown value in the range $(1.5, 2)$, while for the deadline-oblivious setting, the best ratio is in the range $(2, 3)$.

For the advice complexity, we established a lower bound of $1.5 - \epsilon$ for the competitive ratio of algorithms with $o(n)$ bits of advice. At the same time, with $\Omega(n \log(\text{Opt}))$, one can encode the entire optimal solution. For instances of bounded depth, we presented an algorithm that can encode the optimal solution with $\mathcal{O}(n)$ bits of advice. An interesting open question here is whether we can achieve a better competitive ratio (possibly an optimal solution) with $\Theta(n)$ bits of advice for general instances or not.

This thesis explores various settings and restrictions on instances to relax the Timely Bin Packing problem. A summary of the case study results is available in Table 4.2. However, in most settings, gaps still remain between the lower and upper bounds for attainable competitive ratios; tightening/closing these gaps is a good topic for future work. For example, instances with uniform-sized items can be solved optimally in a deadline-aware setting. However, in the deadline-oblivious setting, the lower and upper bounds for the competitive ratio are respectively 2 and 3. Also, one can study the power and limitations of randomization in these settings.

Another promising direction is to study the TIMELY BIN PACKING problem under relaxed settings, such as a setting where algorithms are augmented with machine-learned predictions. These predictions could convey information about forthcoming items, such as their size, arrival time, or deadline. Alternatively, they may provide statistical information, like the frequency distribution of item sizes.

Timely Bin Covering Problem. The bin covering problem [117] is often seen as the dual of bin packing. In bin covering, items of different sizes must be packed into bins so that each bin reaches at least a specified minimum total size. The goal is to maximize the number of bins that are successfully covered.

Bin covering can be naturally studied in a "timely" setting, where each item has an availability interval, and items can be packed if their intervals intersect. Unlike bin packing, however, we want to maximize the number of delivered bins. The following theorem shows that, even under a simple setting of restricted-size items, the online timely bin covering problem does not admit a bounded competitive ratio (note that bin covering is a maximization problem, and the competitive ratio of an online algorithm A is the maximum ratio between the profit of Opt and that of A).

Theorem 6.0.1. *No deterministic algorithm can have a competitive ratio better than $\Omega(n)$ for the online timely bin covering problem, even for restricted inputs in which all item sizes are in $\{1/n, 1-1/n\}$.*

Proof. The adversary first sends an item $a_1 = (s_1 = 1/n, r_1 = 0, d_1 = 1)$. It also sends $n-1$ other items, $b_i = (1/n, 0, 3n+1)$.

- If the algorithm does not cover a bin with a_1 and all b_is before time 1 (the deadline of b_1), the adversary ends the input instance. In this case, OPT has a profit of 1 as all items together can cover a bin, while the algorithm does not cover any bin. Thus, the competitive ratio would be infinity.

- Otherwise, when the algorithm covers a bin using all a_1 and all b_i, the adversary sends $n-1$ more items $c_i = (1-1/n, 3i, 3i+1)$. As all c_is are pairwise disjoint (and b_i's are already used to deliver a bin with a_1), the algorithm cannot cover any bin with them. The optimal solution, on the other hand, can cover every c_i with one item b_i that ends at time $3n+1$. Hence, the competitive ratio would be $n-1$. □

Since one cannot hope for a positive result concerning deterministic algorithms for the bin covering problem, a promising direction for future work is to explore randomization and relaxed variants, such as predictions. Predictions could provide information about the size of the items, statistical properties of the input sequence, or even hints about how the optimal solution might pack items together.

Dynamic TCP Acknowledgement. For the DYNAMIC TCP ACKNOWLEDGEMENT problem, we designed an algorithm and established matching upper and lower bounds on its competitive ratio as a function of the prediction error. Exploring the optimal consistency-robustness tradeoff for DYNAMIC TCP ACKNOWLEDGEMENT with prediction remains a promising direction for future work. In particular, establishing an impossibility result that shows our algorithm is Pareto-optimal with respect to consistency and robustness is an interesting topic for further investigation.

Recall that three objectives have been proposed for the DYNAMIC TCP ACKNOWLEDGEMENT problem, namely, f_{sum}, $f_{\max}$, and a third objective, which is maximum delay over all packets in the input (See Section 3.2). While previous results and our work have studied the first two objectives under prediction settings, investigating the third one when provided with predictions is an interesting topic for further exploration.

