Alternative Approaches for Analysis of Bin Packing and List Update Problems

by

Shahin Kamali

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis we introduce and evaluate new algorithms and models for the analysis of online bin packing and list update problems. These are two classic online problems which are extensively studied in the literature and have many applications in the real world. Similar to other online problems, the framework of competitive analysis is often used to study the list update and bin packing algorithms. Under this framework, the behavior of online algorithms is compared to an optimal offline algorithm on the worst possible input. This is aligned with the traditional algorithm theory built around the concept of worst-case analysis. However, the pessimistic nature of the competitive analysis along with unrealistic assumptions behind the proposed models for the problems often result in situations where the existing theory is not quite useful in practice. The main goal of this thesis is to develop new approaches for studying online problems, and in particular bin packing and list update, to guide development of practical algorithms performing quite well on real-world inputs. In doing so, we introduce new algorithms with good performance (not only under the competitive analysis) as well as new models which are more realistic for certain applications of the studied problems.

For many online problems, competitive analysis fails to provide a theoretical justification for observations made in practice. This is partially because, as a worst-case analysis method, competitive analysis does not necessarily reflect the typical behavior of algorithms. In the case of bin packing problem, the Best Fit and First Fit algorithms are widely used in practice. There are, however, other algorithms with better competitive ratios which are rarely used in practice since they perform poorly on average. We show that it is possible to optimize for both cases. In doing so, we introduce online bin packing algorithms which outperform Best Fit and First Fit in terms of competitive ratio while maintaining their good average-case performance.

An alternative for analysis of online problems is the advice model which has received significant attention in the past few years. Under the advice model, an online algorithm receives a number of bits of advice about the unrevealed parts of the sequence. Generally, there is a trade-off between the size of the advice and the performance of online algorithms. The advice model generalizes the existing frameworks in which an online algorithm has partial knowledge about the input sequence, e.g., the access graph model for the paging problem. We study list update and bin packing problems under the advice model and answer several relevant questions about the advice complexity of these problems.

Online problems are usually studied under specific settings which are not necessarily valid for all applications of the problem. As an example, online bin packing algorithms are widely used for server consolidation to minimize the number of active servers in a data center. In some applications, e.g., tenant placement in the Cloud, often a 'fault-tolerant' solution for server consolidation is required. In this setting, the problem becomes different and the classic algorithms can no longer be used. We study a fault-tolerant model for the bin packing problem and analyse algorithms which fit this particular application of the problem. Similarly, the list update problem was initially proposed for maintaining self-adjusting linked lists. However, presently, the main application of the problem is in the data compression realm. We show that the standard cost model is not suitable for compression purposes and study a compression cost model for the list update problem. Our analysis justifies the advantage of the compression schemes which are based on Move-To-Front algorithm and might lead to improved compression algorithms.

Acknowledgements

First and foremost, I would like to thank to my Ph.D. advisor, Professor Alejandro (Alex) López-Ortiz. Alex is someone you will never forget once you meet him. He is the funniest advisor and one of the nicest people I know. I have learned a great deal from him, both professionally and personally, and feel very fortunate to have worked with him. Thank you Alex for making my Ph.D. an exciting and joyful experience.

I would like to thank my committee members, Professor Jonathan Buss, Professor Jochen Könemann, and Professor J. Ian Munro, for taking the time to read and critique my thesis and for their constructive comments. I thank Professor David S. Johnson for being the external examiner of my Ph.D. committee. His valuable comments have provided a fresh perspective for improving and continuing this work.

I am grateful to Professor Joan Boyar and Professor Kim S. Larsen, our collaborators in University of Southern Denmark, for their remarkable insights on research problems as well as their hospitality during my visit to Denmark in Spring 2012. I thank Professor Khuzaima Daudjee for guiding me in finding interesting problems with practical significance.

I thank all of my friends in Waterloo for the good times. The Algorithm and Complexity research lab has been a perfect working environment for me. Special thanks goes to my good friends and officemates Francisco Claude, Reza Dorrigiv, Arash Farzan, Bob Fraser, Susana Ladra, Daniela Maftuleac, Patrick Nicholson, Alejandro Salinger, Diego Seco, and Shoeleh Shams.

I would also like to thank Helen Jardine and Wendy Rush for offering their help whenever I needed it.

I gratefully acknowledge the funding sources that made my Ph.D. work possible. I am thankful to the Natural Sciences and Engineering Research Council of Canada (NSERC) for providing me with funding for three years of my Ph.D. and facilitating my research trip to Denmark, to the Ontario Ministry of Training, Colleges and Universities for awarding me the OGS Scholarship, to the family of Dr. Derick Wood for their generous support with the Derick Wood Graduate Scholarship, and to the University of Waterloo for supporting me for four years of my Ph.D. with the President Scholarship and Thesis Completion Award.

Lastly, I would like to thank my family for all their love and encouragement. To my parents for all the support they have provided me over the years. It is the greatest gift anyone has ever given me. And to the presence of my brother Shahab and his wife Erfaneh for most of my years in Waterloo. Thank you for your love and support.

Dedication

To my parents

Table of Contents

List of Tables

List of Figures

List of Algorithms

Part I

Introduction

Chapter 1

Online Algorithms

In online algorithms, in contrast to offline algorithms, the input is not revealed all at once. Instead, it is formed as a sequence of requests which appear in a sequential manner. To serve each request, an online algorithm has to make a decision based on the revealed parts of the sequence. The decisions of the algorithm are irrevocable, i.e., the algorithm cannot revert or change its actions. In other words, an online algorithm has to build a solution for a sequence σ on the partial solutions that it builds for the prefix subsequences of σ.

When studying online algorithms, similar to other algorithms, we are interested in providing worst-case guarantees. The performance of an online algorithm is often compared against an optimal offline algorithm Opt. This kind of worst-case analysis has been studied in the context of online scheduling and bin packing problems since the 1970s. However, it became more popular when it was reintroduced in the 1980s in the form of *competitive analysis* for analysis of paging and list update algorithms. Since then, the competitive ratio has served as a practical measure for the study and classification of online algorithms. An algorithm is said to be c-competitive (assuming a cost-minimization problem) if the cost of the algorithm for any request sequence never exceeds c times the optimal cost (up to some additive constant) of an offline algorithm Opt which knows the entire request sequence in advance. If we do not allow the additive term, the resulting ratio is called the *absolute* competitive ratio. In this thesis, by competitive ratio, we always mean *asymptotic* competitive ratio as defined above.

Although the main initiative for studying most online algorithms is their practicality in the real world, some assumptions behind the models and analysis techniques make the current results not quite useful for all applications. Some of these issues can be listed as follows:

- In most applications, beside providing worst-case guarantees, an online algorithm needs to have a good typical behavior (average-case performance). The focus on improving competitive ratio often results in sacrificing the average-case performance. Consequently, many of the studied online algorithms have no practical significance. This is particularly the case for the bin packing problem. In Chapter 3, we discuss this issue and address it by introducing practical algorithms which provide both average and worst-case guarantees.
- Notwithstanding its wide applicability, competitive analysis has some drawbacks. For certain problems, it gives unrealistically pessimistic results and fails to distinguish between algorithms that have

vastly differing performance in practice. Another drawback is the unfair comparison between online and offline algorithms which gives unjust advantage to OPT. One natural way to address this issue is to give online algorithms the power of deferral, i.e., some limited power in changing their previous decisions (see, e.g., [77, 23, 83]). Another approach is to give online algorithms partial information about the future. A well-studied example is the concept of *lookahead* in which some future requests are revealed to the online algorithm. In the past few years, a more general framework is studied which gives partial information about the future on an *advice tape*. The advice bits are generated by an offline oracle. There is a trade-off between the number of bits of advice and the competitive ratio of the resulting algorithms. In Chapters 4,5,6, we discuss this trade-off in details for bin packing and list update problems.

- The models defined for online problems are not realistic for all applications of the problems. For example, the standard model for the list update problem is defined in a way to suit self-adjusting lists. Later, a novel application of the problem was discovered for data compression; however, the standard model was never adjusted for this application. Another example is when bin packing is used for server consolidation; while this is a well known application of the problem, if a fault-tolerant scheme for server consolidation is required (which is the case in modern applications), the standard model and algorithms will not be valid and should be adjusted. In Chapters 7,8, we introduce practical models for bin packing and list update which suit the above applications.

Bin packing and list update problems are among the most studied online problems with many applications in practice (see, e.g., [49, 9]). In the bin packing problem, the goal is to place a sequence of *items* of different sizes into a minimum number of *bins*. We assume items have sizes in the range $(0, 1]$ and bins have a uniform capacity of 1. Examples of bin packing algorithms are BEST FIT and FIRST FIT which avoid opening new bins for an incoming item (unless they have to) and the HARMONIC algorithm which is based on placing items of similar sizes into the same bins. In the list update problem, the input is a sequence of *requests* to items of a list of fixed size. To answer a request to an item x, an algorithm has to *access* x. Accessing an item at index i in the list has a cost of i. A list update algorithm can reorganize the list using free and paid *exchanges*. The goal is to update (reorganize) the list so that the total cost is minimized. Example of list update algorithms are Move-To-Front (MTF) which moves the requested item to the front of the list using a free exchange and Frequency Count (FC) that maintains items in decreasing order of their observed frequency. (See Chapter 2 for a detailed definition of bin packing and list update problems.)

Throughout the thesis, we use $\mathbb{A}(\sigma)$ to denote the costs of $\mathbb{A}$ for packing a request sequence σ. When σ follows from the context, we simply use $\mathbb{A}$ to denote this cost. We use similar notation for all algorithms, including OPT.

1.1 Analysis Measures

Online algorithms were initially studied under stochastic models which assume a random distribution for input sequences. These measures are good for studying the typical behavior of the algorithms under random input assumptions. Unfortunately however, they do not provide any worst-case guarantee. Moreover, the distributions behind the online sequences are not always fixed and change over time. The competitive ratio provides worst-case guarantees for online algorithms. However, for many problems (e.g., bin packing) an

algorithm with a good worst-case performance does not necessarily perform well on the average. In practice, this discrepancy is addressed by sacrificing the worst-case performance, i.e., the algorithms with good average-case performance (e.g., Best Fit and First Fit) are preferred over the ones with better competitive ratio (e.g., the Harmonic algorithm). There have been efforts to introduce measures which capture worst-case and average-case complexity of algorithms at the same time. Among these measures, random order analysis, relative worst order analysis, and bijective analysis are most relevant.

Random order analysis was introduced by Kenyon [100] in the context of online bin packing. In the bin packing problem, the number of bins used by Opt is the same for all permutations of an input sequence while an online algorithm $\mathbb{A}$ might open different number of bins for different permutations. Instead of considering the number of bins used by $\mathbb{A}$ on a worst-case sequence, one can consider the expected number of bins used by $\mathbb{A}$ over all permutations of a sequence σ and compare it with the number of bins in an optimal packing of σ. The maximum ratio achieved this way (over all sequences) is defined as the random order ratio of $\mathbb{A}$. Since we take the expectation over all permutations, the random order ratio of an algorithm is no larger than its competitive ratio. For example, the random order ratio of Best Fit for bin packing is in the range (1.08,1.5) [100] while its competitive ratio is 1.7 [95]. The random order ratio captures both worst-case and average-case complexity (since we assume all permutations are equally likely, it is related to the uniform distribution for the input sequence). Unfortunately however, it is difficult to compute random order ratios for most bin packing algorithms.

The relative worst order ratio was introduced by Boyar and Favrhold [40, 66]. Under this measure, two algorithms are directly compared and the ratio between their costs on the worst permutations of a given sequence is considered. Note that, for a given sequence, two algorithms might find their maximum cost on different permutations of the sequence. If an algorithm $\mathbb{A}$ does better than algorithm $\mathbb{B}$ for some sequences and worse for other sequences, the two algorithms are incomparable. Otherwise, the maximum ratio over all sequences is taken as the relative worst order ratio. Unfortunately, in many cases, the algorithms are incomparable, e.g., Best Fit and Harmonic are not comparable. Moreover, this measure does not reflect the typical behavior of online algorithms, e.g., Best Fit has no advantage over First Fit or Harmonic (regarding the relative worst order ratio) [40] while it is known that it outperforms both on the average (see Section 2.1.1).

Angelopoulos et al. [16] introduced bijective analysis as another measure which directly compares online algorithms. Under bijective analysis, an algorithm $\mathbb{A}$ is no worse than $\mathbb{B}$, if any sequence σ can be bijected to another sequence σ' so that the cost of $\mathbb{A}$ for σ is no more than that of $\mathbb{B}$ for σ'. This type of analysis captures both worst-case and average case and has proven useful for showing the observed advantage of Move-To-Front algorithm for the list update problem [17, 18]. Bijective analysis requires the set of all input sequences to be countable. Hence, in its standard form, it cannot be applied to the bin packing problem. However, if we modify the definition to restrict σ' to be a permutation of σ, the measure can be adapted to compare bin packing algorithms. Unfortunately, with this adaptation, bin packing algorithms are hard to study and analyze.

Introducing algorithms which provide both average-case and worst-case guarantees is important in practice. List update algorithms were initially studied with respect to their typical behavior on sequences that follow probability distributions. The average cost ratio of an algorithm $\mathbb{A}$ is the expected ratio between the cost of $\mathbb{A}$ for a sufficiently long random sequence and that of a static offline algorithm which maintains items in non-increasing order by probability. Under this setting, the ratio achieved by Fc is 1 [125] while that of Mtf is $\pi/2$ [47]. This indicates that Fc is better than Mtf on average. However, Mtf has an advantage over the other algorithms in the worst case scenarios [134]. Blum et al. [34] introduced

a randomized algorithm that, for any $\epsilon > 0$, achieves competitive ratio of $1.6 + \epsilon$ and average cost ratio of $1 + \epsilon$. In some sense, the algorithm provides both worst-case and average-case guarantees of the best existing algorithms. In Part II of this thesis, we introduce a new algorithm that does the same for the bin packing problem.

1.2 Advice Model of Computation

The total lack of information about the future is unrealistic in many real-world scenarios. The advice model is introduced to address this issue [69]. Under the advice model, an online algorithm receives a number of bits of advice about the unrevealed parts of the sequence. Generally, there is a trade-off between the size of advice and the performance of online algorithms.

In the past few years, slightly different models of advice complexity have been proposed for online problems. All these models assume that there is an offline oracle, with infinite computational power, which provides the online algorithm with some bits of advice. How these bits of advice are given to the algorithm is the source of difference between the models. In the first model, presented by Dobrev et al. [63], an online algorithm poses a series of questions which are answered by the offline oracle in *blocks of answers*. The total size of the answers, measured in the number of bits, defines the advice complexity. The problem with this model is that information can be encoded in the individual length of each block. To address this issue, another model is proposed by Emek et al. [69] which assumes that online algorithms receive a fixed number of bits of advice per request. We call this model the *advice-with-request model*. This model is studied for problems, such as metrical task systems and k-server, and the results tend to use at least a constant number of bits of advice per request [69, 122]. Nevertheless, there are many online problems for which a sublinear and even a constant number of bits of advice in total is sufficient to achieve good competitive ratios. However, under the advice-with-request model, the possibility of sending a sublinear number of advice bits to the algorithm is not well defined. Böckenhauer et al. [37, 36] presented another model of advice complexity which assumes that the online algorithm has access to an *advice tape*, written by the offline oracle. At any time step, the algorithm may refer to the tape and read any number of advice bits. The advice complexity is the number of bits on the tape accessed by the algorithm. We refer to this model as *advice-on-tape model*. Since its introduction, the advice-on-tape model has been used to analyze the advice complexity of many online problems including paging [37, 90, 104], disjoint path allocation [37], job shop scheduling [37, 104], k-server [36, 122], knapsack [38], Steiner tree problem [26], various coloring problems [30, 74, 31, 130], set cover [103, 35], maximum clique [35], and graph exploration [62]. Throughout the rest of the thesis, by *advice model*, we mean advice-on-tape model.

To provide general lower bounds for the advice complexity of online problems, we use the Binary Guessing problem, defined in [69, 35].

Definition 1 ([35]). *The* Binary String Guessing Problem with Known History (2-SGKH) *is the following online problem. The input is a bitstring of length m, and the bits are revealed one by one. For each bit b_t, the online algorithm $\mathbb{A}$ must guess if it is a 0 or a 1. After the algorithm has made a guess, the value of b_t is revealed to the algorithm.*

In a few places in Part III of the thesis, we make use of the following lemma which implies that advice of linear size is required to correctly guess more than half bits of an input bitstring.

Lemma 1 ([35]). *On an input of length m, any deterministic algorithm for 2-SGKH that is guaranteed to guess correctly on more than αm bits, for $1/2 \leq \alpha < 1$, needs to read at least $(1+(1-\alpha)\lg(1-\alpha)+\alpha\lg\alpha)m$ bits of advice.*

Provided with the above lemma, we can reduce the 2-SGKH problem to other problems (e.g., bin packing) to show that a linear number of advice bits are required to achieve close-to-optimal solutions.

1.3 Summary of Results and Organization of the Thesis

In Chapter 2, we conclude the first part of the thesis by reviewing the existing results on the bin packing and list update problems. We do not aim to be exhaustive, but rather present the highlights which are related to other parts of the thesis.

Although many online algorithms are proposed for the bin packing problem, the classic Best Fit and First Fit algorithms are the mostly applied algorithms for the problem. This is because the existing algorithms which provide improvements on the competitive ratio have relatively bad average-case performance. A natural question is whether there exist an algorithm which achieves a better competitive ratio than Best Fit and First Fit without a compromise on the average-case performance. In part II of the thesis, we answer this question positively by introducing two new algorithms with the desired property. In our analysis, we assume a continuous uniform distribution where the probability of an item having size x is the same as having size $1-x$. Our interest in these algorithms is mostly theoretical; however, variants of these algorithms which adapt to different distributions might have practical significance.

In Part III of the thesis, we study the advice complexity of bin packing, square packing, and list update problems. When studying the advice model, there are a few questions to answer. First, how many bits of advice are required and sufficient to achieve an optimal solution? In other words, how much information about an input sequence is sufficient to achieve a solution which is as good as an offline solution? We answer this question for bin packing and list update problems by providing tight lower and upper bounds. Another important question is how many bits of advice are sufficient to outperform all online algorithms, i.e., to break the lower bound on the competitive ratio of all online algorithms? For the bin packing problem, we show that advice of logarithmic size is sufficient. For the box packing problem, advice of also logarithmic size is sufficient to outperform all existing online algorithms. For the list update problem, only two bits of advice are sufficient to outperform all deterministic online algorithms. The bits indicate the better of three list update algorithms for serving an input sequence. Another relevant question asks for the best competitive ratio that one can achieve with advice of linear size (linear to the length of input sequence). We partially answer this question for the bin packing problem by providing upper and lower bounds.

In Part IV, we study two applications of the bin packing and list update problems. Unlike previous parts, the results in this part are merely practical. In Chapter 7, we study the fault tolerant server consolidation as an application of the bin packing problem. In this application, instead of one copy of each item, two copies (replicas) should be placed in two different bins (servers) so that potential failure of one bin does not interrupt the service. Among other results, we provide a new algorithm for this problem which has a visible advantage over two existing heuristics. In Chapter 8, we study the list update problem in the context of compression and analyze list update algorithms under a theoretical framework which is more relevant to compression purposes. On the practical side, we introduce a new compression scheme

which is based on a list update algorithm. The list update algorithm makes use of some bits of advice which are included in the compressed file. Our experiments indicate that our scheme has an advantage over other schemes which make use of list update algorithms.

Part of the work presented in this thesis has already been appeared in the following papers.

- Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. On the list update problem with advice. In *Proc. 8th International Conf. on Language and Automata Theory and Applications (LATA)*, pages 210–221, 2014.
- Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. Online bin packing with advice. In *Proc. 31st Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 174–186, 2014.
- Khuzaima Daudjee, Shahin Kamali, and Alejandro López-Ortiz. On the Online fault-tolerant server consolidation problem. In *Proc. 26th Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 12–21, 2013.
- Shahin Kamali, Susana Ladra, Alejandro López-Ortiz, and Diego Seco. Context-based algorithms for the list-update problem under alternative cost models. In *Proc. 22nd Data Compression Conf. (DCC)*, pages 361–370, 2013.
- Shahin Kamali and Alejandro López-Ortiz. Better compression through better list update algorithms. In *Proc. 23rd Data Compression Conf. (DCC)*, pages 372–381, 2014.
- Shahin Kamali and Alejandro López-Ortiz. A survey of algorithms and models for list update. In *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of *Lecture Notes in Comput. Sci.*, pages 251–266, 2013.
- Shahin Kamali and Alejandro López-Ortiz. An all-around near-optimal solution for the classic bin packing problem. CoRR abs/1404.4526, 2014.
- Shahin Kamali and Alejandro López-Ortiz. Almost online square packing. In *Proc. 26th Canadian Conference on Computational Geometry (CCCG)*, 2014. to appear.

Chapter 2

A Tale of Two Problems

Bin Packing and list update problems are among the most studied online problems which have contributed a lot to the development of online algorithms. In this chapter we briefly review the results related to these problems.

2.1 Bin Packing Problem

In the bin packing problem, the goal is to place items of rational sizes into a minimum number of bins of uniform sizes. The bin packing problem is among the first online problems around which the concept of worst-case analysis was developed. The first upper bounds for the competitive ratio of online bin packing algorithms was introduced by Ullman [139] and was followed by the influential works of Johnson, Garey, Graham, and Ullman in the 1970s [78, 93, 95]. The first general lower bound for competitiveness of online algorithms was introduce by Yao in 1980 [142]. The stochastic analysis of online bin packing algorithms has an even older history. In this section, we review the algorithms and results related to the worst case and average case analysis of bin packing algorithms.