Another promising direction for future research in the DYNAMIC TCP ACKNOWLEDGEMENT is to explore whether instance-specific properties can lead to improved competitive bounds. For example, one can study a *bounded inter-arrival times* which guarantees that the arrival times of any two consecutive packets differ by at most a constant d, or in contrast, sparse inputs in which the number of arrived intervals in any window of length w is bounded (by a constant or say sublinear function of w). Studying purely online or learning-augmented algorithms for these restricted settings is an interesting avenue for future research.

Bibliography

[1] Zhadyra Khattar. Online timely bin packing. Master's thesis, Utrecht University, 2023.

[2] Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. Online analysis of the TCP acknowledgment delay problem. *Journal of ACM*, 48(2):243–273, 2001.

[3] Dritan Nace and Michal Pióro. Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial. *IEEE Communications surveys & tutorials*, 10(4):5–17, 2009.

[4] Edward G. Coffman, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: Survey and classification. In *Handbook of Combinatorial Optimization*, pages 455–531. Springer, 2013.

[5] Thomas Rothvoß. Approximating bin packing within $O(\log OPT. \log \log OPT)$ bins. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 20–29, 2013.

[6] David S Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[7] Chan C Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *Journal of the ACM*, 32(3):562–572, 1985.

[8] Steven S Seiden. On the online bin packing problem. *Journal of the ACM*, 49(5): 640–671, 2002.

[9] János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. *Theoretical Computer Science*, 440:1–13, 2012.

[10] Edward G Coffman, Michael R Garey, and David S Johnson. Approximation algorithms for bin-packing—an updated survey. In *Algorithm design for computer system design*, pages 49–106. Springer, 1984.

[11] Jiří Sgall. Online bin packing: Old algorithms and new results. In *Conference on Computability in Europe*, pages 362–372. Springer, 2014.

[12] Henrik I Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

[13] Yusen Li, Xueyan Tang, and Wentong Cai. On dynamic bin packing for resource allocation in the cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 2–11, 2014.

[14] Shahin Kamali and Alejandro López-Ortiz. Efficient online strategies for renting servers in the cloud. In *SOFSEM 2015: Theory and Practice of Computer Science*, pages 277–288, 2015.

[15] Mahtab Masoori, Lata Narayanan, and Denis Pankratov. Renting servers in the cloud: Parameterized analysis of firstfit. In *Proceedings of the 25th International Conference on Distributed Computing and Networking*, pages 199–208, 2024.

[16] Yaqiao Li, Mahtab Masoori, Lata Narayanan, and Denis Pankratov. Renting servers for multi-parameter jobs in the cloud. In *Proceedings of the 26th International Conference on Distributed Computing and Networking*, pages 36–45, 2025.

[17] Mahtab Masoori, Lata Narayanan, and Denis Pankratov. Renting servers in the cloud: The case of equal duration jobs. *Discrete Applied Mathematics*, 362:82–99, 2025.

[18] Michele Flammini, Gianpiero Monaco, Luca Moscardelli, Hadas Shachnai, Mordechai Shalom, Tami Tamir, and Shmuel Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. *Theoretical Computer Science*, 411(40-42): 3553–3562, 2010.

[19] Peter Winkler and Lisa Zhang. Wavelength assignment and generalized interval graph coloring. In *Proceedings of the fourteenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 830–831, 2003.

[20] Mordechai Shalom, Ariella Voloshin, Prudence WH Wong, Fencol CC Yung, and Shmuel Zaks. Online optimization of busy time on parallel machines. *Theoretical Computer Science*, 560:190–206, 2014.

[21] Rohit Khandekar, Baruch Schieber, Hadas Shachnai, and Tami Tamir. Minimizing busy time in multiple machine real-time scheduling. In *IARCS Annual Conference*

on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010), pages 169–180, 2010.

[22] Runtian Ren and Xueyan Tang. Online flexible job scheduling for minimum span. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 55–66, 2017.

[23] Vincent Chau and Minming Li. Active and busy time scheduling problem: A survey. In *Complexity and Approximation: In Memory of Ker-I Ko*, pages 219–229. Springer, 2020.

[24] Susanne Albers and G Wessel van der Heijden. Online busy time scheduling with flexible jobs. *arXiv preprint arXiv:2405.08595*, 2024.

[25] Edward G Coffman, Michael R Garey, and David S Johnson. Dynamic bin packing. *SIAM Journal on Computing*, 12(2):227–258, 1983.

[26] Prudence WH Wong, Fencol CC Yung, and Mihai Burcea. An 8/3 lower bound for online dynamic bin packing. In *International Symposium on Algorithms and Computation*, pages 44–53. Springer, 2012.

[27] Yossi Azar and Danny Vainstein. Tight bounds for clairvoyant dynamic bin packing. *ACM Transactions on Parallel Computing (TOPC)*, 6(3):1–21, 2019.

[28] Niv Buchbinder, Yaron Fairstein, Konstantina Mellou, Ishai Menache, and Joseph Naor. Online virtual machine allocation with lifetime and load predictions. *ACM SIGMETRICS Performance Evaluation Review*, 49(1):9–10, 2021.

[29] Xueyan Tang, Yusen Li, Runtian Ren, and Wentong Cai. On first fit bin packing for online cloud server allocation. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 323–332, 2016.

[30] Leah Epstein. On bin packing with clustering and bin packing with delays. *Discrete Optimization*, 41:100647, 2021.

[31] Yossi Azar, Yuval Emek, Rob van Stee, and Danny Vainstein. The price of clustering in bin-packing with applications to bin-packing with delays. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–10, 2019.

[32] Spyros Angelopoulos, Shahin Kamali, and Kimia Shadkami. Online bin packing with predictions. *Journal of Artificial Intelligence Research*, 78:1111–1141, 2023.

[33] Lauri Ahlroth, André Schumacher, and Pekka Orponen. Online bin packing with delay and holding costs. *Operations Research Letters*, 41(1):1, 2013.

[34] Bozidar Radunovic and Jean-Yves Le Boudec. A unified framework for max-min and min-max fairness with applications. *IEEE/ACM Transactions on networking*, 15(5): 1073–1083, 2007.

[35] Leonidas Georgiadis, Panos Georgatsos, Kostas Floros, and Stelios Sartzetakis. Lexicographically optimal balanced networks. In *IEEE Conference on Computer Communications*, volume 2, pages 689–698, 2001.

[36] Ashish Chiplunkar, Monika Henzinger, Sagar Sudhir Kale, and Maximilian Vötsch. Online min-max paging. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1545–1565, 2023.

[37] Sylvain Bouveret, Yann Chevaleyre, and Nicolas Maudet. Fair allocation of indivisible goods. *Handbook of Computational Social Choice*, pages 284–310, 2016.

[38] Jérôme Lang and Jörg Rothe. Fair division of indivisible goods. In *Economics and Computation: An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*, pages 605–680. Springer, 2024.

[39] Georgios Amanatidis, Haris Aziz, Georgios Birmpas, Aris Filos-Ratsikas, Bo Li, Hervé Moulin, Alexandros A Voudouris, and Xiaowei Wu. Fair division of indivisible goods: Recent progress and open questions. *Artificial Intelligence*, 322:103965, 2023.

[40] R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce*. Association for Computing Machinery, 2004.

[41] Dries Cornilly, Giovanni Puccetti, Ludger Rüschendorf, and Steven Vanduffel. Fair allocation of indivisible goods with minimum inequality or minimum envy. *European Journal of Operational Research*, 297(2):741–752, 2022.

[42] Daman Deep Singh, Amit Kumar, and Abhijnan Chakraborty. Towards fairness in online service with k servers and its application on fair food delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19929–19936, 2024.

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

[44] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 2005.

[45] Joan Boyar, Lene M Favrholdt, Christian Kudahl, Kim S Larsen, and Jesper W Mikkelsen. Online algorithms with advice: A survey. *ACM Computing Surveys*, 50(2): 1–34, 2017.

[46] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011.

[47] Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theoretical Computer Science*, 554:95–108, 2014.

[48] Hans-Joachim Böckenhauer, Elisabet Burjons, Juraj Hromkovic, Henri Lotze, and Peter Rossmanith. Online simple knapsack with reservation costs. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 187, pages 16:1–16:18, 2021.

[49] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM*, 68(4):1–25, 2021.

[50] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1834–1845, 2020.

[51] Alexander Wei. Better and Simpler Learning-Augmented Online Caching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 176, pages 60:1–60:17, 2020.

[52] Sungjin Im, Ravi Kumar, Aditya Petety, and Manish Purohit. Parsimonious learning-augmented caching. In *International Conference on Machine Learning (ICML)*, pages 9588–9601, 2022.

[53] Jakub Chłędowski, Adam Polak, Bartosz Szabucki, and Konrad Tomasz Żołna. Robust learning-augmented caching: An experimental study. In *International Conference on Machine Learning (ICML)*, pages 1920–1930, 2021.

[54] Nikhil Bansal, Christian Coester, Ravi Kumar, Manish Purohit, and Erik Vee. Learning-augmented weighted paging. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 67–89, 2022.