Definition 2. *An instance of the classic bin packing problem is defined by a sequence* $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ *of* items. *We use* $s(\sigma_i)$ *to denote the* size *of an item* σ_i*, which is a value in the range* $(0, 1]$. *The goal is to pack these items in the minimum number of bins of capacity* 1. *In the online version of the problem, the items are revealed in an online manner, and an algorithm should place an item into a bin without looking at the future items. After placing an item, the algorithm cannot move the item to another bin.*

A natural algorithm for the problem is Next Fit (Nf) which keeps one *open* bin at each time. If a given item does not fit in the bin, the algorithm *closes* the bin (i.e., it does not refer to it in future) and opens a new bin. In contrast to Nf, First Fit (Ff) does not close any bin; it maintains the bins in the order they are opened and places a given item in the first bin which has enough space for it. In case such a bin does not exist, it opens a new bin for the item. Best Fit (Bf) performs similarly to Ff, except that it maintains the bins in decreasing order of their *levels*; the level of a bin is the total size of items placed in the bin. Ff and Bf are members of the family of Any Fit algorithms. An algorithm is Any Fit if opens

a new bin for an item only if the item does not fit in any of the existing bins. A subfamily of Any Fit algorithms are Almost Any Fit algorithms. An algorithm is Almost Any Fit if it is Any Fit, and it avoids placing an item in the bin with the lowest level (unless there is no other bin with enough space for the item). An alternative approach is to divide items into a constant number of classes based on their sizes and pack items of the same class apart from other classes. An example is the Harmonic (Ha) algorithm of Lee and Lee [106] which has a parameter K and defines K intervals $(1/2, 1], (1/3, 1/2], \ldots, (1/(K-1), 1/K]$, and $(0, 1/K]$; items which belong to the same interval are separately treated using the Next Fit strategy.

2.1.1 Average-Case Analysis

Online bin packing algorithms are usually compared through their average-case performance and worst-case performance. Under average-case analysis, it is assumed that item sizes follow a fixed distribution that is typically the uniform distribution over the interval $[0, C]$, where C is the bin capacity, typically assumed without loss of generality to be 1. The results described below are for this distribution. With this assumption, one can define the *asymptotic average-case performance ratio* (or simply *average ratio*) of an online algorithm as the expected ratio between the number of bins used by the algorithm and that of Opt for placing a long randomly-generated sequence. Recall that Opt is an optimal offline algorithm. More precisely, the average ratio of an online algorithm $\mathbb{A}$ is $\lim_{n\to\infty} E\left[\frac{A(\sigma_{(n)})}{\text{Opt}(\sigma_{(n)})}\right]$, where $\sigma_{(n)}$ is a randomly generated sequence of length n.

It is known that Nf has an asymptotic average ratio of $1.\bar{3}$[1] [50] for sequences generated uniformly at random while Ff and Bf are optimal in this sense and have an average ratio of 1 [28]. To further compare algorithms with average ratio of 1, one can consider the *expected waste* which is the expected amount of wasted space for packing a sequence of length n. More precisely, the wasted space of an algorithm $\mathbb{A}$ for packing a sequence $\sigma_{(n)}$ of length n is $E[A(\sigma_{(n)}) - s(\sigma_{(n)})]$, i.e., the difference between the number of bins used by $\mathbb{A}$ for placing $\sigma_{(n)}$ and the total size of items in $\sigma_{(n)}$, denoted by $s(\sigma_{(n)})$. Ff and Bf have expected waste of sizes $\Theta(n^{2/3})$ and $\Theta(\sqrt{n}\lg^{3/4} n)$, respectively [132, 107]. It is also known that all online algorithms have expected waste of size $\Omega(\sqrt{n}\lg^{1/2} n)$ [132]. These results show that Bf is almost optimal with respect to average performance.

There are other algorithms which perform almost as well as Bf on average. These algorithms are based on matching a 'large' item with a 'small' item to place them in the same bin. We call an item large if it is larger than $1/2$ and small otherwise. Interval First Fit (Iff) [55] and Online Match (Om) [52] are among the matching-based algorithms. Iff has a parameter K and divides the unit interval into K intervals of equal length, namely $I_t = (\frac{t-1}{K}, \frac{t}{K}]$ for $t = 1, 2, \ldots, K$. Here, K is an odd integer and we have $K = 2j + 1$. The algorithms defines $j + 1$ classes so that intervals I_c and I_{K-c} form class c $(1 \le c \le j)$ and interval I_K forms class $j + 1$. Items in each class are packed separately from other classes. The items in class c $(2 \le c \le j + 1)$ are treated using the Ff strategy, while items in the first class are treated using an Almost Ff strategy. Almost Ff is similar to Ff except that it closes a bin when it includes a small and a large item; further, a large item is never placed in a bin which includes more than one small items, and a bin with K small items is declared as being closed. The average ratio of Iff is 1; precisely, it has an expected waste of $\Theta(n^{2/3})$. Algorithm Om also has a parameter K and declares two items as being *companions* if their sum is in the range $[1 - \frac{1}{K}, 1]$. To place a large item, Om opens a new bin. To place a small item x, the algorithm checks whether there is an open bin β with a large companion of x; in case

[1] By $1.\bar{3}$ we mean $1.33333\ldots$. Similar notation is used throughout the thesis.

there is, OM places x in β and closes β. Otherwise, it packs x using the NF strategy in a separate list of bins. The average ratio of OM converges to 1 for large values of K [52].

2.1.2 Worst-Case Analysis

Although the matching-based algorithms have good average performance, they do not perform well in the worst-case. In particular, IFF has an unbounded competitive ratio [55], and the competitive ratio of OM is 2 [52]. Among other matching algorithms we might mention MATCHING BEST FIT (MBF) which performs similarly to BF except that it closes a bin as soon as it receives the first small item. The average ratio of MBF is as good as BF while it has unbounded competitive ratio [132]. There is another online algorithm with expected waste of size $\Theta(\sqrt{n}\lg^{1/2} n)$ [133] which matches the lower bound of [132][2]. This algorithm also has a non-constant competitive ratio [49].

Recall that the competitive ratio reflects the worst-case performance of online algorithms and is defined as the asymptotically maximum ratio between the number of bins used by an online algorithm and that of OPT for packing the same sequence[3]. More precisely, the (asymptotic) competitive ratio of an online algorithm $\mathbb{A}$ is defined as $inf\{r \geq 1 : \text{for some } N > 0, A(\sigma)/\,\text{OPT}(\sigma) \leq r \text{ for all } \sigma \text{ with } \text{OPT}(\sigma) \geq N\}$ [49].

It is known that NF has a competitive ratio of 2 while FF and BF have the same ratio of 1.7 [93]. More generally, every Almost Any Fit algorithms has a competitive ratio of 1.7 [93, 94] (see Section 2.1.2 for definition of Almost Any Fit algorithms). The competitive ratio of HA converges to 1.691 for sufficiently large values of K [106]. To be more precise, it approaches $T_\infty = \sum_1^\infty \frac{1}{t_i - 1}$, where $t_1 = 2$ and $t_{i+1} = t_i(t_i - 1) + 1, i \geq 1$[4]. There are online algorithms which have even better competitive ratios. These include REFINED FIRST FIT (RFF) with ratio 1.666 [142], REFINED HARMONIC (RH) with ratio 1.636 [106], MODIFIED HARMONIC (MH) with ratio around 1.616 [118], and HARMONIC++ with ratio 1.588 [131]. These algorithms are members of a general framework of Super Harmonic algorithms [131]. Similar to HA, Super Harmonic algorithms classify items by their sizes and pack items of the same class together. However, to handle the bad sequences of HA, a fraction of opened bins include items from different classes. These bins are opened with items of small sizes in the hopes of subsequently adding items of larger sizes. At the time of opening such a bin, it is pre-determined how many items from each class should be placed in the bin. As the algorithms runs, the reserved spot for each class is occupied by an item of that class. It is guaranteed that the reserved spot is enough for any member of the class. This implies that the expected total size of items in the bin is strictly less than 1 by a positive value. Consequently, the expected waste of the algorithm is linear to the number of opened bins. Hence, for a sequence of length n, these algorithm have an expected waste of $\Omega(n)$. Since the expected wasted space of OPT is $o(n)$, the average performance ratio of Super Harmonic algorithms is strictly larger than 1. In particular, REFINED HARMONIC and MODIFIED HARMONIC have average performance ratios around 1.28 and 1.18, respectively [84, 117].

Regarding the lower bound, Yao showed that no online algorithm has a competitive ratio better than 1.5 [142]. This lower bound was subsequently improved. The best existing lower bound is presented by

[2]For the *closed* bin packing problem in which the length of the input sequence is known, there is an algorithm that achieves an optimal expected waste of size $\sqrt{n}$ [20].

[3]As mentioned previously, by competitive ratio, we mean asymptotic competitive ratio. For the results related to the absolute competitive ratio of bin packing algorithms, we refer the reader to [49, 54].

[4]Some notations are borrowed from [49].

Algorithm	Average Ratio	Expected waste	Competitive Ratio
Next Fit (Nf)	$1.\bar{3}$ [50]	$\Omega(n)$	2
Best Fit (Bf)	1 [28]	$\Theta(\sqrt{n}\lg^{3/4} n)$ [132, 107]	1.7 [93]
First Fit (Ff)	1 [107]	$\Theta(n^{2/3})$ [132, 51]	1.7 [93]
Harmonic (Ha)	1.2899 [106]	$\Omega(n)$	$\to T_\infty \approx 1.691$ [106]
Refined First Fit (Rff)	> 1	$\Omega(n)$	$1.\bar{6}$ [142]
Refined Harmonic (Rh)	1.2824 [84]	$\Omega(n)$	1.636 [106, 84]
Modified Harmonic (Mh)	1.189 [117]	$\Omega(n)$	1.615[118]
Harmonic++	> 1	$\Omega(n)$	1.588 [131]
Opt	1	$\Theta(\sqrt{n})$ [101, 110]	N/A

Table 2.1: Average performance ratio, expected waste (under continuous uniform distribution), and competitive ratios for different bin packing algorithms

Balogh et al. [22] and implies that no online algorithm can have a competitive ratio better than 1.54037. Table 2.1 shows the existing results for major bin packing algorithms.

2.1.3 Offline Bin Packing

The bin packing problem is known to be NP-hard (see, e.g., [79]). The hardness proof which is based on a simple reduction from the Partition problem also shows that the *absolute approximation ratio* of any polynomial time algorithm is no less than 3/2 (assuming $P \neq NP$). The absolute approximation ratio of an algorithm $\mathbb{A}$ is the maximum ratio between the number of bins opened by $\mathbb{A}$ and that of Opt for any input, while the *asymptotic approximation ratio* is the maximum value of the same ratio restricted to inputs for which Opt opens a sufficiently large number of bins. In what follows, by approximation ratio, we mean asymptotic approximation ratio.

There are many offline algorithms which are based on sorting items in decreasing order of item sizes. In particular, First Fit Decreasing (Ffd) and Best Fit Decreasing (Bfd) respectively apply First Fit and Best Fit strategies after sorting the sequence. These algorithms both have expected waste of size $\Theta(\sqrt{n})$ [110, 101] and an approximation ratio of 11/9 [93]. More generally, if an Any Fit algorithms is applied after sorting, the approximation ratio of the resulting algorithm is at least 11/9. A variant of Ffd, called Modified Ffd, has an improved approximation ratio of 71/60 while preserving the good average case behavior [96]. There are other approaches which are based on encoding the problem into an Integer Programming formulation and solving the relaxed Linear Programming formula. For a long time, the best approximation algorithm was that of Karmarkar and Karp [98] which opened $\text{Opt} + O(\lg^2 \text{Opt})$ bins for a sequence that can packed optimally in Opt bins. This result was finally improved by Rothvoß with an algorithm which opens $\text{Opt} + O(\lg \text{Opt} \times \lg\lg \text{Opt})$ bins [126]. On the other hand, it is not known whether the problem of packing a sequence into $\text{Opt}+1$ bins is NP-hard or not. In particular, it is not known whether the additive gap between the optimum Linear Program solution and the optimum integral solution is more than 1. This relates the problem to the Modified Integer Round-up Conjecture [128]. It should be mentioned that the approximation algorithms which are based on Integer Programming formulation of the problem are not useful in practice because of their complicated nature and slow running

time. For example, the algorithm of Karmarkar and Karp has a running time of $O(n^8)$ which makes it too slow for practical scenarios.

2.1.4 Two Dimensional Bin Packing

There are many extensions of the 1-dimensional bin packing problem as defined above. The simplest extension might be the *square packing* problem in which, instead of items to be placed into one-dimensional bins, the input is a set of squares which need to be packed in squares of unit sizes. This variant of the problem can be further generalized to the *box packing* problem in which the input is a sequence of boxes (rectangles) to be placed into unit squares. Both square packing and box packing problems can be further generalized to d dimensions.

Square packing and box packing problems are studied under both offline and online settings. In the offline setting, all squares (boxes) are available in advance. In the online setting, the squares (boxes) appear in an online and sequential manner. Similar to the bin packing problem, the decisions of the online algorithms are irrevocable, i.e., it is not possible to move a square (box) after it is placed into a bin.

The offline version of the square packing problem is NP-hard [108]. There were numerous efforts to introduce algorithms with good approximation ratios (see, e.g., [102, 71]) until Bansal et al. introduced an APTAS for the problem [24], i.e., they introduced an algorithm whose output never exceeds $(1+\epsilon)\,\textsc{Opt}(\sigma)+1$ bins for an input σ and a given $\epsilon > 0$. The running time of this algorithm is $O(n \lg(n))$ but it depends exponentially on ϵ. In [70] a *robust* APTAS is provided in which the items are packed one by one, and the total volume of items which may migrate between bins, or change their positions inside bins, is bounded by a constant factor of the volume of the new item. The running time of this robust APTAS is $\Theta(n^2 \log n)$. For the online setting, the best existing algorithm has a competitive ratio of 2.1187 [88]. It is also known that any online algorithm has a competitive ratio of at least 1.6406.[72]. Note that there is a big gap between the best upper and lower bounds. It should be mentioned that most existing results extend to the *cube packing* problem which is the generalization of the problem into d dimensions $d \geq 2$.

The box packing problem is much harder than square packing. In the offline setting, in contrast to square packing, there is no APTAS for the problem unless P=NP [24]. The best existing approximation algorithm was recently introduced by Bansal and Khan [25] and has an approximation ratio of 1.4055. It is also known that no algorithm can have an approximation ratio better than $1 + 1/2196$ [45]. The online setting for the box packing problem is also well-studied (see, e.g.,[87, 75, 76, 140, 33, 33]). The best existing online box packing algorithm has a competitive ratio of 2.5545 [87], while there is a lower bound of 1.907 for the competitive ratio of any online box packing algorithm [33].

2.2 List Update Problem

List update is a fundamental problem in the context of online computation. The problem was first studied by McCabe [113] more than 45 years ago under distributional analysis in the context of maintaining a sequential file. In 1985, Sleator and Tarjan [134] introduced the framework of competitive analysis for the study of the worst-case behavior of list update algorithms. Since then, many deterministic and randomized online algorithms have been proposed and studied under this framework.

Definition 3. *[9] Consider an unsorted list of l items. An instance of the list update problem is a sequence of n requests that must be served in an online manner. To serve a request to an item x, an online algorithm $\mathbb{A}$ linearly searches the list until it finds x. If x is the i'th item in the list, $\mathbb{A}$ incurs a cost i to access x. Immediately after this access, $\mathbb{A}$ can move x to any position closer to the front of the list at no extra cost; this is called a* free exchange. *Also, $\mathbb{A}$ can exchange any two consecutive items at a cost of 1; these are called* paid exchanges. *An efficient algorithm can thus use free and paid exchanges to minimize the overall cost of serving a sequence. This model is called the* standard cost model.

Three well-known deterministic online algorithms are Move-To-Front (MTF), TRANSPOSE, and Frequency Count (FC). MTF moves the requested item to the front of the list; whereas TRANSPOSE exchanges the requested item with the item that immediately precedes it. FC maintains an access count for each item ensuring that the list always contains items in non-increasing order of frequency count. TIMESTAMP is an efficient list update algorithm introduced by Albers [2]. After accessing an item x, TIMESTAMP inserts x in front of the first item y that is before x in the list and was requested at most once since the last request for x. If there is no such item y, or if this is the first access to x, TIMESTAMP does not reorganize the list.

As mentioned earlier, list update algorithms were initially studied with respect to their average-case behavior on random sequences. Recall that the average cost ratio of an algorithm $\mathbb{A}$ is the expected ratio between the cost of $\mathbb{A}$ and that of an optimal static ordering for serving randomly generated sequences of large lengths. Under this setting, the ratio achieved by FC is 1 [125] while that of MTF is $\pi/2$ [46]. Moreover, there are distributions in which TRANSPOSE has a better ratio than MTF [125]. These results indicate that FC and TRANSPOSE are no worse than MTF. However, in practice, MTF has an advantage over the other algorithms. This is partially because the input sequences do not necessarily follow a fixed probability distribution.

List update algorithms were among the first algorithms studied using competitive analysis. Sleator and Tarjan [134] showed that MTF is 2-competitive, while TRANSPOSE and FC do not have constant competitive ratios. Karp and Raghavan proved a lower bound of $2 - 2/(l+1)$ (reported in [91]), and Irani [91] proved that MTF gives a matching upper bound. It is known that TIMESTAMP is also a best possible online algorithm, with respect to competitive ratios.

Besides MTF and TIMESTAMP, El-Yaniv showed that there are many other algorithms which have optimal competitive ratios [67]. In doing so, he introduced a family of algorithms called MOVE-TO-RECENT-ITEM (MRI). A member of this family has an integer parameter $k \geq 1$, and inserts an accessed item x just after the last item y in the list which precedes x and is requested at least $k+1$ times since the last request to x. If such item y does not exist, or if this is the first access to x, the algorithm moves x to front of the list. It is known that any member of MRI family of algorithms is 2-competitive [67]. Schulz proposed another family of algorithm called SORT-BY-RANK (SBR) [129], which is parametrized by a real value α where $0 \leq \alpha \leq 1$. The extreme values of α result in MTF (when $\alpha = 0$) and TIMESTAMP (when $\alpha = 1$). Members of SBR family are also 2-competitive [129].

Classic list update algorithms have also been studied under relative worst order analysis [66]. It is known that MTF, TIMESTAMP, and FC perform identically according to the relative worst order ratio, while TRANSPOSE is worse than all these algorithms.

In terms of the optimal offline algorithm for the list update problem, Manasse et al. [111] presented an offline optimal algorithm which computes the optimal list ordering at any step in time $\Theta(n \times (l!)^2)$, where n is the length of the request sequence and l is the size of the list. This time complexity was improved

Algorithm	Competitive Ratio	deterministic	Projective	Economical
MTF	2 [134, 91]	✓	✓	✓
TRANSPOSE	non-constant [134]	✓	×	✓
FREQUENCY COUNT	non-constant [134]	✓	✓	✓
RANDOM-MTF (RMTF)	≥ 2 [39]	×	✓	✓
MOVE-FRACTION (MF_k)	$2k$ [134]	✓	×	✓
TIMESTAMP	2 [2]	✓	✓	✓
MRI family	2 [67]	✓	✓	✓
SBR family	2 [129]	✓	✓	✓
TIMESTAMP family (randomized)	1.618 [2]	×	✓	✓
SPLIT	1.875 [91, 92]	×	×	✓
BIT	1.75 [121]	×	✓	✓
MTF2 (deterministic)	2.5 (Section 6.4)	✓	✓	✓
Random Reset (RST)	1.732 [121]	×	✓	✓
COMB	1.60 [8]	×	✓	✓

Table 2.2: A review of online strategies for list update problem.

to $\Theta(n \times 2^l(l-1)!)$ by Reingold and Westbrook [120]. Hagerup proposed another offline algorithm with running time $O(2^l\ l!\ f(l)+l\times n)$, where $f(l) \leq l!\ 3^{l!}$ [86]. Note that the time complexity of these algorithms is incomparable to one another. Another algorithm by Pietzrak is reported to run in time $\Theta(n\ l^3\ l!)$ [114]. It should be mentioned that Ambühl claims that the offline list update problem is NP-hard for non-constant values of l [12], although a full version of the proof remains to be published.

All the main existing online algorithms for the list update problem are *economical*, i.e., they only use free exchanges (look at Table 2.2). In fact, there is only one known non-trivial class of algorithms that uses paid exchanges [81]. While it is still not clear how an online algorithm can benefit from paid exchanges, Reingold and Westbrook showed that there are optimal offline algorithms which only make use of paid exchanges [120].

2.2.1 Randomization

As mentioned earlier, any deterministic list update algorithm has a competitive ratio of at least $2-2/(l+1)$. In order to go past this lower bound, a few randomized algorithms have been proposed. Randomized online algorithms are usually compared against an *oblivious* adversary which has no knowledge of the random bits used by the algorithm. To be more precise, an oblivious adversary generates a request sequence before the online algorithm starts serving it, and in doing so, it does not look at the random bits used by the algorithm. Two stronger types of adversaries are *adaptive online* and *adaptive offline* adversaries. An adaptive online adversary generates the tth requests of an input sequence by looking at the actions of the algorithm for serving the last $t-1$ requests. An adaptive offline adversary is even more powerful and knows the random bits used by the algorithm, i.e., before giving the input sequence to the online algorithm, it can observe how the algorithm serves the sequence. The definition of the competitive ratio of an online algorithm is slightly different when compared with different adversaries, and is based on the expectations

over the random choices made by the online algorithm (and adaptive adversaries). For a precise definition of competitiveness for randomized algorithms, we refer the reader to [121].

Ben-David et al. [27] proved that if there is a randomized algorithm which is c-competitive against an adaptive offline adversary, then there exist deterministic algorithms which are also c-competitive. In this sense, randomization does not help to improve the competitive ratio of online algorithms against adaptive offline adversaries. In fact, for the list update problem, the adaptive online and adaptive offline adversaries are equally powerful, and the lower bound $2-2/(l+1)$ for deterministic algorithms extends to both adaptive adversaries [119, 121]. So, randomization can only help in obtaining a better competitive ratio when compared against an oblivious adversary. Throughout, by the notion of c-competitiveness for randomized list update algorithms, we mean c-competitiveness against an oblivious adversary.

Random-MTF (Rmtf) is a simple randomized algorithm for the list update problem: after accessing an item, Rmtf moves it to the front with probability 0.5. The competitive ratio of Rmtf is at least 2 [39], which is not better than the best deterministic algorithms. The first randomized algorithm that beats the deterministic lower bound was introduced by Irani [91]. In this algorithm, called Split, each item x has a pointer to another item which precedes it in the list, and after each access to x, the algorithm moves x to either the front of the list or front of the item that x points to (we omit the details here). This randomized algorithm has a competitive ratio of 1.875 [91, 92]. Reingold et al. [121] proposed another randomized algorithm called Bit. Before serving the sequence, the algorithm assigns a bit $b(x)$ to each item x which is randomly set to be 0 or 1. At the time of an access to an element x, the content of $b(x)$ is complemented. Then, if $b(x)=0$, the algorithm moves x to the front; otherwise (when $b(x)=1$), it does nothing. Note that Bit uses randomness only in the initialization phase, and after that it runs deterministically; in this sense the algorithm is *barely random*. It is known that Bit has a competitive ratio of 1.75 [121]. Bit is an instance of the class of Move-To-Front-Every-Other-Access algorithms (also called Mtf2 algorithms). Like Bit, these algorithms maintain a bit for each item and, depending on the value of the maintained bit, move an accessed item to the front of the list or do nothing. Two deterministic examples of Mtf2 algorithms are Move-To-Front-Even (MtfE) and Move-To-Front-Odd (MtfO) in which all bits are initially 0 and 1, respectively. In Section 6.4, we will prove that all deterministic Mtf2 algorithms are 2.5-competitive.

Bit is a member of a more generalized family of online algorithms called Counter [121]. A Counter algorithm has two parameters: an integer s and a fixed subset S of $\{0,1,\ldots,s-1\}$. The algorithm keeps a counter modulo s for each item. With each access to an item x, the algorithm decrements the counter of x and moves it to the front of the list if the new value of the counter is a member of S. With good assignments of s and S, a Counter algorithm can be better than Bit. For example, with $s=7, S=\{0,2,4\}$, the ratio will be 1.735 which is better than the 1.75 of Bit [121].

It is also known that a random reset policy can improve the ratio even further. The algorithm Random Reset (Rst) maintains a counter $c(x)$ for each item x in the list. The counter is initially set randomly to be a number i in the set $\{1,2,\ldots,s-1\}$ with probability π_i. When the requested item has a counter larger than 1, the algorithm makes no move and decrements the counter. If the counter is 1, it moves the item to front and resets the item counter to $i<s$ with probability π_i. Unlike Counter algorithms, Random Reset algorithms are not barely random. The best values of s and π_i's result in an algorithm with a competitive ratio of $\sqrt{3}\approx 1.732$ [121].

The deterministic Timestamp algorithm described earlier is indeed a special case of a family of randomized algorithms introduced by Albers [2]. A randomized Timestamp (p) algorithm has a parameter

p. Upon a request to an item, the algorithm applies the MTF strategy with probability p and the (deterministic) TIMESTAMP strategy with probability $1-p$. The competitive ratio of TIMESTAMP (p) is $\max\{2-p, 1+p(2-p)\}$ which achieves its minimum when $p = (3-\sqrt{5})/2$; this gives a competitive ratio of 1.618. Albers et al. [8] proposed another hybrid algorithm which randomly chooses between two other algorithms. This algorithm is called COMB. Upon a request to an item, the algorithm applies the BIT strategy with probability 0.8 and the (deterministic) TIMESTAMP strategy with probability 0.2. COMB has a competitive ratio of 1.6 [8], which is the best competitive ratio among the existing randomized algorithms for the list update problem.

There has been some research for finding lower bounds for the competitive ratio of randomized list update algorithms [91, 121, 138]. The best existing lower bound is 1.5 proven by Teia [138], assuming the list is sufficiently large. Ambühl et al. [13] proved a lower bound of 1.50084 for randomized online algorithms under the partial cost model, where an algorithm incurs $i-1$ units to access an item in the ith position. Note that there is still a gap between the best upper and lower bounds.

2.2.2 Projective Property

Most existing algorithms for the list update problem satisfy the *projective property*. Intuitively, an algorithm is projective if the relative position of any two items x, y in the list maintained by the algorithm only depends on the requests to x and y in the input sequence and their relative position in the initial configuration.

Assume $\mathbb{A}$ is an online algorithm with the projective property so that the decision of $\mathbb{A}$ on each request is independent of the cost it has paid for previous requests. In order to achieve an upper bound for the competitive ratio of $\mathbb{A}$, it suffices to compare the cost of $\mathbb{A}$ on sequences of two items with that of an optimal algorithm OPT_2 for serving those sequences [39]. Fortunately, the nature of OPT_2 is well-understood and there are efficient optimal offline algorithms for lists of size two [120]. This opens the door for deriving upper bounds for the competitive ratio of projective algorithms under the partial cost model, which also extend to the full cost model.

Ambühl et al. [14, 15] showed that COMB is an optimal randomized projective algorithm under competitive analysis. Consequently, if one wants to improve on the randomized competitive ratio 1.6 of COMB, a new algorithm which is not projective is needed.

2.2.3 List Update and Compression

An important application of the list update problem is in data compression. Such an application was first reported by Bentley et al. [29] who suggested that an online list update algorithm can be used as a subroutine for a compression scheme. Consider each character of a text as an item in the list and the text as the input sequence. A compression algorithm writes an arbitrary initial configuration in the compressed file as well as the access cost of $\mathbb{A}$ for serving each character in unary format. Hence, the size of the compressed file is equal to the access cost of the list update algorithm. The initial scheme proposed in [29] used MTF as its subroutine. Albers and Mitzenmacher [7] used TIMESTAMP and showed that in some cases it outperforms MTF.

In order to enhance the performance of the compression schemes, the Burrows-Wheeler Transform (BWT) can be applied to the input string [41]. In the BWT transformation, the characters of an input

sequence are rearranged into runs of similar characters. Consequently, the resulting permutation has high amount of locality. Moreover, the transformation is reversible in the sense that the original sequence can be retrieved from the BWT permutation without any loss in data. Dorrigiv et al. [65] observed that after applying the BWT, the schemes which use MTF outperform other schemes in most cases.

All the above studies adopt the standard cost model for analysis of compressions schemes. More formally, when an item is accessed in the ith position of the list, the value of i is written in unary format on the compressed file. In practice, however, the value of i is written in binary format using $\Theta(\lg i)$ bits. Hence, the true 'cost' per access is logarithmic in what the standard model assumes. This was first observed in the literature by Dorrigiv et al. [65] where they proposed a new model for the list update problem which is more appropriate for compression purposes. Under this model, the cost of accessing an item in the ith position is $\Theta(\lg i)$. In fact, there is a meaningful difference between the standard model and compression model. Consider the MOVE-FRACTION (MF) family of list update algorithms proposed by Sleator and Tarjan [134]. An algorithm in this family has a parameter k ($k \geq 2$) and upon a request to an item in the ith position, moves that item $\lceil i/k \rceil - 1$ positions towards the front. While MF_k is known to be $2k$-competitive under the standard model [134], it is not competitive under the compression model [65]. For example, MF_2 is 4-competitive under the standard model and $\Omega(\lg l)$ competitive under the compression model. This raises the question whether MTF, which is widely used in compression schemes, is competitive under the compression model. In Section 8, we answer this question in the affirmative.

A randomized algorithm can also be applied for text compression if the random bits used by the algorithm are included in the compressed file. The number of random bits does not change the size of the file dramatically, specially for barely random algorithms like BIT. Similarly, an algorithm under the advice model can be used for compression. As before, the advice bits should be included in the compressed file. We will illustrate this in Section 8 and show that the compression schemes that make use of advice bits perform well in practice.

2.2.4 Locality of Reference

Another issue in the analysis of list update algorithms is that real-life sequences usually exhibit *locality of reference*. Informally speaking, this property suggests that the currently requested item is likely to be requested again in near future. The locality of sequences is particularly evident when the a list update algorithm is used for compression purposes after Burrows-Wheeler Transform. Hester and Hirschberg [89] claimed that, *'Move-To-Front performs best when the list has a high degree of locality'*. Angelopoulos et al. [17, 18] formalized this claim by showing that MTF is the unique optimal solution under bijective analysis for sequences that have locality of reference with respect to *concave analysis*. Under concave analysis, a sequence has locality if it is consistent with a concave function f so that the maximum number of distinct requests in a window of size τ is at most $f(\tau)$ [3].

Inspired by the concave analysis, Dorrigiv et al. [64] quantified the locality of input sequences. The *non-locality* of a sequence σ of length n, denoted by $\hat{\lambda}$, is defined as $\sum_{i=1}^{n} d_i$ in which d_i is the number of distinct items requested since the last request to $\sigma[i]$ (the item requested at index i of σ). For the first request to an item, d_i is equal to the length of the list. It is known that the cost of any online algorithm is at least $\hat{\lambda}$, while MTF is optimal in this sense and has a cost of $\hat{\lambda}$. The cost of TIMESTAMP is at least $2\hat{\lambda}$, and TRANSPOSE and FC both have a cost of at least $m/2 \times \hat{\lambda}$ [64]. These results imply an advantage for MTF when sequences have high locality.

Albers and Lauer [4] further studied the problem under locality of reference assumptions. They defined a new model which is based on the number of *runs* in input sequences. For an input sequence $\sigma_{x,y}$ involving two items x and y, a run is a maximal subsequence of requests to the same item. A run is *long* if it contains at least two requests and *short* otherwise. Consider a long run R of requests to x, and let R' denote the next long run which comes after R in the sequence. Note that there might be short runs between R and R'. If R' is formed by requests to y, then a *long run change* happens. Also, a single extra long run change happens when the first long run and the first request of the sequence reference the same item. For an arbitrary sequence σ (defined on potentially more than two items), let the *projected sequence* over a pair (x, y) of items be a copy of σ in which all requests except those to x or y are removed. Define the number of runs (respectively long run changes) of σ as the total number of runs (respectively long run changes) of the projected sequences over all pairs (x, y). Let $r(\sigma)$ and $l(\sigma)$ respectively denote the total number of runs and long run changes in σ. Define $\lambda = \frac{l(\sigma)}{r(\sigma)}$, i.e., λ represents the fraction of long run changes among all the runs. Note that we have $0 \leq \lambda \leq 1$. The larger values of λ imply a higher locality of the sequence, e.g., when all runs are long, we get $\lambda = 1$. Also, note that the length of long runs does not affect the value of λ. Using this notion of locality, the competitive ratio of MTF is at most $\frac{2}{1+\lambda}$, i.e., for sequences with high locality MTF is almost 1-competitive. The ratio of TIMESTAMP does not improve on request sequences satisfying λ-locality, i.e., it remains 2-competitive. The same holds for COMB, i.e., it remains 1.6-competitive. However, for BIT, the competitive ratio improves to $\min\{1.75, \frac{2+\lambda}{1+\lambda}\}$.

2.3 Remarks

Although both list update and bin packing algorithms are well-studied from a theoretical point of view, there are many open questions which remain to be answered. In the case of bin packing, there is still a gap between the competitive ratio of the best algorithm and the best existing lower bound. The same holds for the competitive ratio of the best randomized list update algorithm. While studying these open questions is important from a theoretical point of view, answering them seems to have little bearing on practice. This is partially due to the fact that the existing models and analysis methods are not suitable for all practical scenarios. In our study of these two problems in the following parts of the thesis, we consider different perspectives which bring the applicability of the problems into account.

Part II

Beyond Competitive Analysis

Chapter 3

A Near-Optimal Online Bin Packing Algorithm

In the survey of bin packing by Coffman et al. [49], it is stated that *'All algorithms that do better than* First Fit *in the worst-case seem to do much worse in the average case.'* In this chapter, however, we show that this is not necessarily true and give an algorithm whose competitive ratio, average-case ratio, and expected wasted space are all at or near the top of each class. This also addresses a conjecture by Gu et al. [84] stated as *'Harmonic-K is better than* First Fit *in the worst-case performance, and* First Fit *is better than Harmonic-K in the average-case performance. Maybe there exists an on-line algorithm with the advantages of both First Fit and Harmonic-K.'*

3.1 Introduction

We would like to present an algorithm with optimal average-case and close-to-best worst-case performance for the bin packing problem. As mentioned in the previous chapter, it has long been observed that bin packing algorithms with optimal average-case performance were not optimal in the worst-case sense. We introduce an algorithm called Harmonic Match (Hm) which is better than Bf and Ff in the worst case. At the same time, it performs as well as Bf and Ff on average. In particular, we show the competitive ratio of Hm is as good as Ha, i.e., it approaches $T_\infty \approx 1.691$ for sufficiently large values of K, where K is the parameter of the algorithm (number of classes that it defines). For sequences generated uniformly at random, the average performance ratio of Hm is 1 which is as good as Bf and Ff. The expected waste of Hm is $\Theta(\sqrt{n}\lg^{3/4} n)$ which is as good as Bf and better than Ff.

The idea behind Hm can be used as a general way to improve the performance of the Super Harmonic class of algorithms. We illustrate this for the simplest member of this family, namely the Refined Harmonic algorithm of Lee and Lee [106]. To do so, we introduce a new algorithm called Refined Harmonic Match (Rhm). We show that the competitive ratio of this algorithm is at most equal to the 1.636 of Refined Harmonic. At the same time, the average-case ratio of Rhm is 1 which is as good as Bf and Hm. The expected waste of Rhm is equal to that of Bf. Consequently, the algorithm achieves the desired

Algorithm	Average Ratio	Expected waste	Competitive Ratio
Next Fit (Nf)	$1.\bar{3}$ [50]	$\Omega(n)$	2
Best Fit (Bf)	1 [28]	$\Theta(\sqrt{n}\lg^{3/4} n)$ [132, 107]	1.7 [95]
First Fit (Ff)	1 [107]	$\Theta(n^{2/3})$ [132, 51]	1.7 [95]
Harmonic (Ha)	1.2899 [106]	$\Omega(n)$	$\to T_\infty \approx 1.691$ [106]
Refined First Fit (Rff)	> 1	$\Omega(n)$	$1.\bar{6}6$ [142]
Refined Harmonic (Rh)	1.2824 [84]	$\Omega(n)$	1.636[106, 84]
Modified Harmonic (Mh)	1.189 [117]	$\Omega(n)$	1.615[118]
Harmonic++	> 1	$\Omega(n)$	1.588 [131]
Harmonic Match (Hm)	**1**	$\mathbf{\Theta(\sqrt{n}\lg^{3/4} n)}$	$\mathbf{\to T_\infty \approx 1.691}$
Refined Harmonic Match (Rhm)	**1**	$\mathbf{\Theta(\sqrt{n}\lg^{3/4} n)}$	**1.636**

Table 3.1: Average performance ratio, expected waste, and competitive ratios for different bin packing algorithms. Results in bold are our contributions.

average-case performance of Bf and also the worst-case performance of Refined Harmonic. Table 3.1 is an extension of Table 2.1 which gives a summary of our results when compared to the existing online bin packing algorithms.

Recall that Harmonic (Ha) algorithm with parameter K defines K classes $(1/2, 1]$, $(1/3, 1/2]$, ..., $(1/(K-1), 1/K]$, and $(0, 1/K]$. Items in the same class are separately treated using the Next Fit strategy. Just as with the Harmonic algorithm, Hm and Rhm are based on classifying items based on their sizes and treating items of each class (almost) separately. To boost the average-case performance, these algorithms *match* large items with proportionally smaller items through assigning them to the same classes (Recall that an item is large if it is larger than 1/2 and small otherwise). Careful definition of classes results in the same average-case performance as Mbf. To some extent, our competitive analyses of Hm and Rhm are similar to those of Ha and Refined Ha, respectively. Similarly, the average-case analyses of the algorithms are closely related to the analysis of Mbf algorithm and uses similar techniques.

For the bulk of this Chapter, we assume the item sizes are distributed uniformly in the interval $[0, 1]$, where bins are of unit capacity. However, to evaluate the average-case performance of the introduced algorithms in the real-world, we test them on sequences that follow *discrete* uniform distribution as well as other distributions. The results of our comparisons suggest that Hm and Rhm have comparable performance with Bf and Ff. At the same time, they have a considerable advantage over other members of the Harmonic family of algorithms.

3.2 Harmonic Match Algorithm

Similarly to Harmonic algorithm, Harmonic Match has a parameter K and divides items into K classes based on their sizes. We use Hm_K to refer to Harmonic Match with parameter K. The algorithm defines K pairs of intervals as follows. The ith pair $(1 \le i \le K-1)$ contains intervals $(\frac{1}{i+2}, \frac{1}{i+1}]$ and $(\frac{i}{i+1}, \frac{i+1}{i+2}]$. The Kth pair includes intervals $(0, \frac{1}{K+1}]$ and $(\frac{K}{K+1}, 1]$. An item x belongs to class i if the size of x lies in any of the two intervals associated with the ith pair (see Figure 3.1). Intuitively, the items which are 'very

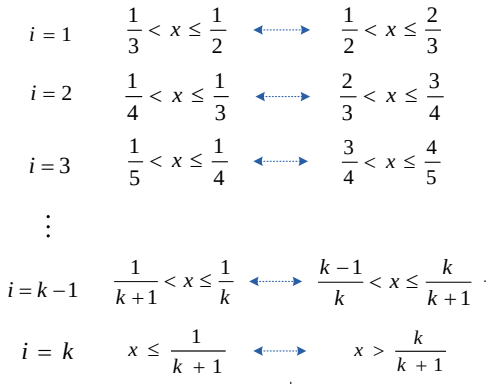


Figure 3.1: The classes defined by HM. The algorithm matches items from intervals indicated by arrows.

large' or 'very small' belong to the Kth class, and as the item sizes become more moderate, they belong to classes with smaller indices.

When compared to the intervals of HARMONIC algorithm, one can see that the first interval of the ith pair in the HARMONIC MATCH algorithm HM_K is the same as the $(i+1)$th interval of HARMONIC algorithm HA_{K+1} $(1 \le i \le K)$. Namely, the intervals are the same in both algorithms except that the interval $(\frac{1}{2}, 1]$ of HA is further divided into $K+1$ more intervals. In other words, HM_K is similar to HA_{K+1}, except that it tries to match large items with proportionally smaller items. The pairs of intervals which define a class in HM have the same length, e.g., in the first pair, both intervals have length $\frac{1}{6}$. This property is essential for having good average-case performance for our uniform distribution on $[0, 1]$. In other words, items of sizes x and $1 - x$ appear with the same probability in a class. This allows boosting the average case performance by following a similar strategy as BEST FIT for items inside each class. Moreover, the HARMONIC-type classification of items allows improvement on the competitive ratio. In what follows, we formalize these intuitions.

The packing maintained by HM includes two types of bins: the *mature* bins which are almost full (can be thought as being closed) and *normal* bins which might become mature by receiving more items. For placing an item x, HM detects the class that x belongs to and applies the following strategy to place x. If x is a large item $(x > 1/2)$, HM opens a new bin for x and declares it as a normal bin. If x is small $(x \le 1/2)$, the algorithm applies the BEST FIT (BF) strategy to place x in a mature bin. If there is no mature bin with enough space, the BF strategy is applied again to place x in a normal bin that contains the largest 'companion' of x. A companion of x is a large item of the same class that fits with x in the same bin. In case the BF strategy succeeds to place x in a bin (i.e., there is a normal bin with a companion of x) the selected bin is declared as a mature bin. Otherwise (when there is no companion for x), the algorithm applies the NEXT FIT strategy to place x in a single normal bin maintained for that class; such a bin only includes small items of the same class. If the bin maintained by the NF strategy does not have enough space, it is declared as a mature bin and a new NF-bin is opened for x. Note that HM, as defined above, is simple to implement and its time complexity is as good as BF.

HM treats items of the same class in a similar way that ONLINE MATCH (OM) does, except that there is no restriction on the sum of the sizes of two companion items. Recall that OM has a parameter which

defines a lower bound for the sum of two items in a bin. In order to facilitate our analysis in the following sections, we define Relaxed Online Match (Rom) algorithm as a subroutine of Hm that works as follows. To place a large item, Rom opens a new bin. To place a small item x, it applies the Bf strategy to place x in an open bin with a single large item and closes the bin. If such a bin does not exists, Rom places x using the Nf strategy (and opens a new bin if necessary). Using Rom, we can describe the Harmonic Match algorithm in the following way. To place a small item, Hm_K tries to place it in a mature bin using the Bf strategy. Large items and the small items which do not fit in mature bins are treated using the Rom strategy along with other items of their classes (which did not fit in the mature bins). The bins which are closed by the Rom strategy are declared as mature bins.

3.2.1 Worst-Case Analysis

For the worst-case analysis of Hm, we observe that the Harmonic algorithm is *monotone* in the sense that removing an item does not increase its cost.