[55] Antonios Antoniadis, Christian Coester, Marek Eliáš, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *ACM transactions on algorithms*, 19(2):1–34, 2023.

[56] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. *ACM Transactions on Algorithms*, 18(4):1–27, 2022.

[57] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

[58] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. *Advances in Neural Information Processing Systems (NeurIPS)*, 31: 9661–9670, 2018.

[59] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Communications of the ACM*, 65(7):33–35, 2022.

[60] Juris Hartmanis. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *SIAM Review*, 24(1):90, 1982.

[61] Brenda S Baker. A new proof for the first-fit decreasing bin-packing algorithm. *Journal of Algorithms*, 6(1):49–70, 1985.

[62] Minyi Yue. A simple proof of the inequality $FFD(L) \leq 11/9OPT(L) + 1, \forall L$ for the ffd bin-packing algorithm. *Acta Mathematicae Applicatae Sinica*, 7:321–331, 1991.

[63] Rongheng Li and Minyi Yue. The proof of $FFD(L) \leq OPT(L) + 7/9$. *Chinese Science Bulletin*, 42:1262–1265, 1997.

[64] György Dósa. The tight bound of first fit decreasing bin-packing algorithm is $FFD(I) \leq 11/9OPT(I) + 6/9$. In *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11. Springer, 2007.

[65] David S Johnson and Michael R Garey. A 7160 theorem for bin packing. *Journal of Complexity*, 1(1):65–106, 1985.

[66] Wenceslas Fernandez de la Vega and George S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

[67] Narendra Karmarkar and Richard M Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 312–320, 1982.

[68] Rebecca Hoberg and Thomas Rothvoß. A logarithmic additive integrality gap for bin packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2616–2625, 2017.

[69] Joan Boyar, Shahin Kamali, Kim S Larsen, and Alejandro López-Ortiz. Online bin packing with advice. *Algorithmica*, 74:507–527, 2016.

[70] Spyros Angelopoulos, Christoph Dürr, Shahin Kamali, Marc P Renault, and Adi Rosén. Online bin packing with advice of small size. *Theory of Computing Systems*, 62:2006–2034, 2018.

[71] Jesper W Mikkelsen. Randomization can be as helpful as a glimpse of the future in online computation. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 39–1, 2016.

[72] Sashka Davis, Jeff Edmonds, and Russell Impagliazzo. Online algorithms to minimize resource reallocations and network communication. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 104–115, 2006.

[73] John Noga, Steve Seiden, and Gerhard J Woeginger. A faster off-line algorithm for the tcp acknowledgement problem. *Information Processing Letters*, 81(2):71–73, 2002.

[74] Anna R Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgement and other stories about $e/(e-1)$. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing (STOC)*, pages 502–509, 2001.

[75] Steven S Seiden. A guessing game and randomized online algorithms. In *Proceedings of the thirty-second annual ACM Symposium on Theory of Computing (STOC)*, pages 592–601, 2000.

[76] Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *European Symposium on Algorithms*, pages 253–264. Springer, 2007.

[77] Susanne Albers and Helge Bals. Dynamic TCP acknowledgment: Penalizing long delays. *SIAM Journal on Discrete Mathematics*, 19(4):938–951, 2005.

[78] Jens S Frederiksen, Kim S Larsen, John Noga, and Patchrawat Uthaisombut. Dynamic TCP acknowledgment in the LogP model. *Journal of Algorithms*, 48(2):407–428, 2003.

[79] Csanád Imreh and Tamás Németh. On time lookahead algorithms for the online data acknowledgement problem. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 288–297, 2007.

[80] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 322–333, 1988.

[81] A. Wei and F. Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 8042–8053, 2020.

[82] S. Lattanzi, T. Lavastida, B. Moseley, and S. Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1859–1877, 2020.

[83] Spyros Angelopoulos and Shahin Kamali. Contract scheduling with predictions. *Journal of Artificial Intelligence Research*, 77:395–426, 2023.

[84] Spyros Angelopoulos, Diogo Arsénio, and Shahin Kamali. Competitive sequencing with noisy advice. *CoRR*, abs/2111.05281, 2021.

[85] Y. Azar, S. Leonardi, and N. Touitou. Flow time scheduling with uncertain processing time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1070–1080, 2021.

[86] R. Lee, J. Maghakian, M. Hajiesmaili, J. Li, R. K. Sitaraman, and Z. Liu. Online peak-aware energy scheduling with untrusted advice. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems (e-Energy)*, pages 107–123, 2021.