Lemma 2. *Removing an item does not increase the number of bins used by the* Harmonic *algorithm.*

Proof. Recall that the Harmonic algorithm with K classes (Ha_K) defines a class for each item and applies the Nf strategy to place each item together with items of the same class. So, the number of bins used by the algorithm to pack a sequence σ is $\text{Nf}(\sigma_1) + \text{Nf}(\sigma_2) + \ldots + \text{Nf}(\sigma_K)$, where σ_i is the sequence of items which belong to class i. Assume an item x is removed from σ, and let j denote the class that x belongs to ($1 \leq j \leq K$). The number of bins used by Ha_K for the reduced sequence (in which x is removed) will be the same except that $\text{Nf}(\sigma_j)$ is replaced by $\text{Nf}(\sigma'_j)$, where σ'_j is a copy of σ_j in which x is missing. It is known that Nf is monotone (see, e.g., [116]). Hence, we have $\text{Nf}(\sigma'_j) \leq \text{Nf}(\sigma_j)$. Consequently, the cost of Ha cannot increase after removing x. □

We use the above lemma to show that the number of bins used by Hm_K to pack any sequence σ is no larger than that of Ha_{K+1}. Consequently, the competitive ratio of Hm_K is no larger than that of Ha_{K+1}.

Theorem 1. *The number of bins used by* Hm_K *to pack any sequence* σ *is no larger than that of* Ha_{K+1}.

Proof. Consider the packing of σ by Hm . We say a small item is *red* if it is packed with a large item in the same bin; otherwise, it is *white*. Consider the sequence σ' which is the same as σ except that the red items are removed. We claim $\text{Hm}_K(\sigma) = \text{Ha}_{K+1}(\sigma')$. Let σ_i denote the sequence of items which belong to class i of Hm_K ($1 \leq i \leq K$). The number of bins used by Hm_K to pack σ_i is $l_i + \text{Nf}(W_i)$, where l_i is the number of large items σ_i and W_i is the sequence formed by white items in σ_i. Let σ'_i be a subsequence of σ_i in which red items are removed (hence, it is also a subsequence of σ'). Since small and large items are treated separately by Ha_{K+1}, the number of bins used by Ha_{K+1} to pack σ'_i is also $l_i + \text{Nf}(W_i)$. Hence, $\text{Hm}_K(\sigma_i) = \text{Ha}_{K+1}(\sigma'_i)$. Taking the sum over all classes, we get $\text{Hm}_K(\sigma) = \text{Ha}_{K+1}(\sigma')$. On the other hand, by Lemma 2, Ha is monotone and $\text{Ha}_{K+1}(\sigma') \leq \text{Ha}_{K+1}(\sigma)$. Consequently, $\text{Hm}_K(\sigma) \leq \text{Ha}_{K+1}(\sigma)$. □

We show that the upper bound given in the above theorem is tight. Consequently, we have:

Corollary 1. *The competitive ratio of* Hm_K *is equal to that of* Ha_{K+1}*, i.e., it converges to* $T_\infty \approx 1.691$ *for large values of* K.

Proof. Let $\alpha < T_\infty$ denote a lower bound for the competitive ratio of HA_{K+1} and consider a sequence σ for which the number of bins used by HA_{K+1} is α times more than that of OPT. Define a sequence σ_π as a permutation of σ in which items are sorted in increasing order of their sizes. When applying HM_K on σ_π, all large items will be unmatched in their bins (no other item is packed in their bins). Hence, the number of bins used by HM_K to pack σ_π is equal to the number of bins used by HA_{K+1} to pack σ, i.e., α times the number of bins used by OPT for σ and σ_π (Note that OPT creates an identical packing for both σ and σ_π). □

To achieve a competitive ratio better than 1.7 of BF and FF for HM_K, it is sufficient to have $K \geq 6$. In that case, HM_6 performs as well as HA_7, which has a competitive ratio of at most 1.695.

3.2.2 Average-Case Analysis

In this section, we study the average-case performance of the HM algorithm, assuming the item sizes are distributed uniformly in the interval $[0, 1]$. Like most related work, we make use of the results related to the *up-right matching* problem. An instance of this problem includes n points generated uniformly at random in a unit-square in the plane. Each point receives a $\oplus$ or $\ominus$ label with equal probability. The goal is to find a maximum matching of $\oplus$ points with $\ominus$ points so that in each pair of matched points the $\oplus$ point appears above and to the right of the $\ominus$ point. Let U_n denote the number of unmatched points in an optimal up-right matching of n points. For the expected size of U_n, it is known that $E[U_n] = \Theta(\sqrt{n}\lg^{3/4} n)$ [132, 107, 124, 53]. Given an instance of bin packing defined by a sequence σ, one can make an instance of up-right matching as follows [99]. Each item x of σ is plotted as a point in the unit square. the vertical coordinate of the point corresponds to the index of x in σ (scaled to fit in the square). If x is smaller than $1/2$, the point associated with x is labelled as $\oplus$ and its horizontal coordinate will be $1 - 2s(x)$ (recall that $s(x)$ denotes the size of x); otherwise, the point will be $\ominus$ and its horizontal coordinate will be $2s(x) - 1$. Note that the resulting point will be bounded in the unit square. A solution to the up-right matching instance gives a packing of σ in which the items associated with a pair of matched points are placed in the same bin. Note that the sum of the sizes of these two items is no more than the bin capacity. Also, in such a solution, each bin contains at most two items.

For our purposes, we study σ_t as a subsequence of σ which only includes items which belong to the same class in the HM algorithm. The items in σ_t are generated uniformly at random from $(\frac{1}{t+1}, \frac{1}{t}] \cup (\frac{t-1}{t}, \frac{t}{t+1}]$. Since the two intervals have the same length, the items can be plotted in a similar manner on the unit square as follows. The horizontal coordinate of a small item with size x is $1 - (s(x) \times t(t+1) - t)$ and for large items it is $s(x) \times t(t+1) - (t^2 - 1)$. The label of the item and its vertical coordinate are defined as before.

Any bin packing algorithm which closes a bin after placing a small item can be applied to the up-right matching problem. Each edge in the up-right matching instance corresponds to a bin which includes one small and one large item. Recall that the algorithm MATCHING BEST FIT (MBF) applies a BF strategy except that it closes a bin as soon as it receives an item with size smaller than or equal to $1/2$. So, MBF can be applied for the up-right matching problem. Indeed, it creates an optimal up-right matching, i.e., if we apply MBF on a sequence σ_t which is randomly generated from $(0, 1]$, the number of unmatched points will be $\Theta(\sqrt{n_t}\lg^{3/4} n_t)$, where n_t is the length of σ_t [132]. We show the same result holds for the bin packing sequences in which items are taken uniformly at random from $(\frac{1}{t+1}, \frac{1}{t}] \cup (\frac{t-1}{t}, \frac{t}{t+1}]$:

Lemma 3. *For a sequence σ_t of length n_t in which item sizes are selected uniformly at random from $(\frac{1}{t+1}, \frac{1}{t}] \cup (\frac{t-1}{t}, \frac{t}{t+1}]$, we have $E[\textsc{Mbf}(\sigma_t)] = n_t/2 + \Theta(\sqrt{n_t}\lg^{3/4} n_t)$.*

Proof. Define an instance for up-right matching from σ_t as follows. Let x be the ith item of σ_t ($1 \le i \le n_t$). If x is small, plot a point with $\oplus$ label at position $(1 - (s(x) \times t(t+1) - t), i/n_t)$; otherwise, plot a point with $\oplus$ label at position $(s(x) \times t(t+1) - (t^2 - 1), i/n_t)$. This way, the points will be bounded in the unit square. Since the item sizes are generated uniformly at random from the two intervals and the sizes of the intervals are the same, the point locations and labels are assigned uniformly at random. As a result, the number of unmatched points in the up-right matching solution by Mbf is expected to be $\Theta(\sqrt{n_t}\lg^{3/4} n_t)$. The unmatched points are associated with the items in σ_t which are packed as a single item in their bins by Mbf. Let sg denote the number of such items; hence, $E[sg] = \Theta(\sqrt{n_t}\lg^{3/4} n_t)$. Except these sg items, other items are packed with exactly one other item in the same bin. So we have $\textsc{Mbf}(\sigma_t) - sg \le n_t/2$ which implies $E[\textsc{Mbf}(\sigma_t)] = n_t/2 + E[sg]$. Since $E[sg] = \Theta(\sqrt{n_t}\lg^{3/4} n_t)$, the statement of the lemma follows. □

Recall that we used Rom as a subroutine of Mbf. We show that Rom is no worse than Mbf.

Lemma 4. *For any instance σ of the bin packing problem, the number of bins used by* Rom *to pack σ is no more than that of* Mbf.

Proof. Both Rom and Mbf open a new bin for each large item. Also, they treat small items which have companions in the same way, i.e., they place the item in the bin of the largest companion and close that bin. The only difference between Rom and Mbf is in placing small items without companions where Rom applies the Nf strategy while Mbf opens a new bin for each item. Trivially, Rom does not open more bins than Mbf for these items. □

Lemma 5. *Removing an item does not increase the number of bins used by* Mbf.

Proof. Let σ denote an input sequence and n denote the length of σ. We use a reverse induction to show that removing the $(n-i)$th item ($0 \le i \le n-1$) does not increase the number of bins used by Mbf to pack σ. Note that removing the last item does not increase the number of bins used by of any algorithm and the base of induction holds. Assume the statement holds for $i = k+1$, i.e., removing any item from index $i \ge k+1$ does not increase the number of bins used by Mbf. We show the same holds for $i = k$. Let n_l denote the number of large items in σ and n_{ss} denote the number of *single small* items, which are the small items which have no companion. When placing a single small item, Mbf opens a bin and closes the bin right after placing the item. So, the number of bins used by Mbf to pack σ will be $\textsc{Mbf}(\sigma) = n_l + n_{ss}$. Let x denote the kth item in σ. We show that removing x does not increase the number of bins used by Mbf. There are a few cases to consider.

First, note that if x is a single small item, removing it decreases the number of bins of Mbf by one unit. Since the packing of other items does not change, the inductive step trivially holds. Next, assume x is a small item which has a companion. Removing x might create a space for another small item x' in the bin of x. In case such an item does not exist (i.e., no other item replaces x in its bin), the packing and consequently the number of bins used by the algorithm does not change. Otherwise, x' is placed in the bin which includes the companion of x and closes that bin. Let k' denote the index of x' in the sequence and note than $k' > k$. Also, let σ^{-a} denote a copy of σ from which an item a is removed. We have

$\textsc{Mbf}(\sigma^{-x}) = \textsc{Mbf}(\sigma^{-x'})$, i.e., removing item x changes the number of bins used by Mbf in the same way that removing x' does. By the induction hypothesis, removing x' does not increase the number of bins of Mbf and we are done.

The only remaining cases is when x is a large item. If x does not have a companion, removing x decreases the number of bins by one unit and we are done. Next, assume x has a companion x' and let σ^{--} denote the same sequence as σ in which both x and x' are removed. As before, let σ^{-x} denote a copy of σ in which x is removed. We have $\textsc{Mbf}(\sigma^{--}) = \textsc{Mbf}(\sigma) - 1$. On the other hand, $\textsc{Mbf}(\sigma^{-x}) \leq \textsc{Mbf}(\sigma^{--}) + 1$. This is because adding a small item to a sequence does not increase the number of bins used by Mbf by more than one unit; this holds because Mbf closes a bin as soon as a small item is placed in the bin. We conclude that $\textsc{Mbf}(\sigma^{-x}) \leq \textsc{Mbf}(\sigma)$, and the inductive step holds. □

Provided with the above lemmas, we prove the following theorem.

Theorem 2. *Let σ be a sequence of length n in which item sizes are selected uniformly at random from $(0, 1]$. The expected wasted space of* Hm *for packing σ is $\Theta(\sqrt{n}\lg^{3/4} n)$.*

Proof. Let σ^- be a copy of σ in which the items which are placed in mature bins are removed. Let $\sigma_1^-, \ldots, \sigma_K^-$ be the subsequences of σ^- formed by items belonging to different classes of Hm. We have:

$$\textsc{Hm}(\sigma) = \sum_{t=1}^{K} \textsc{Rom}(\sigma_t^-) \leq \sum_{t=1}^{K} \textsc{Mbf}(\sigma_t^-) \leq \sum_{t=1}^{K} \textsc{Mbf}(\sigma_t)$$

The inequalities come from Lemmas 4 and 5, respectively. Consequently, by Lemma 3, we have:

$$E[\textsc{Hm}(\sigma)] \leq \sum_{t=1}^{K} \left(n_t/2 + \Theta(\sqrt{n_t}\lg^{3/4} n_t) \right) = \frac{n}{2} + \Theta(\sqrt{n}\lg^{3/4} n)$$

Note that the last equation holds since K is a constant. The expected value of $s(\sigma)$, the total size of items in σ, is $n/2$ (since the expected size of an item is $1/2$ and item sizes are independent). Consequently, for the expected waste of Hm algorithm, we have:

$$E[\textsc{Hm}(\sigma) - s(\sigma)] = E[\textsc{Hm}(\sigma)] - E[s(\sigma)] = n/2 + \Theta(\sqrt{n}\lg^{3/4} n) - n/2 = \Theta(\sqrt{n}\lg^{3/4} n)$$

□

It should be mentioned that, although the expected waste of Hm algorithms is $\Theta(\sqrt{n}\lg^{3/4} n)$, there is a multiplicative constant involved in this expression which depends on K. This implies that the rate of convergence to Bf is slower for larger values of K.

3.3 Refined Harmonic Match

In this section, we introduce a slightly more complicated algorithm, called Refined Harmonic Match (Rhm), which has a better competitive ratio than Bf and Hm while performing as well as them on average.

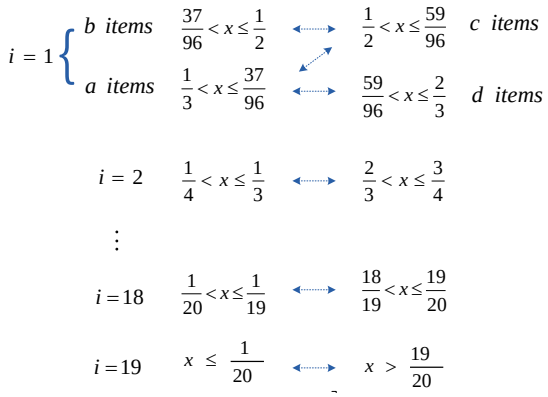


Figure 3.2: The classes defined by RHM. The algorithm matches items from intervals indicated by arrows.

Similar to HM, RHM divides items into a constant (*i.e.*, $O(1)$) number of classes and treat items of each class separately. The classes defined for RHM are the same as those of HM_K with $K = 19$. The items which belong to class $t \geq 2$ are treated using the HM strategy, i.e., a set of mature bins are maintained. If an item fits in mature bins, it is placed there using the BF strategy; otherwise, it is placed together with similar items of its class using the ROM strategy. At the same time, the bins closed by the ROM strategy are declared as being mature.

The only difference between HM and RHM in packing items of class 1, i.e., items in the range $(1/3, 2/3]$. RHM divides items in this range into four groups $a = (1/3, 37/96]$, $b = (37/96, 1/2]$, $c = (1/2, 59/96]$, and $d = (59/96, 2/3]$ (see Figure 3.2). Similar to REFINED HARMONIC, to handle the bad sequences which result in the lower bound of T_∞ for competitive ratios of HA and MH, RHM designates a fraction of bins opened by items of group a to host future c items. Note that the total size of a c item and an a item is no more than 1. To ensure a good average-case performance, RHM should be more elaborate than REFINED HARMONIC as it cannot treat b items apart from other items using a strategy like NF (as REFINED HARMONIC does).

In what follows, we introduce an online algorithm called REFINED RELAXED ONLINE MATCH (RRM) as a subroutine of RHM that is specifically used for placing items of class 1. To place an item x of class 1 ($x \in (1/3, 2/3]$), RRM uses the following strategy. At each step of the algorithm, when two items of class 1 are placed in the same bin, that bin is declared as being mature and will be used for placing small items of other classes. More precisely, it will be added to the set of mature bins maintained by the HM algorithm that packs items of other classes. If x is a d-item, RRM opens a new bin for x. If x is a c item, the algorithms checks whether there are bins with a single a item designated to be paired with a c item. In case there are, x is placed in a bin with an a item using the BF strategy. Otherwise, a new bin is opened for x. For a and b items (small items of class 1), RRM uses the BF strategy to select a bin with enough space which includes a single large item (if there is such a bin). This is particularly important to guarantee a good average-case behavior. If x is a b item, the algorithms checks the bin with the highest level in which x fits; if such a bin includes a c or a b item, x is placed there. Otherwise (when there is no selected bin or when it has an a item), a new bin is opened for x. If x is an a item, the algorithm uses the BF strategy to place it into a bin with a d or c item. If no suitable bin exist, x is placed in a bin with a single a item (there is at most one such bin). If there is no such bin, a new bin is opened for x.

Algorithm 1: RRM algorithm: Placing a sequence of items in the range (1/3,2/3]

```
input: A sequence σ = ⟨σ_1, σ_2, ..., σ_n⟩ of items in the range (1/3,2/3]
N_{a_1}, N_{a_2}, N_{aa}, N_{ab}, N_{ac}, N_b, N_{bc} ← 0
for i ← 1 to n do
    switch σ_i do
        case d item:
            open a new bin for σ_i
        case c item:
            if N_{a_1} > 0 then
                use the BF strategy to place σ_i in a bin with an a item
                N_{a_1} ← N_{a_1} − 1; N_{ac} ← N_{ac} + 1;
                {N_{a_1} is the number of bins with a single a item which are designated to have a c item}
            else open a new bin for σ_i; N_c ← N_c + 1;
        case b item:
            select the bins with a c item which have enough capacity for σ_i
            if there is a selected bin then
                place σ_i into the bin with the highest level among the selected bins;
                N_{bc} ← N_{bc} + 1; N_c ← N_c − 1;
            else if N_b = 1 then
                place σ_i into the bin with a single b item; N_b ← 0;
            else open a new bin for σ_i; N_b ← 1;
        case a item:
            select the bins with a d item which have enough capacity for σ_i
            if there is such a bin then
                place σ_i into the bin with the highest level among those bins
            else if N_c > 0 then
                place σ_i into the bin with the largest c item; N_{ac} ← N_{ac} + 1; N_c ← N_c − 1;
            else if N_{a_2} = 1 then
                place σ_i into any with a single a item; N_{a_2} ← 0; N_{aa} ← N_{aa} + 1;
            else
                place σ_i in a new (empty) bin
                {compare the number of red bins with 3 times number of blue bins}
                if N_{aa} < 3(N_{ac} + N_{a_1} + N_{bc}) then
                    N_{a_2} ← 1; {declare the opened bin as a red bin (an a_2-bin) }
                else
                    N_{a_1} ← N_{a_1} + 1; {declare the opened bin as a blue bin (an a_1-bin) }
    endsw
end
```

We define *red bins* as those which include two a items or a single a item designated to be paired with another a item, and define *blue bins* as those which include either a c item together with an a or a b item or a single a item designated to be paired with a c item in the future. When opening a new bin for an a item, RHM tries to maintain the number of red bins as close to three times the number of blue bins as possible. Namely, if the number of red bins is less than 3 times of blue bins, it declares the opened bin as a red bin to host another a item in the future; otherwise, the new open bin is declared as a blue bin to host a c item in the future. This way, the number of red bins is close to (but never more than) three times that of blue bins. Note that, when many b items are placed together with c items, the resulting bins are blue. In this case, the algorithm does not limit the number of blue bins unless it opens bins for a items. Consequently, the number of red bins can be less than three times the number of blue bins. Algorithm 1 illustrates how RRM works.

3.3.1 Worst-Case Analysis

In this section, we prove an upper bound of 1.636 for the competitive ratio of RHM. We start by introducing some notation. Bins in a packing by RRM can be divided into the following groups: *d-bins* which include a d-item (might also include an a item), *c-bins* (respectively *b-bins)* which include a single c (respectively b) item, a_1-*bins* (respectively a_2-*bins*) which include a single a item and are designated to include a c (respectively an a) item in the future, *bb-bins* (respectively *aa-bins*) which include two b (respectively a) items, *ac-bins* which include an a item and a c item, and *bc-bins* which include a b item and a c item. Note that there is at most one b-bin and one a_2-bin (otherwise, two of those bins should have formed a *bb*-bin or an *aa*-bin, respectively). We use capital N and lower case n to refer to the number of bins and items, respectively. N_α denotes the number of bins of type α, e.g., N_{ac} indicates the number of *ac*-bins. Similarly, N_{red}, N_{blue} denote the number of red and blue bins in the packing. Note that $N_{red} = N_{aa} + N_{a_2}$ and $N_{blue} = N_{ac} + N_{bc} + N_{a_1}$. We use n_τ to denote the number of items of type τ ($\tau \in \{a, b, c, d\}$). Moreover, we use n_{b_1} to denote the number of b items which are packed with a c item ($n_{b_1} = N_{bc}$) and n_{b_2} to denote the number of other b items ($n_{b_1} + n_{b_2} = n_b$). Counting the number of a and b_1 items we get $n_a + n_{b_1} = 2N_{aa} + N_{ac} + N_{a_1} + N_{a_2} + N_{bc}$. Since $N_{a_2} \leq 1$, by definition of red and blue bins, we get the following.

$$n_a + n_{b_1} \leq 2N_{red} + N_{blue} \leq n_a + n_{b_1} + 1 \tag{3.1}$$

We refer to the above inequalities in a few places in our analysis. Since RHM uses the same strategy as HM for placing items in classes $k \geq 2$, and HM never opens more bins tha HA does (Theorem 1), we can prove the following lemma.

Lemma 6. *The number of bins used by* RHM *to pack items of classes $t \geq 2$ is upper bounded by*

$$\text{RHM}(\sigma) \leq \text{RRM}(\sigma_{cl_1}) + n_X + \sum_{t=2}^{18} \frac{n_t}{t+1} + 20W'/19 + 20$$

in which σ_{cl_1} is the subsequence formed by items of class 1, n_X is the number of large items in classes other than class 1, n_t is the number of small items in class t, and W' is the total size of small items in class 19 (the last class).

Proof. Since RHM performs similarly to HM for placing items of class $t \geq 2$, the proof of Theorem 1 can be applied to state that RHM does not open more bins than the HARMONIC algorithm for placing these items. HARMONIC opens a new bin for each large item; this sums up to n_X for all large items (except those of class 1). HARMONIC places $t + 1$ items of class t in the same bin ($2 \leq t \leq 18$); hence, it opens at most $\frac{n_t}{t+1} + 1$ bins for small items in such class. The empty space in any bin assigned to items of the last class (class 19) is at most 1/20 since the size of items in this class is no more than 1/20. Hence, the number of opened bins for this class is at most $20W'/19 + 1$. Note that some items in classes $t \geq 2$ might be placed in the mature bins maintained by HM (including the bins released by RRM). When comparing with the HARMONIC algorithm, we can think of these items as being removed. Since the HARMONIC is monotone (Lemma 2), removing these items does not increase the number of bins. Hence, the claimed upper bound still holds. □

Theorem 3. *The competitive ratio of* RHM *is at most* $373/228 < 1.636$.

Proof. RRM is defined in a way that no c-bin and a_1-bin can be open at the same time. We consider the following two cases based on the packing of RRM for the subsequence σ_{cl_1} formed by items of type 1.

- Case 1: In the final packing, there is at least one a_1-bin while there is no c-bin.
- Case 2: There is no a_1-bin in the final packing.

We prove the theorem for the above cases separately.

Case 1: Assume there is no c-bin in the final packing while there is at least one a_1-bin. Let x be the last a item for which an a_1-bin is opened. We claim that no blue bin is added to the packing after placing x. Blue bins are opened by a or c items. A new blue bin cannot be opened by a c item as such a c item should have been placed in one of the existing a_1 bins. Also, a new blue bin cannot be opened by an a item since that results in a bin with a single a item (an a_1-bin)); this contradicts x being the last item for which an a_1 bin is opened. So, the number of blue bins does not increase after placing x. At the time of placing x, the number of red bins is no less than three times the number of blue bins; otherwise, the bin opened for x would have been declared as a red bin (i.e., an a_2-bin). So, for the final packing, we have $3N_{blue} - N_{red} \leq 3$. Using Equation 3.1 we get:

$$N_{red} + N_{blue} \leq 4n_a/7 + 4n_{b_1}/7 + 1$$

And for the total number of bins used by RRM we will have:

$$\begin{aligned}
\text{RRM}(\sigma_{cl_1}) =& N_d + N_{ac} + N_{bc} + N_b + N_{bb} + N_{a_1} + N_{a_2} + N_{aa} \\
&\leq n_d + n_{b_2}/2 + N_{blue} + N_{red} \\
&\leq n_d + n_{b_2}/2 + 4n_a/7 + 4n_{b_1}/7 + 1 \\
&\leq n_d + 4n_a/7 + 4n_b/7 + 1
\end{aligned}$$

Plugging this into the upper bound given by Lemma 6, we get:

$$\text{RHM}(\sigma) \leq n_X + \sum_{t=2}^{18} \frac{n_t}{t+1} + 20W'/19 + n_d + 4n_a/7 + 4n_b/7 + 21 \tag{3.2}$$

To further analyze the algorithm, we use a weighting function similar to that of [106]. We define a weight for each item so that the total weight of items in a sequence, denoted by $W(\sigma)$, becomes an upper bound for the number of bins used by RHM (within an additive constant). At the same time, we show that the total weight of any set of items which fit in a bin is at most 1.63; this implies that the number of bins used by OPT to pack σ is at least $W(\sigma)/1.63$. Consequently, the ratio between the number of bins used by RHM and OPT is at most 1.63.

The weights are defined in the following manner. The weights of d items and large items of classes other than class 1 (i.e., items larger than 2/3) are 1. The weights of c item are 0. The weights of b and a items are both 4/7. Small items of class t ($2 \leq t \leq 18$) have weight $1/(t+1)$. The weight of a small item x of class 19 is $20s(x)/19$. This way, as Inequality 3.2 suggests, the number of bins used by RHM is no more than the total weight of items (within an additive constant).

Next, we study the maximum weight of items in a bin of OPT. Let $\beta_1, \beta_2, \ldots, \beta_t$ denote the size of items in a bin of OPT so that $\beta_1 \geq \beta_2 \geq \ldots \geq \beta_t$. Let W_{opt} denote the total weight of items in such a bin. We claim that $W_{opt} < 1.63$. Define the *density* of an item as the ratio between the weight and the size of the item. Density of an a item is at most $\frac{4/7}{1/3} = 12/7 < 1.72$. Density of b items is at most $\frac{4/7}{37/96} < 1.48$. For items smaller than a items, the density decreases from at most 4/3 for items of class 2 to at most 20/19 for items of classes 18 and 19. Density of d items and large items of classes other than class 1 is at most $96/59 < 1.63$. To prove the claim, we do a case analysis. In what follows, we will for simplicity use β_i to denote both the item and its size. The particular usage will, we hope, be recognizable from context.

(I) Assume β_1 is larger than 59/96, i.e., it is a d item or larger. If β_2 is an a or a b item, we will have $\beta_3 + \ldots + \beta_t \leq 5/96 < 1/19$. So, all other items belong to classes 18 or 19 and their density is at most 20/19. Hence, the total weight of items in the bin will be at most $1 + 4/7 + 5/96 \times 20/19 < 1.63$. If β_2 is smaller than or equal to 1/3, the total size of all items except β_1 is at most 37/96 and their density is at most 4/3. The total weight will be at most $1 + 37/96 \times 4/3 < 1.52$.

(II) Assume β_1 is a c item, i.e., its weight is 0. The total sizes of other items in the bin (all items except β_1) is at most 1/2 and their density is upper bounded by 12/7. Hence, the total weight of items in the bin will be at most $1/2 \times 12/7 < 1$.

(III) Assume β_1 is a b item and β_2 is also a b item or an a item. The total size of other items is at most 27/96 while their density is at most 4/3 (note that they belong to class 2 or higher). Hence, the total weight of items in the bin will be at most $4/7 + 4/7 + 27/96 \times 4/3 < 1.52$. Next, assume β_2 is smaller than a items. The total size of all items except β_1 will be at most 59/96 while their density is at most 4/3. The total weight of items in a bin will be at most $4/7 + 59/96 \times 4/3 < 1.4$.

(IV) Assume β_1 and β_2 are both a items. The total size of all other items is at most 1/3 and their density is at most 4/3 (note that they belong to class 2 or higher). Hence, the total weight of items in the bin will be at most $4/7 + 4/7 + 1/3 \times 4/3 < 1.59$. Next, assume β_2 is smaller than a items; the total size of all items except β_1 will be at most 2/3 while their density is at most 4/3. The total weight of items in a bin will be at most $4/7 + 2/3 \times 4/3 < 1.46$.

(V) Assume β_1 is smaller than 1/3. In this case, the density of all items and consequently their total weight is at most 4/3.

Recall that the number of bins used by RHM to pack a sequence σ is no more than the total weight of items in σ, i.e., $W(\sigma)$. At the same time, the total weight of items in a bin by OPT is at most 1.63, i.e., the cost of OPT is at least $W(\sigma)/1.63$. We conclude that the competitive ratio of RHM is at most 1.63.

Case 2: Assume there is no a_1-bin in the packing. For the number of bins used by RRM to pack σ_{cl_1}, we have:

$$\begin{aligned}\sigma_{cl_1} &= N_d + N_c + N_{ac} + N_{bc} + N_{bb} + N_{aa} + N_{a_2} \\ &\leq n_d + n_c + n_{b_2}/2 + N_{red} + 1\end{aligned}$$

Recall that the algorithm ensures that $N_{red} \leq 3N_{blue}$. By Equation 3.1, we get $N_{red} \leq 3n_a/7 + 3n_{b_1}/7 + 3/7$. Plugging this into the above inequality, we will get:

$$\begin{aligned}\text{RRM}(\sigma_{cl_1}) &\leq n_d + n_c + n_{b_2}/2 + 3n_a/7 + 3n_{b_1}/7 + 3/7 + 1 \\ &< n_d + n_c + n_b/2 + 3n_a/7 + 2.\end{aligned}$$

By Lemma 6, for the total number of bins used by RHM, we will have:

$$\text{RHM}(\sigma) \leq n_X + \sum_{t=2}^{18} \frac{n_t}{t+1} + 20W'/19 + n_d + n_c + n_b/2 + 3n_a/7 + 22 \tag{3.3}$$

Similar to Case 1, we use a weighting technique. For a, b, and c items, the weights are respectively 3/7, 1/2, and 1. The weights of other items are defined similar to Case 1. As Inequality 3.3 suggests, this definition for weights ensures that the total weight of items is an upper bound for the number of bins used by RHM (within an additive constant). As before, we study the maximum weight of a bin in the packing of OPT. Let $\beta_1 \geq \beta_2 \geq \ldots \geq \beta_t$ be the sizes of items in such a bin and W_{opt} be their total weight. We claim that $W_{opt} < 1.63$. Note that the density of d items and large items of classes other than class 1 is at most $96/59 < 1.63$. Density of c, b and a items are respectively upper bounded by 2, 1.3, and 1.29. Density of items smaller than 1/3 which belongs to class $t \geq 2$ is at most $\frac{t+2}{t+1}$. The only exception is the last class (class 19) for which density of small items is at most 20/19. We do a case analysis as before.

(I) Assume β_1 is not a c-item. In this case, the density of all items, and consecutively their total weight, is less than 1.63.

(II) Assume β_1 is a c item and β_2 is a b item. The total size of other items will be at most $11/96 < 1/8$. Hence, these items belong to class 7 or higher and their density is at most 9/8. The total weight of items in the bin will be $1 + 1/2 + 11/96 \times 9/8 \leq 1.62$.

(III) Assume β_1 is a c item and β_2 is an a item. The total size of other items will be at most 1/6. These items belong to class 5 or higher and their density is at most 7/6. Hence, the total weight of items will be at most $1 + 3/7 + 1/6 \times 7/6 < 1.63$.

(IV) Assume β_1 is a c item and β_2 belongs to class 2. The total size of other items will be at most 1/4. Now, if β_3 belongs to class 3, the size of other items will be at most 1-1/2-1/4-1/5=1/20; so they belong to the last class and their density is at most 20/19. The total weight of items will be at most $1 + 1/3 + 1/4 + 1/20 \times 20/19 = 373/228 \approx 1.636$. If β_3 belongs to class 4 or higher, its density will be at most 6/5, and the total weight of items in the bin will be at most $1 + 1/3 + 1/4 \times 6/5 < 1.634$.

(V) Assume β_1 is a c item and β_2 belongs to class 3 or higher; so, all items except β_1 have a density of at most 5/4. The total weight of items will be at most $1 + 1/2 \times 5/4 = 1.625$.

To summarize, the total weight of items in a bin in OPT's packing is at most 373/228. This implies that number of bins used by OPT to pack σ is at least $228/373 \times W(\sigma)$. Recall that the number of bins used by RHM for is upper bounded by $W(\sigma)$ (within an additive constant). We conclude that the competitive ratio of RHM is at most 373/228.

□

3.3.2 Average-Case Analysis

We show that the average-case performance of RHM is as good as BF and HM. As before, we assume the item sizes are distributed uniformly in the interval $[0, 1]$. Except the following lemma, other aspects of the proof are similar to those in Section 3.2.2.

Lemma 7. *For any instance σ of the bin packing problem in which items are in the range $(1/3, 2/3]$, the number of bins used by* RRM *to pack σ is no more than that of* MATCHING BEST FIT *(*MBF*).*

Proof. The key observation is that RRM uses the BF strategy to place a small item x in a bin which includes a large item. Note that small items are a and b items in the RRM algorithm. Only if such a bin does not exist, RRM deviates from the BF strategy. Let S^t_{RRM} and S^t_{MBF} respectively denote the set of large items which are not accompanied by a small item in the packings maintained by RRM and MBF (respectively) after placing the first t items ($0 \le t \le n$, where n is the length of σ); we refer to these sets as *single-set*s of the algorithms. We claim that for all values of t, the single-set of RRM is a subset of that of MBF, i.e., $S^t_{RRM} \subset S^t_{MBF}$. We prove this by induction. Note that for $t = 0$ both single-sets are empty and the base case holds. Assume $S^t_{RRM} \subset S^t_{MBF}$ for some $t > 0$ and let x denote the $(t+1)$th item in σ. If x is a large item, MBF opens a bin for x, and x will be included in the single-set for MBF; x may or may not be added to the single-set of RRM (it will not be added if it is a c item and there are a_1 bins in the packing). Regardless, we will have $S^{t+1}_{RRM} \subset S^{t+1}_{MBF}$. Next, assume x is a small item. MBF and RRM both use the BF strategy to place x in one of the bins in S^t_{MBF} and S^t_{RRM}. If such a bin does not exist for MBF, by induction hypothesis, it will not exist for RRM and the induction statement holds. Next, assume MBF places x in a bin $B \in S^t_{MBF}$; so, B will be removed from single-set of MBF, i.e., we have $S^{t+1}_{MBF} = S^t_{MBF} - \{B\}$. If $B \notin S^t_{RRM}$, the induction statement holds because S^{t+1}_{RRM} will be a subset of S^t_{RRM} which is a indeed a subset of S^{t+1}_{MBF} (a non-common items is removed from S^t_{MBF}). IF $B \in S^t_{RRM}$, RRM places x in B; this is because, similar to MBF, RRM uses the BF strategy to place small items in bins which include large items. Consecutively, we have $S^n_{RRM} \subset S^n_{MBF}$.

The number of bins used by MBF to pack σ is $n_{small} + |S^n_{MBF}|$ in which n_{small} is the number of small items in σ. This is because MBF opens a new bin for each small item. In the final packing of RRM,

the number of bins which include small items is no more than n_{small}. Other bins in the packing are associated with items in S^n_{RRM}; since the single-set of RRM is a subset of that of single-set of MBF, we have $|S^n_{RRM}| \leq |S^n_{MBF}|$. Hence, in total, the number of bins in the packing of RRM is no more than that of MBF. □

Theorem 4. *Let σ be a sequence of length n in which item sizes are selected uniformly at random from $(0, 1]$. The expected wasted space of* RHM *for packing σ is $\Theta(\sqrt{n}\lg^{3/4} n)$.*

Proof. Let σ^- be a copy of σ in which those items which are placed in mature bins are removed. Also, let $\sigma_2^-, \ldots, \sigma_{19}^-$ be the subsequences of σ^- formed by items belonging to different classes of HM. We have

$$\text{HM}(\sigma) = \text{RRM}(\sigma_1) + \sum_{t=2}^{19} \text{ROM}(\sigma_t^-) \leq \sum_{t=1}^{19} \text{MBF}(\sigma_t^-) \leq \sum_{t=1}^{19} \text{MBF}(\sigma_t)$$

The second-to-last inequality comes from Lemmas 4 and 7 and the last inequality comes from Lemma 5. Consequently, by Lemma 3, we have:

$$E[\text{HM}(\sigma)] \leq \sum_{t=1}^{19} \left(n_t/2 + \Theta(\sqrt{n_t}\lg^{3/4} n_t) \right) = \frac{n}{2} + \Theta(\sqrt{n}\lg^{3/4} n)$$

The expected value of $s(\sigma)$, the total size of items in σ, is $n/2$. Consequently, $E[\text{HM}(\sigma) - s(\sigma)] = \Theta(\sqrt{n}\lg^{3/4} n)$ which completes the proof. □

3.4 Experimental Evaluation

The results of the previous sections indicate that HM and RHM have similar average-case performance as BF if we assume a uniform, continuous distribution for item sizes. In this section, we expand the range of distributions beyond continuous uniform distribution (for which the algorithm were developed) to further observe the performance of these algorithms on sequences which follow discrete distributions. In doing so, we experimentally compare HM and RHM against classic bin packing algorithms. Table 3.2 gives details of the datasets that we generated for our experiments. In all cases, the item sizes are randomly and independently taken from a subset of the set $\{1, 2, \ldots, E\}$ of integers; this subset defines the *range* of the items in the set-instance. Here, E indicates the capacity of the bins. Typically, we have $E = 1000$, and the range is $[1, 1000)$. In what follows, we briefly describe the set-instances that we used in our experiments.

- Discrete Uniform Distribution (DU) sequences: We test the algorithms on *discrete* uniform distributions. In these distribution, the probability of an item having size x is the same for all values of $x \in \{1, 2, \ldots, E\}$. It is known that average-case behavior of bin packing algorithms can be different under discrete and continuous distributions. In particular, the wasted space of OPT in these distributions is expected to be constant [48]. The bin packing problem is extensively studied under discrete uniform distributions (see, e.g., [48, 58, 6]).
- NORMAL and POISSON sequences: Both Normal and Poisson distributions have been previously studied for generating bin packing sequences (see, e.g., [73, 136]).

Set-instance	Distribution	E	Range
DU0	Uniform	100	[1,E)
DU1	Uniform	500	[1,E)
DU2	Uniform	1,000	[1,E)
DU3	Uniform	1,000	[1,E/2)
DU4	Uniform	1,000	[1,E/10)
NORMAL	Normal ($\mu = E/2$, $\sigma = E/6$)	1,000	[1,E)
POISSON	Poisson ($\lambda = E/3$)	1,000	[1,E)
ZIPF1	Zipfian ($\theta = 1/2$)	1,000	[1,E)
ZIPF2	Zipfian ($\theta = 1/3$)	1,000	[1,E)
SORTED	Uniform, sorted decreasing	1,000	[1,E)
WD1	Weibull ($k = 0.454$, $\lambda = E/2$)	1,000	[1,E)
WD2	Weibull ($k = 1.044$, $\lambda = E/2$)	1,000	[1,E)
BPSD1	BPS distribution ($s = 100$)	1,000	[1,E/2)
BPSD2	BPS distribution ($s = 100$)	1,000	[1,E/4)

Table 3.2: . The distributions used to create set-instances to compare algorithms.

- Zipfian Distribution (ZD) sequences: In sequences that follow the Zipfian distribution, item sizes follow the power-law, i.e., a large number of items are pretty small while a small number of items are quite large. The distribution has a parameter θ ($0 < \theta < 1$) that indicates how skewed the distribution is. Bin packing sequences with Zipfian distribution are considered in the experiments in [19].

- Sorted-Decreasing (SORTED) sequences: To create an instance of this family, we take a sequence of uniformly random-sized items and sort it in the decreasing order of item sizes. This way, we can compare the performance of *offline* versions of the algorithms (where sequences are sorted in decreasing order before being packed in an online manner).

- Weibull Distribution (WD) sequences: It is known that the Weibull distribution can be used to model real-world bin packing benchmarks [43]. This distribution has a shape parameter λ and a scale parameter k. The parameters considered here are among the ones suggested in [43].

- Bounded Probability Sampled Distributions (BPSD) sequences: To get these sequences, a random distribution is generated as follows. Given a parameter s, we select s random numbers in a given range and assign random weights to them. The probability associated with an item in the distribution is proportional to its weight. These sequences were first introduced by Degraeve and Peeters [61] and later used in the experiments of Applegate et al. [19].

For each of the indicated set-instances, we create 1000 random sequences of length 10^6 and compute the average number of bins used by different algorithms for packing these sequences. Beside the Any-Fit and Harmonic family of algorithms, we also consider *Sum-of-Squares (*Ss*)* algorithm which performs well for discrete distributions [58, 56, 57]. To place an item into a partial packing P, Ss defines *sum of squares* of P, denoted by $ss(P)$, as $\sum_h N_P(h)^2$; here $N_P(h)$ denotes the number of bins with level h in P. To place an item x, Ss places x into an existing bin, or opens a new bin for x, so as to yield the minimum possible

	DU0	DU1	DU2	DU3	DU4	NORMAL	POISSON	ZIPF1	ZIPF2	SORTED	WD1	WD2	BPSD1	BPSD2
NF	664,996	666,327	666,519	298,400	51,689	698,725	405,401	514,464	433,172	644,744	324,147	458,133	299,898	137,011
WF	584,585	585,536	585,680	275,235	50,907	606,569	371,060	448,165	376,331	500,287	281,821	397,966	276,922	131,691
HA	642,759	644,296	644,718	289,682	51,592	713,657	414,238	502,309	425,421	644,752	320,099	454,963	290,945	135,667
RFF	643,428	644,517	644,480	289,200	50,004	719,629	454,676	500,605	423,017	644,349	317,709	452,763	291,015	126,103
RHA	646,366	648,066	648,436	297,125	51,592	719,921	448,736	505,943	428,804	648,469	322,767	459,280	298,372	135,667
OM	500,988	501,193	501,279	298,400	51,689	500,677	405,401	403,962	346,265	500,429	263,364	368,887	299,898	137,011
FF	502,042	502,984	503,209	251,250	50,004	502,648	343,865	392,609	333,397	500,281	251,093	352,958	254,444	126,103
BF	500,884	501,081	501,171	251,137	50,004	500,642	343,822	392,235	333,236	500,281	251,067	352,740	254,291	126,090
HM	**507,016**	**502,553**	**502,251**	**251,014**	**50,009**	**501,142**	**351,595**	**392,358**	**333,416**	**501,250**	**251,403**	**352,987**	**254,333**	**126,086**
RHM	**507,016**	**502,553**	**502,251**	**260,381**	**50,009**	**501,142**	**392,812**	**392,357**	**333,416**	**501,250**	**251,403**	**352,981**	**264,203**	**126,086**
SS	501,083	502,374	503,374	250,003	50,000	503,135	333,802	390,271	331,938	503,031	251,019	350,568	251,250	125,641
OPT	500,005	499,993	500,009	249,992	49,999	499,519	332,999	390,198	331,915	500,008	251,015	350,533	251,219	125,638

(a) The average number of bins used by online algorithms.