[87] E. Bampis, K. Dogeas, A. V. Kononov, G. Lucarelli, and F. Pascual. Scheduling with untrusted predictions. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4581–4587, 2022.

[88] E. Balkanski, V. Gkatzelis, and X. Tan. Strategyproof scheduling with predictions. In *14th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 251, pages 11:1–11:22, 2023.

[89] Joan Boyar, Lene M Favrholdt, Shahin Kamali, and Kim S Larsen. Online interval scheduling with predictions. In *Algorithms and Data Structures Symposium (WADS)*, pages 193–207, 2023.

[90] A. Zeynali, B. Sun, M. Hajiesmaili, and A. Wierman. Data-driven competitive algorithms for online knapsack and set cover. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 10833–10841, 2021.

[91] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Online knapsack with frequency predictions. *Advances in neural information processing systems (NeurIPS)*, 34:2733–2743, 2021.

[92] Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. Online unit profit knapsack with predictions. *Algorithmica*, 86(9):2786–2821, 2024.

[93] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM*, 68(4):Article No. 24, 2021.

[94] Alexander Wei. Better and simpler learning-augmented online caching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 176, pages 60:1–60:17, 2020.

[95] A. Antoniadis, J. Boyar, M. Eliás, L. M. Favrholdt, R. Hoeksma, K. S. Larsen, A. Polak, and B. Simon. Paging with succinct predictions. In *40th International Conference on Machine Learning (ICML)*, volume 202, pages 952–968, 2023.

[96] Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. *Discrete Optimization*, 48(Part 2):100778, 2023.

[97] T. Lavastida, B. Moseley, R. Ravi, and C. Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. In *29th Annual European Symposium on Algorithms (ESA)*, volume 204, pages 59:1–59:17, 2021.

[98] T. Lavastida, B. Moseley, R. Ravi, and C. Xu. Using predicted weights for ad delivery. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA)*, pages 21–31, 2021.

[99] S. Angelopoulos, S. Kamali, and D. Zhang. Online search with best-price and query-based predictions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9652–9660, 2023.

[100] J. Y. Chen, S. Silwal, A. Vakilian, and F. Zhang. Faster fundamental graph algorithms via learned predictions. In *International Conference on Machine Learning (ICML)*, volume 162, pages 3583–3602, 2022.

[101] F. Eberle, A. Lindermayr, N. Megow, L. Nölke, and J. Schlöter. Robustification of online graph exploration methods. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9732–9740, 2022.

[102] J. Y. Chen, T. Eden, P. Indyk, H. Lin, S. Narayanan, R. Rubinfeld, S. Silwal, T. Wagner, D. P. Woodruff, and M. Zhang. Triangle and four cycle counting with predictions in graph streams. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[103] Y. Azar, D. Panigrahi, and N. Touitou. Online graph algorithms with predictions. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 35–66, 2022.

[104] S. Banerjee, V. Cohen-Addad, A., and Z. Li. Graph searching with predictions. In *14th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 251, pages 12:1–12:24, 2023.

[105] M. Mitzenmacher and S. Vassilvitskii. Algorithms with predictions. In *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020.

[106] Algorithms with predictions. `https://algorithms-with-predictions.github.io/`. Accessed: 2025-06-25.

[107] Csanád Imreh and Tamás Németh. Parameter learning algorithm for the online data acknowledgment problem. *Optimization Methods and Software*, 26(3):397–404, 2011.

[108] Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:20083–20094, 2020.

[109] Sungjin Im, Benjamin Moseley, Chenyang Xu, and Ruilong Zhang. Online dynamic acknowledgement with learned predictions. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1–10, 2023.

[110] Shom Banerjee. Improving online rent-or-buy algorithms with sequential decision making and ml predictions. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:21072–21080, 2020.

[111] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[112] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17: 449–467, 1965.

[113] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. Traffic data repository at the wide project. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, pages 263–270, 2000.

[114] George Zames. Feedback and optimal sensitivity: Model reference transformations, multiplicative seminorms, and approximate inverses. *IEEE Transactions on Automatic Control*, 26(2):301–320, 2003.

[115] Lamia Karray, Pierre Duhamel, and Olivier Rioul. Image coding with an L^∞ and confidence interval criteria. *IEEE Transactions on Image Processing*, 7(5):621–631, 1998.

[116] Vern Paxson and Sally Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[117] Susan F Assmann, David S. Johnson, Daniel J. Kleitman, and JY-T Leung. On a dual version of the one-dimensional bin packing problem. *Journal of Algorithms*, 5(4): 502–525, 1984.