	DU0	DU1	DU2	DU3	DU4	NORMAL	POISSON	ZIPF1	ZIPF2	SORTED	WD1	WD2	BPSD1	BPSD2
NF	1.330	1.333	1.333	1.194	1.034	1.399	1.217	1.318	1.305	1.289	1.291	1.307	1.194	1.091
WF	1.169	1.171	1.171	1.101	1.018	1.214	1.114	1.149	1.134	1.001	1.123	1.135	1.102	1.048
HA	1.286	1.289	1.289	1.159	1.032	1.429	1.244	1.287	1.282	1.289	1.275	1.298	1.158	1.080
RFF	1.287	1.289	1.289	1.157	1.000	1.441	1.365	1.283	1.274	1.289	1.266	1.292	1.158	1.004
RHA	1.293	1.296	1.297	1.189	1.032	1.441	1.348	1.297	1.292	1.297	1.286	1.310	1.188	1.080
OM	1.002	1.002	1.003	1.194	1.034	1.002	1.217	1.035	1.043	1.001	1.049	1.052	1.194	1.091
FF	1.004	1.006	1.006	1.005	1.000	1.006	1.033	1.006	1.004	1.001	1.000	1.007	1.013	1.004
BF	1.002	1.002	1.002	1.005	1.000	1.002	1.033	1.005	1.004	1.001	1.000	1.006	1.012	1.004
HM	**1.014**	**1.005**	**1.004**	**1.004**	**1.000**	**1.003**	**1.056**	**1.006**	**1.005**	**1.002**	**1.002**	**1.007**	**1.012**	**1.004**
RHM	**1.014**	**1.005**	**1.004**	**1.042**	**1.000**	**1.003**	**1.180**	**1.006**	**1.005**	**1.002**	**1.002**	**1.007**	**1.052**	**1.004**
SS	1.002	1.005	1.007	1.000	1.000	1.007	1.002	1.000	1.000	1.006	1.000	1.000	1.000	1.000

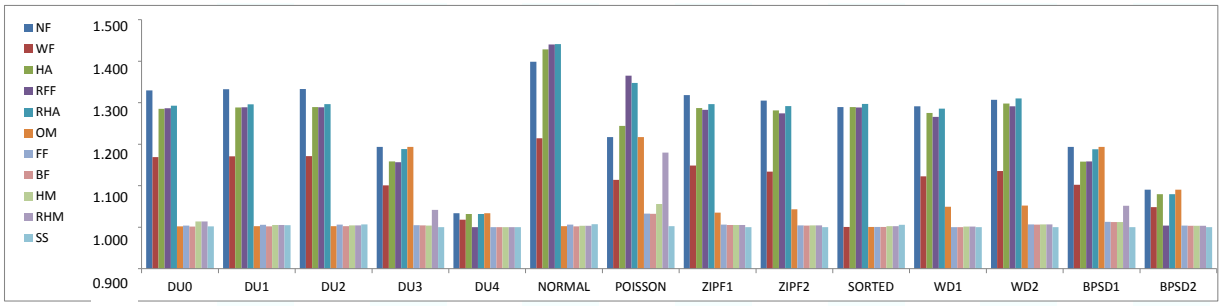
(b) The experimental average ratio online algorithms.

Figure 3.3: Average performance of online bin packing algorithms for different set-instances. The indicated numbers represent the average number of bins used by the algorithms (a) and the experimental average ratios (b). In most cases, HM (and RHM) performs significantly better than other Harmonic-based algorithms.

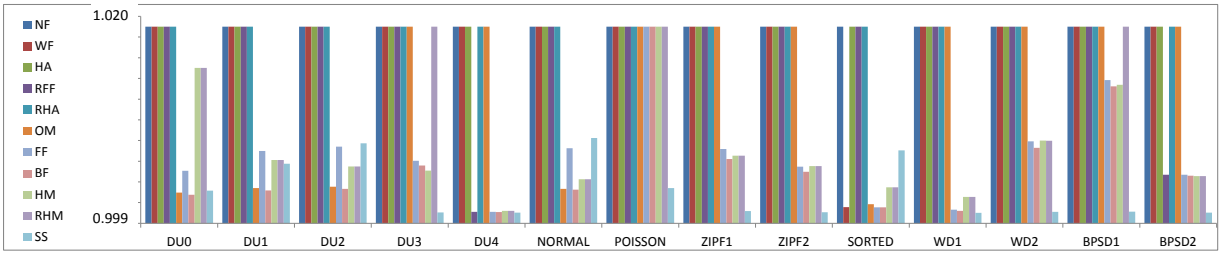
value of $ss(P')$ for the resulting packing P'. Note that SS is not well-defined for the continuous version of the bin packing problem. Csirik et al. [57] proved that for any discrete distribution in which the optimal expected waste is sub-linear, SS also has sub-linear expected waste. In particular, for those distributions where the optimal expected waste is constant, SS has an expected waste of at most $O(\lg n)$. We note that the competitive ratio of SS is at least 2 and at most 2.77 [56] which is worse than most online algorithms.

Figure 3.3a shows the average number of bins used by the classic bin packing algorithms, as well as HM and RHM, to pack the above set-instances. For algorithms that classify items by their sizes (e.g., HA and HM), the number of classes (the value of K) is set to 20. These results indicate that, in general, HM and RHM perform significantly better than other members of the Harmonic family. At the same time, they have comparable performance with BF and FF.

We also compute the *experimental average ratio* of an algorithm as the ratio between the observed expected number of bins used by the algorithm and that of OPT. In doing so, we estimate the cost of OPT as the total size of items. Figure 3.3b shows the experimental average ratio of the considered algorithms. Note that the ratios associated with HM and RHM are close to 1 and much smaller than other members of the Harmonic family (e.g., HA and RH). These results are more visible in Figure 3.4 which shows the bar chart for experimental average ratio of different online algorithms. It can be seen that HM and RHM, along



(a) The vertical scale starts at 0.9.



(b) The vertical scale starts at 0.999 and goes only upto 1.02

.

Figure 3.4: The bar chart associated with the experimental average ratios of online bin packing algorithms. To make the results more visible, the vertical scale is changed to start at 0.9 in (a). The ratios associated with HM and RHM are visibly smaller than Harmonic-based algorithms. To compare these with other algorithms which have ratios close to 1, the vertical scale is changed in (b) to start at 0.9 and go only upto 1.02.

with BF, FF, and SS algorithms, have a significant advantage over other algorithms. To compare these algorithms among themselves, we magnify the bar char in Figure 3.4b so that the scale of the y-coordinate starts at 0.999 and goes only up to 1.02. While the result indicate a slight advantage for SS and BF, there are distributions for which HM outperforms BF or SS. In what follows, we briefly review the results for different distributions.

Comparing the number of bins used by HM and RHM for DU0, DU1, and DU2, we observe that their relative performance improves when the size of the bins (i.e., E) increases. For small value of $E = 100$ (UD0), these algorithms are slightly worse than FF. However, as E increases to 1000 (UD2), the algorithms perform better than FF and converge to BF. This is in accordance with the results in Sections 3.2.2 and 3.3.2 which imply that, for continues uniform distribution, the expected waste of HM and RHM converge to that of BF. Note that as E goes to infinity, the discrete distribution approaches a continuous one.

For symmetric distributions, where items of sizes x and $E - x$ appear with the same probability, the expected number of bin used by HM and RHM are equal. A difference between the packings of HM and RHM happens when a number of small items of the first class (items of type a in RHM) appear before any large item of the same class (an item of type c). In these cases, RHM reserves some bins for subsequent large items (by declaring the bins as being blue). For symmetric distributions, however, it is unlikely that

many small items appear before the next large item. Consequently, the average number of bins used by HM and RHM are the same for symmetric sequences. On the other hand, for asymmetric sequences where small items are more likely to appear, e.g., DU3 and POISSON, HM has a visible advantage over RHM. In these sequences, there is no reason to reserve bins for the large items since they are unlikely to appear.

Finally, we note that for SORTED, the number of bins used by HM and RHM are comparable to those of BF and FF. This implies that, on average, the offline versions of these algorithms are comparable with the well-known First-Fit-Decreasing and Best-Fit-Decreasing algorithms. It remains open whether the same statement holds for the worst-case performance of these algorithms.

3.5 Remarks

HM and RHM can be seen as variants of HARMONIC and REFINED HARMONIC algorithms in which small and large items are carefully matched in order to improve the average performance while preserving the worst-case performance. We believe that the same approach can be applied to improve the average performance of other Super Harmonic algorithms and in particular that of HARMONIC++ (which is currently the best online bin packing algorithm regarding the competitive ratio). Given the complicated nature of these algorithms, modifying them involves a detailed analysis which we leave as a future work.

While HM and RHM have better competitive ratio than BF, they are not expected to be preferred in practical scenarios. Recall that the worst-case sequences are unlikely to appear in practice, and regarding the average-case performance, HM and RHM have no significant advantage over BF (although they have comparable performance).

It is possible to study the performance of the introduced algorithms under the relative worst order analysis (see Section 1.1 for a review). It is known that under relative worst order analysis, FIRST FIT is no worse than any Any Fit algorithm (and in particular BEST FIT) [40]. Also, HARMONIC algorithm is not comparable to FF for sequences which include very small items. However, when all items are larger than $\frac{1}{K+1}$ (K is the parameter of the HARMONIC algorithm), HARMONIC is better than FF by a factor of 6/5 [40]. Applying Theorem 1, we conclude that when all items are larger than $\frac{1}{K+2}$, HARMONIC MATCH with parameter K is strictly better than FF and BF under the relative worst order analysis. This provides another theoretical evidence for the advantage of HARMONIC MATCH over BF and FF.

Several online bin packing algorithms, such as BF and FF, have the undesired property that removing an item might increase the number of bins used the algorithm [116]. This 'anomalous' behavior results in an unstable algorithm which is harder to analyze. As mentioned earlier, an algorithm is called monotone if removing an item does not increase its cost. It is not clear whether HM and RHM are monotone; however, a slight twist in HM results in a monotone algorithm. Consider a modified algorithm, called MODIFIED HARMONIC MATCH (MHM), that works similar to HM except that it does not maintain mature bins, i.e., it closes a bin as soon as it becomes mature. It is not hard to see that MHM is a monotone algorithm. At the same time, the results related to the worst-case and average-case performance of HM hold also for MHM (Corollary 1 and Theorem 2). However, MHM performs slightly worse than HM on discrete distributions evaluated in Section 3.4. We leave further analysis of monotonous behavior of this algorithm as a future work.

Part III

Advice Model of Computation

Chapter 4

Online Bin Packing with Advice

In this chapter, we consider the online bin packing problem under the advice model of complexity. Recall that under the advice model, the 'online constraint' is relaxed and an algorithm receives partial information about the future requests. We investigate the trade-off between the amount of advice and the quality of the resulting algorithms. In doing so, we provide tight bounds for the amount of advice that is required and sufficient for an algorithm to achieve an optimal packing. We also introduce a simple algorithm that achieves a competitive ratio of $3/2$ when provided with a logarithmic number of bits of advice. We introduce another algorithm that achieves a competitive ratio of $4/3 + \varepsilon$ provided with a linear number of bits of advice. Finally, we provide a lower bound argument that implies that advice of linear size is required for an algorithm to achieve a competitive ratio better than $9/8$.

4.1 Introduction

Recall that in the bin packing problem the goal is to pack a given sequence of items into a minimum number of bins with fixed and equal capacities. We consider the bin packing problem under the advice-on-tape model. Recall that under the advice model, the advice bits are written on a tape which is available to an online algorithm since the beginning. We formally define the bin packing problem with advice as follows, based on the definition of the advice model in [36]:

Definition 4. *In the* online bin packing problem with advice*, the input is a sequence of items* $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$*, revealed to the algorithm in an online manner. We have* $0 < s(\sigma_i) \leq 1$*, where* $s(\sigma_i)$ *is the size of* σ_i*. The goal is to pack these items in the minimum number of bins of unit size. At time step* t*, an online algorithm should pack item* σ_t *into a bin. The decision of the algorithm to select the target bin is a function of* $\Phi, \sigma_1, \ldots, \sigma_{t-1}$*, where* Φ *is the content of the advice tape. An algorithm* $\mathbb{A}$ *is* c-competitive with advice complexity $s(n)$ *if there exists a constant* c_0 *such that, for all* n *and for all input sequences* σ *of length at most* n*, there exists some advice* Φ *such that* $\mathbb{A}(\sigma) \leq c \cdot \textsc{Opt}(\sigma) + c_0$*, and at most the first* $s(n)$ *bits of* Φ *have been accessed by the algorithm. If* $c = 1$ *and* $c_0 = 0$*, then* $\mathbb{A}$ *is* optimal*.*

We answer several questions about the advice complexity of the online bin packing problem. First, we study the number of advice bits that are required and sufficient to achieve an optimal solution for a

sequence of length n. We consider two different settings of the problem. When there is no restriction on the number of distinct items or their sizes, we show that $n\lceil \lg \textsc{Opt}(\sigma)\rceil$ bits of advice are sufficient to achieve an optimal solution, where $\textsc{Opt}(\sigma)$ is the number of bins in an optimal packing. We also prove that at least $(n - 2\,\textsc{Opt}(\sigma))\lg \textsc{Opt}(\sigma)$ bits of advice are required to achieve an optimal solution. When there are m distinct items in the sequence, we prove that at least $(m-3)\lg n - 2m\lg m$ bits of advice are required to achieve an optimal solution. If m is a constant, there is a polynomial time online algorithm that receives $m\lg n + o(\lg n)$ bits of advice and achieves an optimal solution. We also show that, even if m is not bounded, there is a polynomial time online algorithm that receives $m\lceil \lg(n+1)\rceil + o(\lg n)$ bits of advice and achieves a packing with $(1+\varepsilon)\,\textsc{Opt}(\sigma) + 1$ bins.

We also study a relevant question that asks how many bits of advice are required to perform strictly better than all online algorithms. We bound this by providing an algorithm which receives $\lg n + o(\lg n)$ bits of advice and achieves a competitive ratio of $3/2$. Recall that any online bin packing algorithm has a competitive ratio at least 1.54037 [22]. Hence, our algorithm outperforms all online algorithms.

Moreover, we introduce an algorithm that receives $2n + o(n)$ bits of advice and achieves a competitive ratio of $4/3 + \varepsilon$, for any fixed value of $\varepsilon > 0$. We also prove a lower bound that implies that a linear number of bits of advice are required to achieve a competitive ratio of $9/8 - \delta$ for any fixed value of $\delta > 0$.

Under the advice-on-tape model, we require a mechanism to infer how many bits of advice the algorithm should read at each time step. This could be implicitly derived during the execution of the algorithm or explicitly encoded in the advice string itself. For example, we may use a *self-delimited* encoding as used in [36], in which the value of a non-negative integer X is encoded by writing the value of $\lceil \lg(\lceil \lg(X+1)\rceil + 1)\rceil$[1] in unary (a string of 1's followed by a zero), the value of $\lceil \lg(X+1)\rceil$ in binary, and the value of X in binary. These codes respectively require $\lceil \lg(\lceil \lg(X+1)\rceil + 1)\rceil + 1$, $\lceil \lg(\lceil \lg(X+1)\rceil + 1)\rceil$, and $\lceil \lg(X+1)\rceil$ bits. Thus, the self-delimited encoding of X requires

$$e(X) = \lceil \lg(X+1)\rceil + 2\lceil \lg(\lceil \lg(X+1)\rceil + 1)\rceil + 1$$

bits. The existence of self-delimited encodings at the beginning of the tape usually adds a lower-order term to the number of advice bits required by an algorithm.

4.2 Optimal Algorithms with Advice

In this section we study the amount of advice required to achieve an optimal solution. We first investigate the theoretical setting in which there is no restriction on the number of distinct items or on their sizes. We observe that there is a simple algorithm that receives $n\lceil \lg \textsc{Opt}(\sigma)\rceil$ bits of advice and achieves an optimal solution. Such an algorithm basically reads $\lceil \lg \textsc{Opt}(\sigma)\rceil$ bits for each item, encoding the index of the bin that includes the item in an optimal packing. We show that the upper bound given by this algorithm is tight up to lower order terms, when $n - 2\,\textsc{Opt}(\sigma) = \Theta(n)$.

Theorem 5. *To achieve an optimal packing for a sequence of length n and optimal cost $\textsc{Opt}(\sigma)$, it is sufficient to receive $n\lceil \lg \textsc{Opt}(\sigma)\rceil$ bits of advice. Moreover, any deterministic online algorithm requires at least $(n - 2\,\textsc{Opt}(\sigma))\lg \textsc{Opt}(\sigma)$ bits of advice to achieve an optimal packing.*

[1] Throughout the thesis, we use $\lg x$ to denote $\log_2(x)$.

Proof. *Upper Bound:* Consider an offline oracle that knows an optimal packing[2]. Note that such an oracle has unbounded computational power. This oracle simply writes on the advice tape, for each item x, except for the last two, the index of the bin in an optimal packing that x is packed in. To pack any item x, the online algorithm simply reads the index of the bin that x should be packed in and packs x accordingly. For the last two items, the algorithm simply uses Best-Fit. Since the packing is the same as one for an optimal algorithm up to that point, if it is impossible to fit both of the remaining items in the bins already used, Best-Fit will ensure that at least one fits if that is possible. If both of the remaining items fit in the same already open bin, it is fine to put the first one of the last two items anywhere it fits, since there will still be space remaining for the last. If both of the remaining items fit in open bins, but should be in different bins, using Best-Fit will ensure that they are both placed there. This requires $\lceil \lg \textsc{Opt}(\sigma) \rceil$ bits of advice per item which sums up to $(n-2)\lceil \lg \textsc{Opt}(\sigma) \rceil$ bits of advice. The algorithm should also know the value of $X = \lceil \lg \textsc{Opt}(\sigma) \rceil$ in order to read the appropriate number of bits on each request. This can be done by encoding X in unary and terminating with a zero (or self-delimited encoding of X as indicated in Section 1.2). This uses no more than $2\lceil \lg \textsc{Opt}(\sigma) \rceil$ bits. Consequently the number of advice bits used by the algorithm is $n\lceil \lg \textsc{Opt}(\sigma) \rceil$ as stated by the theorem.

Lower Bound: Consider a set $S = \{\sigma^1, \ldots, \sigma^N\}$ of sequences, so that each σ^r has length n for $1 \leq r \leq N$. Let $1 \leq k \leq n-1$. The sizes of items in each sequence σ^r in the set has the form

$$\left\langle \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \ldots, \frac{1}{2^{n-k+1}}, u_1^r, u_2^r, \ldots, u_k^r \right\rangle$$

in which $u_1^r, \ldots, u_k^r$ are defined as follows. Consider a set V of vectors of the form

$$V^r = (v_1^r = 1, v_2^r = 2, \ldots, v_k^r = k, v_{k+1}^r, v_{k+2}^r, \ldots, v_{n-k}^r)$$

such that each $v_h^r \in \{1, \ldots, k\}$ for $k+1 \leq h \leq n-k$.

For example, when $n = 8$ and $k = 3$, the vector $(1, 2, 3, 2, 1)$ is a vector in V.

We associate with each vector $V^r \in V$ a sequence $\sigma^r \in S$. For a vector $V^r \in V$ and bin j, define $u_j^r = 1 - \sum_{\substack{1 \leq i \leq n-k \\ v_i^r = j}} a_i$, where a_i is the ith item in the sequence σ^r, i.e., $a_i = \frac{1}{2^{i+1}}$. Note that all u_js are strictly larger than 0.5. Clearly, $\textsc{Opt}(\sigma_r) = k$ for all r. We refer to the first $n - \textsc{Opt}(\sigma)$ items as *small* items and the last $\textsc{Opt}(\sigma)$ items as *large* items.

For example, assume $n = 8$ and $\textsc{Opt}(\sigma) = 3$. For a vector $V^r = (1, 2, 3, 2, 1)$, we have $u_1^r = 1 - (\frac{1}{4} + \frac{1}{64}) = 0.734375$, $u_2^r = 1 - (\frac{1}{8} + \frac{1}{32}) = 0.84375$, and $u_3^r = 1 - \frac{1}{16} = 0.9375$. Hence, the sequence σ^r associated with V^r is $\langle \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}, 0.734375, 0.84375, 0.9375 \rangle$.

In fact, V^r indicates in which bin each of the first $n - \textsc{Opt}(\sigma)$ items of σ^r should be packed, and at the end, u_j^r fills the empty space of the jth bin to capacity to achieve an optimal packing P for a given sequence (it is optimal since all bins are fully packed). We claim that P is the unique optimal packing.

[2]When studying the number of advice bits for achieving an optimal solution, we are merely measuring the information content of online problems. In this sense, similar to most related works, we do not make any assumption on the computational power of the offline oracle that generates advice.

Suppose there is another optimal packing P'. Observe that each bin includes at most one large item, and indeed exactly one since we assume it is also optimal. Let $a_i (1 \leq i \leq n - \textsc{Opt}(\sigma))$ be the first item which is packed in some other bin in P' than the one prescribed by P. Consider the bin B that a_i is packed into in P. This bin cannot be fully packed in P' since a_i is strictly larger than the total size of all remaining small items, i.e., even if we put all of them in the empty space of a_i, there is still some empty space in B. As a result P' cannot be optimal. Hence, there is unique solution for packing each sequence in the set S.

Note that there are $N = \textsc{Opt}(\sigma)^{n-2\,\textsc{Opt}(\sigma)}$ sequences S. We claim that these sequences need separate advice strings. Suppose otherwise, and let $\sigma^r, \sigma^{r'} \in S$ $(r \neq r')$ be two different sequences with the same advice string. Note that the first $n - \textsc{Opt}(\sigma)$ items in these sequences are the same. Since the online algorithm performs deterministically and we assume it receives the same advice for both σ^r and $\sigma^{r'}$, the partial packings of the algorithms after packing the first $n - \textsc{Opt}(\sigma)$ items are the same for both sequences. However, this implies that the final packing of the algorithm is different from the optimal packing prescribed by $V^{r''}$ for at least one of the sequences. As discussed, such a packing is the unique optimal packing and deviating from that increases the number of bins used by the algorithm by at least one unit. As a result, the algorithm performs non-optimally for at least one of σ^r or $\sigma^{r'}$. We conclude that the sequences in the set S need separate advice strings. Since there are $N = \textsc{Opt}(\sigma)^{n-2\,\textsc{Opt}(\sigma)}$ sequences in S, at least $\lg(\textsc{Opt}(\sigma)^{n-2\,\textsc{Opt}(\sigma)}) = (n - 2\,\textsc{Opt}(\sigma)) \lg \textsc{Opt}(\sigma)$ bits of advice are required to get that many distinct advice strings. □

Next, we consider a more realistic scenario where there are $m = o(n)$ distinct item sizes and the values of these item sizes are known to the algorithm. Assume that the advice tape specifies the number of items of each size. If we are not concerned about the running time of the online algorithm, there is enough information to obtain an optimal solution. If we are concerned, we can use known results for solving the offline problem [32, 60, 141]. We formalize this in the following two lemmas.

Lemma 8 ([80])**.** *Consider the restriction of the bin packing problem to instances in which the number of distinct item sizes is a constant non-negative integer m. There is a polynomial time algorithm that optimally solves this restricted problem.*

If there are more than a constant number of distinct items sizes, we can solve the problem almost optimally.

Lemma 9 ([60, 141])**.** *There is a polynomial algorithm for the bin packing problem which opens at most $(1 + \varepsilon)\,\textsc{Opt}(\sigma) + 1$ bin, in which ε is any small but fixed value.*

We use the above results to obtain the following:

Theorem 6. *Consider the online bin packing problem in which there are m distinct item sizes, and the sizes are assumed to be known. If m is a constant, there is a (polynomial time) optimal online algorithm that receives $m \lg n + o(\lg n)$ bits of advice. If m is not a constant, there is a (polynomial time) online algorithm that reads $m\lceil \lg(n + 1)\rceil + o(\lg n)$ bits of advice and achieves an almost optimal packing with at most $(1 + \varepsilon)\,\textsc{Opt}(\sigma) + 1$ bins, for any small but fixed value of ε.*

Proof. The offline oracle simply encodes the input sequence, considered as a multi-set, in $m\lceil \lg(n + 1)\rceil$ bits of advice. In order to do that, it writes the number of occurrences of each of the m distinct items on the tape. The online algorithm uses the algorithms of Lemma 8 (for constant values of m) or that of

Lemma 9 (for non-constant m) to compute an (almost) optimal packing. Then it packs the items in an online manner according to such an (almost) optimal packing. The algorithms reads frequencies of items in chunks of $X = \lceil \lg(n+1) \rceil$ bits and consequently needs to know the value of X. So, we add self-delimited encodings of X at the beginning of the tape using $e(X)$ bits. The number of advice bits used by the algorithm is thus $m\lceil \lg(n+1) \rceil + O(\lg \lg n)$, which is $m\lceil \lg(n+1) \rceil + o(\lg n)$ as $m = o(n)$. □

We show that the above upper bound is asymptotically tight. We start with the following simple lemma.

Lemma 10. *Consider the equation $x_1 + 2x_2 + \ldots + \alpha x_\alpha = X$ in which the x_is ($i \leq \alpha$) and X are non-negative integers. If X is sufficiently large, then this equation has at least $\left(1 + \frac{2X}{\alpha(\alpha+1)}\right)^{\alpha-1}$ solutions.*

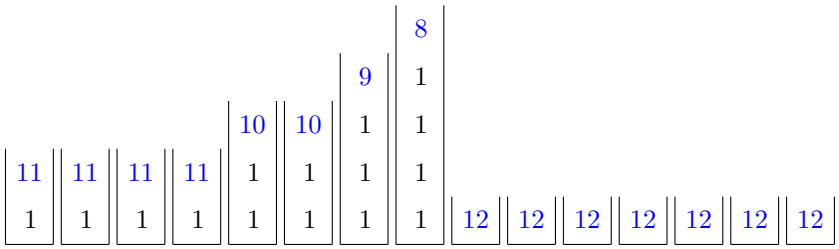
Proof. Define $A = \sum_{i=1}^{\alpha} i$. Assign arbitrary values in the range $[0..X/A]$ to all x_is for $2 \leq i \leq \alpha$ (for simplicity assume X/A is an integer). There are $(1 + X/A)^{\alpha-1}$ different such assignments. Any of these assignments defines a valid solution for the equation since by definition of A we have $\sum_{i=2}^{\alpha} i x_i \leq X$, and we can assign $x_1 = X - \sum_{i=2}^{\alpha} i x_i$. Replacing A with $\alpha(\alpha+1)/2$ completes the proof. □

Theorem 7. *At least $(m-3)\lg n - 2m \lg m$ bits of advice are required to achieve an optimal solution for the online bin packing problem on sequences of length n with m distinct item sizes.*

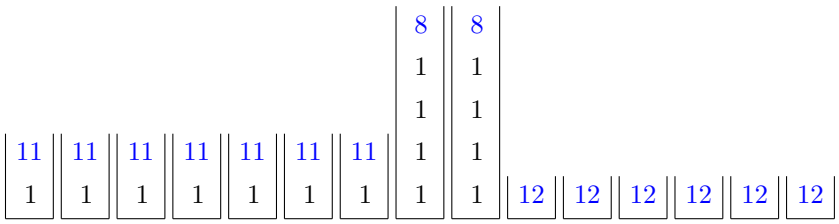
Proof. We define a family of sequences of length n and containing m distinct item sizes and show that the sequences in this family need separate advice strings to be optimally served by an online algorithm. To define the family, we fix m item sizes as being $\{\frac{1}{2m}, \frac{m+2}{2m}, \frac{m+3}{2m}, \ldots, \frac{2m-1}{2m}, 1\}$. To simplify the argument, we scale up the sizes of bins and items by a factor of $2m$. So, we assume the item sizes are $\{1, m+2, m+3 \ldots, 2m-1, 2m\}$, and the bins have capacity $2m$. Each sequence in the family starts with $n/2$ items of size 1. Consider any packing of these items in which all bins have level at most equal to $m-2$. Such a packing includes a_1 bins of level 1 (one item of size 1 in each), a_2 bins of level 2 (two items of size 1 in each), etc., such that the a_is are non-negative integers and $a_1 + 2a_2 + \ldots + (m-2)a_{m-2} = n/2$. By Lemma 10, there are at least $\left(1 + \frac{n}{(m-1)(m-2)}\right)^{m-3}$ distinct packings with the desired property. For any of these packings, we define a sequence in our family. Such a sequence starts with $n/2$ items of size 1 and is followed by another $n/2$ items. Let B denote the number of bins in a given packing of the first $n/2$ items, so that $B \leq n/2$. The sequence associated with the packing is followed by B items of size larger than $m+1$ which completely *fit* these bins (in non-increasing order of their sizes). Finally, we include another $n/2 - B$ items of size $2m$ in the sequence to achieve a sequence of length n.

We claim that any of the sequences in the family has a unique optimal packing of size $n/2$. This is because there are exactly $n/2$ *large* items of size strictly greater than m (more than half the capacity of the bin), and the other $n/2$ items have *small* size 1 (which fit the empty space of all bins). So each bin is fully packed with one large item of size x and $2m - x$ items of size 1 (see Figure 4.1).

The unique optimal packing of each sequence is defined by the partial packing of the first $n/2$ small items. Consider a deterministic online algorithm $\mathbb{A}$ that receives the same advice string for two sequence σ_1 and σ_2. Since $\mathbb{A}$ is deterministic and both sequences start with the same sub-sequence of small items, the partial packing of the algorithm after packing the first $n/2$ items is the same for both σ^1 and σ^2. As a result, the final packing of $\mathbb{A}$ is sub-optimal for at least one them. We conclude that any deterministic online algorithm should receive distinct advice strings for each sequence in the family. Since there are at least



The packing of sequence $\langle 1^{(15)}\ 11\ 11\ 11\ 11\ 10\ 10\ 9\ 8\ 12^{(7)}\rangle$



The packing of sequence $\langle 1^{(15)}\ 11\ 11\ 11\ 11\ 11\ 11\ 11\ 8\ 8\ 12^{(6)}\rangle$

Figure 4.1: The optimal packings for two sequences of the family when $n = 30$ and $m = 6$ (item sizes and bin capacities are scaled by $2m = 12$).

$\left(1 + \frac{n}{(m-1)(m-2)}\right)^{m-3}$ sequences in the family, at least $(m-3)\lg\left(1 + \frac{n}{(m-1)(m-2)}\right) > (m-3)\lg n - 2m\lg m$ bits of advice are required. □

4.3 An Algorithm with Sublinear Advice

In this section, we introduce an algorithm that receives $\lg n + o(\lg n)$ bits of advice and achieves a competitive ratio of $\frac{3}{2}$, for any instance of the online bin packing problem. An offline oracle can compute and write the advice on the tape in linear time, and the online algorithm runs as fast as First-Fit. Thus, the algorithm might be applied in practical scenarios in which it is allowed to have a 'quick look' at the input sequence.

We call items *tiny*, *small*, *medium*, and *large* if their sizes lie in the intervals $(0, 1/3]$, $(1/3, 1/2]$, $(1/2, 2/3]$, and $(2/3, 1]$, respectively. The advice that the algorithm receives is the number of medium items, which we denote by α.

The algorithm reads the advice tape, obtains α, opens α bins, called *critical bins*, and reserves $2/3$ of the space in each of them. This reserved space will be used to pack a medium item in each of the critical bins, and these bins have a *virtual level* of size $2/3$ at the beginning. All other bins have virtual level zero when they are opened. The algorithm serves an item x in the following manner:

- If x is a large item, open a new bin for it. Set the virtual level to its size.

- If x is a medium item, put it in the reserved space of a critical bin B. Update the virtual level to the actual level. (B will not have any reserved space now.)
- If x is small or tiny, use the First Fit (Ff) strategy to put it into any of the open bins, based on virtual levels (open a new bin if required). Add the size of the item to the virtual level.

Note that the critical bins appear first in the ordering maintained by the algorithm as they are opened before other bins.

Theorem 8. *There is an online algorithm which receives* $\lg n + o(\lg n)$ *bits of advice and opens at most* $3/2\,\textsc{Opt}(\sigma) + 2$ *bins to pack any sequence* σ *of size* n.

Proof. We prove that the algorithm described above has the desired property. The value of α is encoded in $X = \lceil \lg(n+1) \rceil$ bits of advice. In order to read this properly from the tape, the algorithm needs to know the value of X. This can be done by adding the self-delimited encoding of X in $e(X) = \lceil \lg X \rceil + 2\lceil \lg \lg(X) \rceil + 2$ bits at the beginning of the tape (see Section 1.2). Consequently the number of advice bits used by the algorithm is $X + O(\lg X)$, which is $\lg n + o(\lg n)$ as stated by the theorem.

Consider the packing of a sequence σ by the algorithm. There are two cases. In the first case, there is a critical bin B so that no other item, except a medium item, is packed in it. Since all tiny items are smaller than $1/3$ and can fit in B, all the non-critical bins that are opened after B include small and large items only. More precisely, they include either a single large item or two small items (except the last one which might have a single small item). Let L, M, and S denote the number of large, medium, and small items. The number of bins used by the algorithm is at most $L + M + S/2 + 1$. Now, if $S \leq M$, this would be at most $L + 3/2M + 1$. Since $L + M$ is a lower bound on the number of bins of Opt, the number of bins used by the algorithm is at most $3/2\,\textsc{Opt}(\sigma) + 1$ and we are done. If $S > M$, Opt should open $L + M$ bins for large and medium items, and in the best case, it packs M small items together with medium ones. For the other $S - M$ bins, Opt has to open at least $(S - M)/2$ bins. Hence, the number of bins of Opt is at least $L + M + (S - M)/2 = L + M/2 + S/2$, and we have $3/2\,\textsc{Opt}(\sigma) \geq 3L/2 + 3M/4 + 3S/4 > L + M + S/2$. Thus, the number of bins used by the algorithm is at most $3/2\,\textsc{Opt}(\sigma) + 1$.

In the second case, we assume that all critical bins include another item in addition to the medium item. We claim that, at the end of packing a sequence, all bins, except possibly two, have level at least $2/3$. First, we verify this for non-critical bins (bins without medium items). If a non-critical bin is opened by a large item, it clearly has level higher than $2/3$. All other non-critical bins only include items of size at most $1/2$. Hence, these bins, except possibly the last one, include at least two items. Among the non-critical bins that include two items, consider two bins b_i and b_j ($i < j$) that have levels smaller than $2/3$. Since b_j contains at least two items, at least one of them has size smaller than $1/3$. This item could fit in b_i by the Ff property. We conclude that all non-critical bins, except possibly two, have level at least $2/3$. Now, suppose two critical bins b_i and b_j have levels smaller than $2/3$. Consider the first non-medium item x which is packed in b_j (in the second case, such an item exists). Since a medium item is packed in the bin, x should be either tiny or small. If x is small, then the level of b_j is at least $1/2 + 1/3$, which contradicts the level of b_j being smaller than $2/3$. Similarly, x cannot be a tiny item of size larger than $1/6$ (since $1/2 + 1/6 \geq 2/3$). Hence, x is a tiny item of size at most $1/6$. This implies that at the time the online algorithm packs x, bin b_i has a virtual level of at least $5/6$. The virtual level is at most $1/6$ larger than the actual level (the final level). Hence, the actual level of b_i is at least $5/6 - 1/6 = 2/3$. We conclude that at most one critical bin has level smaller than $2/3$.

To summarize, there are at most two non-critical bins and one critical bin which have level smaller than 2/3. On the other hand, it is not possible to have one critical bin with contents less than 2/3 and one non-critical one (other than the last) with such contents; this is because the rightmost bin will have an item of size at most 1/3 which should have gone in the leftmost. To conclude, in the final packing, there is at most two bins with content less than 2/3 (in which case both will be non-critical). Hence, the number of bins used by the algorithm is at most $3/2\,\textsc{Opt}(\sigma) + 2$. □

4.4 An Algorithm with Linear Advice

In this section, we present an algorithm that receives $2n + o(n)$ bits of advice and achieves a competitive ratio of $4/3+\varepsilon$ for any sequence of size n, and arbitrarily small but fixed values of ε. Consider an algorithm that receives an *approximate size* for each sufficiently large item x encoded using k bits. The approximate size of x would be larger than its *actual size* by at most an additive term of $1/2^k$. The algorithm can optimally pack items by their approximate sizes and achieve an *approximate packing* which includes a reserved space of size $x + \varepsilon$ ($\varepsilon \leq 1/2^k$) for each item. Precisely, for each sufficiently large item x, the approximate packing includes a reserved space of size $x + \varepsilon$ ($\varepsilon \leq 1/2^k$) for x. This enables the algorithm to place x in the reserved space for it in the approximate packing. Smaller items are treated differently and the algorithm does not reserve any space for them. In the remainder of this section, we elaborate this idea to achieve a 4/3-competitive algorithm.

Notice that the number of bins in an approximate packing can be as large as $\frac{3}{2}$ times the number of bins in an optimal packing. To see that, consider a sequence which is a permutation of

$$\langle \frac{1}{2} + \varepsilon_1, \frac{1}{2} - \varepsilon_1, \frac{1}{2} + \varepsilon_2, \frac{1}{2} - \varepsilon_2, \ldots, \frac{1}{2} + \varepsilon_{n/2}, \frac{1}{2} - \varepsilon_{n/2} \rangle,$$

where $\varepsilon_i < 1/2^n (1 \leq i \leq n/2)$. Since Opt packs all bins tightly, an increase in the sizes of items by a constant (small) ε results in opening a new bin for each two bins Opt uses. Hence, the number of bins in an optimal approximate packing can be as bad as $\frac{3}{2}$ Opt. This example suggests that using approximate packings is not good for the bins in which a small number of large items are tightly packed. To address this issue we divide the bins of Opt into two groups. Informally speaking, a bin is 'good' if we can transfer a set of items, with a relatively small total size, from the bin, so that the sizes of the remaining items can be approximated with advice of small size. Otherwise, we say the bin is 'bad' (in the above example, all bins are bad). This is formalized in the following definition.

Definition 5. *Consider an optimal packing of a sequence σ. Given a small parameter $\varepsilon' < 1/60$, define* good bins *to be those where the total size of the items smaller than 1/4 in the bin is at least $5\varepsilon'$. Define all other bins to be* bad bins.

A part of the advice received for each item x indicates if x is packed by Opt in a good bin or in a bad bin. This enables us to treat items packed in these two groups separately.

Lemma 11. *Consider sequences for which all bins in the optimal packing are good (as defined above). There is an online algorithm that receives $o(n)$ bits of advice and achieves a competitive ratio of 4/3.*

Proof. Call an item *small* if it is smaller than or equal to 1/6 and *large* otherwise. The advice bits define the approximate sizes of all large items with a precision of ε', i.e., it gives the counts for each possible

rounded item size. The amount of advice will be roughly $1/\varepsilon' \lg n$ which is $o(n)$ for constant values of ε'. The online algorithm $\mathbb{A}$ can build the optimal approximate packing of large items. In such a packing, there is a reserved space of size at most $s(x)+\varepsilon'$ for any large item x. The algorithm considers this packing as a partial packing and initializes the level of each bin to be the total sizes of approximated items in that bin. For packing an item x, if x is large, $\mathbb{A}$ packs it in the space reserved for it in the approximate packing. It also updates the level of the bin to reflect the actual size of x. If x is small, $\mathbb{A}$ simply applies the First-Fit strategy to pack x in a bin of the partial packing (and opens a new bin for it if necessary). We prove that $\mathbb{A}$ is 4/3-competitive. In the final packing by $\mathbb{A}$, call a bin 'red' if all items packed in it are small items and call it 'blue' otherwise (the blue bins constitute the approximated packing at the beginning). There are two cases to consider.

In the first case, there is no red bin in the final packing of $\mathbb{A}$, i.e., all small items fit in the remaining space of the bins in the approximate packing of large items. Let σ' be a copy of the input sequence in which the sizes of large items are approximated, i.e., increased by at most ε'; also let X be the number of bins for the optimal packing of σ'. Since there is no red bin in the final packing of $\mathbb{A}$, the number of bins used by $\mathbb{A}$ is equal to X. Consider the optimal packing of the actual input sequence σ. Since all bins are good, one can transfer a subset of items to provide an available space of size at least $5\varepsilon'$ in each bin. After such a transfer, we can increase the sizes of large items to their approximate sizes. Since there are at most 5 large items in each bin and also available space of size at least $5\varepsilon'$, the packing constructed this way is a valid packing for the sequence σ'. Since the size of the transferred items for each bin is at most 1/4, the transferred items from each group of four bins can fit in one new bin. Consequently the number of bins in the new packing is at most $5/4\,\textsc{Opt}(\sigma)$. We know that the final packing by $\mathbb{A}$ is the optimal packing for σ' (with X bins), and in particular not worse than the packing constructed above. Hence, the number of bins used by $\mathbb{A}$ is not more than $5/4\,\textsc{Opt}(\sigma)$.

In the second case, there is at least one red bin in the final packing of $\mathbb{A}$. We claim that all bins in the final packing of $\mathbb{A}$, except possibly the last, have levels larger than 3/4. The claim obviously holds for the red bins since the levels of all these bins (excluding the last one) are larger than 5/6. Moreover, since there is a bin which is opened by a small item, all blue bins have levels larger than 5/6, i.e., the total size of packed items and reserved space for the large items is larger than 5/6. Since there are at most 5 large items in each bin, the actual level of each bin in the final packing of $\mathbb{A}$ is at least $5/6 - 5\varepsilon'$, which is not smaller than 3/4 for $\varepsilon' \leq 1/60$. So, all bins, except possibly one, have levels larger than 3/4. Consequently, the algorithm is 4/3-competitive. □

It remains to address how to deal with bad bins. The next three lemmas do this.

Lemma 12. *Consider sequences for which all bins in the optimal packing include precisely two items. There is an algorithm that receives* 1 *bit of advice per request and achieves an optimal packing.*

Proof. The single bit of advice for an item x determines whether or not the *partner* of x appeared as a previous request, where the partner of x is the item which is packed in the same bin as x in Opt's packing. Consider an algorithm $\mathbb{A}$ that works as follows: If the partner of x has not been requested yet, $\mathbb{A}$ opens a new bin for x. Otherwise, it uses the Bf strategy to pack x in one of the open bins. We claim that $\mathbb{A}$ achieves an optimal packing.

Assume that initially we have a mapping that maps the last item to go into a bin to the item it goes on top of in the optimal packing, i.e., it maps the second item of each bin to the first item. We update

this mapping when necessary and maintain the invariant that we can always pack optimally according to the mapping. For packing a request x , if BF does not pack according to this mapping, it packs x on top of y', while, according to the mapping, it was supposed to pack x on top of y, and a later x' is supposed to go on top of y'. Due to the BF strategy, $y' \geq y$, so we can update the mapping to map the currently unprocessed x' to y, and, of course, x to y'. □

Lemma 13. *Consider a sequence σ for which all items have sizes larger than $1/4$ and for which each bin in* OPT*'s packing includes precisely three items. The* HARMONIC *algorithm uses at most $4/3\,\mathrm{OPT}(\sigma) + 3$ bins to pack such a sequence.*

Proof. The proof is based on a simple weighting function. Call an item x *large* if $1/3 < x \leq 1/2$ and *small* otherwise ($1/4 < x \leq 1/3$). Note that, since there are three items in each bin and all are larger than $1/4$, no item can have size $1/2$ or larger. Define the weight of x to be $1/2$ if x is large and $1/3$ if it is small. Consider a bin B in the packing of σ by OPT. Since there are three items in B, its weight is maximized when there are two large items and one small item in it (three large item do not fit in the same bin). Hence, the weight of each bin in the OPT packing is at most $2 \times 1/2 + 1/3 = 4/3$. Consequently, we have $\mathrm{OPT}(\sigma) \geq 3/4W$, where W is the total weights of all items. The HARMONIC algorithm (HA) simply packs small and large items in separate collections of bins. So, each of the algorithm's bins, except possibly two bins, contains either three small items or two large items. In both cases, the weight of each bin is at least 1 and we have $\mathrm{HA}(\sigma) \leq W + 2$. As a conclusion $\mathrm{HA}(\sigma) \leq 4/3\,\mathrm{OPT}(\sigma) + 2$ which completes the proof. □

Lemma 14. *Consider a sequence σ for which all bins in the optimal packing are bad bins (as defined earlier). There is an algorithm that receives two bits of advice for each request, and opens at most $(4/3 + \frac{5\varepsilon'}{1-5\varepsilon'})\,\mathrm{OPT}(\sigma) + 3$ bins.*

Proof. By the definition of bad bins, for any bin in the optimal packing, all items are either smaller than $5\varepsilon'$ or larger than $1/4$. We call the former group of items *tiny* items and pack them separately using the FF strategy. We refer to other items as *normal items*. Consider an offline packing P which is the same as OPT's packing, except that all tiny items are removed from their bins and packed separately in new bins using the FF strategy. This implies that the number of bins in P is more than $\mathrm{OPT}(\sigma)$ by a multiplicative factor of at most $1 + \frac{5\varepsilon'}{1-5\varepsilon'}$. Let Q be the optimal packing for normal items. Since all normal items are larger than $1/4$, each bin of Q contains at most three items. We say a bin of Q has type i ($i \in \{1, 2, 3\}$), if it contains i normal items. Similarly, we say an item x has type i if it is packed in a type i bin. All items in type 3 bins have sizes smaller than $1/2$ (otherwise one will have size at most $1/4$ which contradicts the assumption). Moreover, the sizes of the items in all type 1 bins (except possibly the last one) are larger than $1/2$ (otherwise a better packing is achieved by pairing two of them). With two bits of advice, we can detect the type of an item as follows: Let b denote the two bits of advice with item x. If b is '01' and $x > 1/2$, then x has type 1; if b is '01' and $x \leq 1/2$, then x has type 3; and if b is '10' or b is '11', then x has type 2. Note that the code '00' is not used at this point (this is used later on), and the use of '10' and '11' is still to be detailed.

Let X_i denote the number of bins of type i ($1 \leq i \leq 3$). Hence, the number of bins in Q is $X_1 + X_2 + X_3$, and consequently the number of bins in P is at least $X_1 + X_2 + X_3 + X'$, where X' is the number of bins filled by tiny items. Consider an algorithm $\mathbb{A}$ that performs as follows. If an item x has type 1, $\mathbb{A}$ simply opens a new bin for x. If x has type 2, $\mathbb{A}$ applies the strategy of Lemma 12 to place it in one of the bins maintained for items of type 2. Recall that the advice in this case is either '10' or '11', so the second bit

provides the advice required by Lemma 12. If x has type 3, $\mathbb{A}$ applies the HA strategy to pack the item in a set of bins maintained for type 3 items. By Lemma 13, the number of bins used by $\mathbb{A}$ for these items is at most $4/3X_3+3$. Finally, $\mathbb{A}$ uses the FF strategy to pack tiny items in separate bins. Consequently, the number of bins used by the algorithm is at most $X_1+X_2+4/3X_3+X'+3 \leq (1+\frac{5\varepsilon'}{1-5\varepsilon'})\,\textsc{Opt}(\sigma)+X_3/3+3 \leq (4/3+\frac{5\varepsilon'}{1-5\varepsilon'})\,\textsc{Opt}(\sigma)+3$. □

Provided with the above lemmas, we arrive at the following result:

Theorem 9. *There is an online algorithm which receives two bits of advice per request, plus an additive lower order term, and achieves a competitive ratio of $4/3+\varepsilon$, for any positive value of ε.*

Proof. Define ε' to be $\frac{11\varepsilon}{60}$. For $\varepsilon < 1/11$, we have $\varepsilon' < 1/60$. Moreover, we have $\frac{5\varepsilon'}{1-5\varepsilon'} \leq \frac{5\varepsilon'}{1-1/12} = \frac{60\varepsilon'}{11} = \varepsilon$. In an optimal packing, divide bins into good and bad bins using Definition 5. Also, let Gd and Bd respectively denote the number of good and bad bins. Use advice bits to distinguish items which are packed in good and bad bins, and pack them in separate lists of bins. More precisely, let the two bits of advice for an item x be '00' if it is packed by OPT in a good bin, and apply Lemma 11 to pack these items in at most $4/3Gd$ bins. Similarly, apply Lemma 14 to pack items from bad bins in at most $(4/3+\frac{5\varepsilon'}{1-5\varepsilon'})Bd+3 \leq (4/3+\varepsilon)Bd+3$ bins, using bits of advice of the form '01', '10', or '11', as discussed in the proof of Lemma 14. Consequently, the number of bins used by the algorithm of the algorithm will be at most $4/3Gd+(4/3+\varepsilon)Bd+3 \leq (4/3+\varepsilon)\,\textsc{Opt}(\sigma)+3$. □

4.5 A Lower Bound for Linear Advice

In this section, we show that an advice of linear size is required to achieve a competitive ratio better than 9/8 for the online bin packing problem. In our analysis, we make a reduction from the Binary String Guessing Problem with Known History (2-SGKH) as defined in Section 1.2. Recall that in 2-SGKH problem, a bitstring is revealed in an online manner and an online algorithm has to guess the content of each bit before it is revealed. Any algorithm that correctly guesses more than half of the input bits must receive an advice of linear size (Lemma 1).

Since the number of bits needed to express the number of '0's in the input is at most $\lceil \lg(n+1) \rceil \leq \lg n+1$, and this number can be given as advice by an oracle, if it is not given to the algorithm otherwise, we easily obtain the following lemma from Lemma 1. Recall that the definition of e, the length of the encoding function, is given in Section 1.2.

Lemma 15. *Consider instances of size n of the 2-SGKH problem in which the number of '0's is given to the algorithm as part of the input. For these instances, any deterministic algorithm that is guaranteed to guess correctly on more than αn bits, for $1/2 \leq \alpha < 1$, needs to read at least $(1+(1-\alpha)\lg(1-\alpha)+\alpha\lg\alpha)n-e(n)$ bits of advice.*

Proof. Assume to the contrary that the statement is not true. Hence, there is an algorithm, BSGA , that knows the number of '0's and receives fewer than $(1+(1-\alpha)\lg(1-\alpha)+\alpha\lg\alpha)n-e(n)$ bits of advice while guessing correctly on more than αn bits. This algorithm can be used to serve arbitrary instances of the 2-SGKH problem (in which the number of '0's is not known). Modify the advice tape used by the

algorithm BSGA so that it contains at most $e(n)$ additional bits at the beginning specifying the number of '0's. (This can be done with the self-delimited encoding of the number of '0's.) The algorithm for 2-SGKH reads this number and gives it to BSGA. Then it asks BSGA for its guess for each bit in the sequence and answers the same as BSGA. It also informs BSGA of when it is correct and when it is wrong, with the same information it is given. The algorithm is correct exactly when BSGA is correct. The total number of advice bits will be less than $e(n)+(1+(1-\alpha)\lg(1-\alpha)+\alpha\lg\alpha)n-e(n) = (1+(1-\alpha)\lg(1-\alpha)+\alpha\lg\alpha)n$. However, Lemma 1 implies that no algorithm can guess correctly on more than αn bits with this many bits of advice. In conclusion, the initial assumption is incorrect and the statement holds. □

In order to relate the Binary String Guessing Problem to the online bin packing problem, we introduce another problem called the Binary Separation Problem.

Definition 6. *The* Binary Separation Problem *is the following online problem. The input* $I = (n_1, \sigma = \langle y_1, y_2, \ldots, y_n\rangle)$ *consists of* $n = n_1 + n_2$ *positive values which are revealed one by one. There is a fixed partitioning of the set of items into a subset of* n_1 large *items and a subset of* n_2 small *items, so that all large items are larger than all small items. Upon receiving an item* y_i*, an online algorithm for the problem must guess if* y *belongs to the set of small or large items. After the algorithm has made a guess, it is revealed to the algorithm whether* y_i *actually belongs to class of small or large items.*

We provide reductions from the modified Binary String Guessing Problem to the Binary Separation Problem, and from the Binary Separation Problem to the online bin packing problem. In order to reduce a problem P_1 to another problem P_2, given an instance of P_1 defined by a sequence σ_1 and a set of parameters η_1 (such as the length of σ_1 or the number of '0's in it), we create an instance of P_2 which is defined by a sequence σ_2 and also a set of parameters η_2. In our reductions, we assume η_2 is derived from η_1, and since σ_1 is revealed in an online manner, σ_2 is created in an online manner by looking only at η_1 and the revealed items of σ_1.

Lemma 16. *Assume that there is an online algorithm that solves the Binary Separation Problem on sequences of length* n *with* $b(n)$ *bits of advice, and makes at most* $r(n)$ *mistakes. Then there is also an algorithm that solves the Binary String Guessing Problem on sequences of length* n*, assuming the number of '0's is given as a part of input, so that the algorithm receives* $b(n)$ *bits of advice and makes at most* $r(n)$ *errors.*

Proof. We assume that we have an algorithm BSA that solves the Binary Separation Problem under the conditions of the lemma statement. Using that algorithm, we define the number n_1 of large items to be the number of '0's in the instance of the Binary String Guessing Problem. Then, we implement our algorithm BSGA for the Binary String Guessing Problem as outlined in Algorithm 2, which defines the reduction. This BSGA implementation, defined in Algorithm 2, functions as an adversary for BSA, e.g., in Line 4, BSGA gives BSA its next request. Notice that we ensure that the BSGA makes a correct guess if and only if BSA makes a correct guess. The advice tape is filled with bits of advice for this combined algorithm. The BSGA uses the BSA as a subroutine, but all the questions are effectively coming from the BSA.

The set-up, reminiscent of binary search, is carried out as specified in the algorithm with the purpose of ensuring that when the BSA is informed of the actual class of the item it considered, no result can contradict information already obtained. Specifically, the next item for the BSA to consider is always in between the largest item which has previously been deemed 'small' and the smallest item which has previously been deemed 'large'. The fact that we give the middle item from that interval is unimportant; any value chosen from the open interval would work. □

Algorithm 2: Implementing Binary String Guessing via Binary Separation.

```
The Binary Guessing algorithm knows the number of '0's (n_1) and passes it as a parameter
(the number of large items) to the Binary Separation algorithm
small = 0; large = 1
repeat
    mid = (large − small) / 2
    class_guess = SeparationAlgorithm.ClassifyThis(mid)
    if class_guess = 'large' then
        bit_guess = 0
    else
        bit_guess = 1
    actual_bit = Guess(bit_guess) {The actual value is received after guessing (2-SGKH).}
    if actual_bit = 0 then
        large = mid {We let 'large' be the correct decision.}
    else
        small = mid {We let 'small' be the correct decision.}
until end of sequence
```

Now, we prove that if we can solve a special case of the bin packing problem, we can also solve the Binary Separation Problem.

Lemma 17. *Consider the bin packing problem on sequences of length $2n$ for which* Opt *opens n bins. Assume that there is an online algorithm $\mathbb{A}$ that solves the problem on these instances with $b(n)$ bits of advice and opens at most $n + r(n)/4$ bins. Then there is also an algorithm* Bsa *that solves the Binary Separation Problem on sequences of length n with $b(n)$ bits of advice and makes at most $r(n)$ errors.*

Proof. In the reduction, we encode requests for the Bsa as items for bin packing. Assume we are given an instance $I = (n_1, \sigma = \langle y_1, y_2, \ldots, y_n \rangle)$ of the Binary Separation problem, in which n_1 is the number of large items ($n_1 + n_2 = n$), and the values of y_ts are revealed in an online manner ($1 \leq t \leq n$). We create an instance of the bin packing problem which has length $2n$. Algorithm 3 shows the details of the reduction. The bin packing sequence starts with n_1 items of size $\frac{1}{2} + \varepsilon_{min}$ (in Algorithm 3, the variable 'NumberOfLargeItems' is n_1 from the Binary Separation Problem). Any algorithm needs to open a bin for each of these n_1 items. We create the next n items in an online manner, so that we can use the result of their packing to guess the requests for the Binary Separation Problem. Let $\tau = y_t$ ($1 \leq t \leq n$) be a requested item of the Binary Separation Problem; we ask the bin packing algorithm to pack an item whose size is an increasing function of τ, and slightly less than $\frac{1}{2}$. Depending on the decision of the bin packing algorithm for opening a new bin or placing the item in one of the existing bins, we decide the type of τ as being consecutively small or large. The last n_2 items of the bin packing instance are defined as complements of the items in the bin packing instance associated with small items in the binary separation instance (the complement of item x is $1 - x$). We do not need to give the last items complementing the small items in order to implement the algorithm, but we need them for the proof of the quality of the correspondence that we are proving.

Call an item in the bin packing sequence 'large' if it is associated with large items in the Binary Separation Problem, and 'small' otherwise. For the bin packing sequence produced by the reduction, an optimal algorithm pairs each of the large items with one of the first n_1 items (those with size $\frac{1}{2} + \varepsilon_{min}$),

Algorithm 3: Implementing Binary Separation via Special Case Bin Packing.

Choose ε_{min} and ε_{max} so that $0 < \varepsilon_{min} < \varepsilon_{max} < \frac{1}{6}$
Choose a decreasing function $f\colon \mathbb{R} \to (\varepsilon_{min}..\varepsilon_{max})$
for i = 1 to NumberOfLargeItems **do**
    BinPacking.Treat($\frac{1}{2} + \varepsilon_{min}$) {The decision can only be to open a bin.}
repeat
    Let τ be the next request
    decision = BinPacking.Treat($\frac{1}{2} - f(\tau)$){Placing an item of size $\frac{1}{2} - f(\tau)$ }
    if decision = 'packed with an $\frac{1}{2} + \varepsilon_{min}$ item' **then**
        class_guess = 'large'
    else
        class_guess = 'small'
    actual_class = Guess(class_guess)
    if actual_class = 'small' **then**
        SmallItems.append($\frac{1}{2} - f(\tau)$) {Collecting small items for later.}
until end of request sequence
for i = 1 to len(SmallItems) **do**
    BinPacking.Treat(1 − SmallItems[i]) {The decision is not used.}

placing them in the first n_1 bins. Opt pairs the small items with their complements, starting one of the next n_2 bins with each of these small items. Hence, the number of bins in an optimal packing is $n_1 + n_2 = n$. The values ε_{min} and ε_{max} in Algorithm 3 must be small enough so that no more than two of any of the items given in the algorithm can fit together in a bin. No other restriction is necessary.

We claim that each extra bin used by the bin packing algorithm, but not by Opt, results in at most four mistakes made by the derived algorithm on the given instance of the Binary Separation Problem. Consider an extra bin in the final packing of $\mathbb{A}$. This bin is opened by a large item which is incorrectly guessed as being small (bins which are opened by small items also appear in Opt's packing). Note that large items do not fit in the same bins as complements of small items. The extra bin has enough space for another large item. Moreover, there are at most two small items which are incorrectly guessed as being large and placed in the space dedicated to the large items of the extra bin. Hence, there is an overhead of at least one for four mistakes. To summarize, $\mathbb{A}$ has to decide if a given item is small or large and performs accordingly, and it incurs a cost of at least 1/4 for each incorrect decision. If $\mathbb{A}$ opens at most $n + r(n)/4$ bins, the algorithm derived from $\mathbb{A}$ for the Binary Separation Problem makes at most $r(n)$ mistakes. □

Theorem 10. *Consider the online bin packing problem on sequences of length n. To achieve a competitive ratio of c $(1 < c < 9/8)$, an online algorithm needs to receive at least* $(n(1 + (4c - 4)\lg(4c - 4) + (5 - 4c)\lg(5 - 4c)) - (\lceil \lg(n + 1) \rceil + 2\lceil \lg(\lceil \lg(n + 1) \rceil + 1) \rceil + 1))/2$ *bits of advice.*

Proof. Consider a bin packing algorithm $\mathbb{A}$ that receives $b(n)$ bits of advice and achieves a competitive ratio of c. This algorithm opens at most $(c - 1)\,\textsc{Opt}(\sigma)$ bins more than Opt, so when $\textsc{Opt}(\sigma) = n/2$, it opens at most $(c - 1)n/2$ more bins. By Lemma 17, the existence of such an algorithm implies that there is an algorithm $\mathbb{A}$ that solves the Binary Separation Problem on sequences of length $n/2$ with b bits of advice and makes at most $2(c - 1)n$ errors. By Lemma 16, this implies that there is an algorithm $\mathbb{B}$ that solves the Binary String Guessing Problem on sequences of length $n/2$ with b bits of advice and makes

at most $2(c-1)n$ mistakes, i.e., it correctly guesses the other $n/2 - 2(c-1)n = (5-4c)n/2$ items. Let $\alpha = 5 - 4c$, and note that α is in the range $[1/2, 1)$ when c is in the range $(1, 9/8]$. Lemma 15 implies that in order to correctly guess more than $\alpha n/2$ of the items in the binary sequence, we must have $b(n)$ larger than or equal to $((1 + (1-\alpha)\lg(1-\alpha) + \alpha \lg \alpha)n - e(n))/2$. Replacing α with $5 - 4c$ completes the proof. □

Thus, to obtain a competitive ratio strictly better than 9/8, a linear number of bits of advice is required. For example, to achieve a competitive ratio of 17/16, at least $0.188n$ bits of advice are required asymptotically.

Corollary 2. *Consider the bin packing problem for packing sequences of length n. To achieve a competitive ratio of $9/8 - \delta$, in which δ is a small, but fixed positive number, an online algorithm needs to receive $\Omega(n)$ bits of advice.*

4.6 Remarks

We conjecture that a sublinear number of bits of advice is enough to achieve competitive ratios smaller than 4/3. Note that our results imply that we cannot hope for ratios smaller than 9/8 with sublinear advice. The lower bound presented here does not give a result better than half a bit per request asymptotically, regardless of how close to a competitive ratio of 1 we wish to obtain. It would be interesting to strengthen this result.

Online bin packing with advice is also studied by Renault et al. in [123]. They present an algorithm for online bin packing which is $(1+2\delta)$-competitive using $s = \frac{1}{\delta}\lg\frac{2}{\delta^2} + \lg\frac{2}{\delta^2} + 3$ bits of advice per request. This is a nice theoretical result, showing that techniques for designing polynomial approximation schemes can also be useful when considering online algorithms with advice. Unfortunately, when applying their theorem for reasonable length input sequences and c-competitiveness, where $c \leq 4/3$, the result is not better than the $\lg \textsc{Opt}(\sigma)$ bits per request, which are sufficient for optimality. To put this in context, the algorithm that we introduced in Section 4.4 is 4/3-competitive and uses only two bits of advice per request (plus some lower order term). To achieve a ratio of 4/3, Renault et al.'s algorithm has $\delta = 1/6$, so the number of bits is at least $6\lg 72 + \lg 72 + 3 > 46$ per request. Thus, there must be more than 2^{46} bins before the result improves on the naive approach of using $\lg \textsc{Opt}(\sigma)$ bits per request.

Chapter 5

Square Packing with Advice

In this chapter, we introduce an almost-online square packing algorithm which places squares in an online, sequential manner. In doing so, it receives advice that can be computed offline in linear time. The algorithm achieves a competitive ratio of at most 1.84 which is significantly better than the best existing online algorithm which has a competitive ratio of 2.1187. Our algorithm can be regarded as a *practical* offline algorithm for square packing.

5.1 Introduction

Recall that in the square packing problem squares of different sizes, called *items*, should be packed into unit squares called *bins*. The problem is a generalization of the bin packing problem into two dimensions (see Section 2.1.4). We refer to the length of each side as the *size* of a square. Each bin has size 1 which is an upper bound for the size of the input squares. Given a set of squares, we would like to place them into the smallest number of bins so that there is no overlap between two squares assigned to a bin.

Square packing is studied under both offline and online settings. Most offline algorithms which are introduced to improve the worst-case guarantees involve an integer programming formulation of the problem and are too complicated to be applied in practice. In this chapter, we consider an *almost online* setting in which a fast offline oracle provides an advice of logarithmic size to the online algorithm. We show that this small amount of advice significantly boosts the performance of the online algorithm. The offline oracle is restricted to run in linear time and only make one pass to collect some basic statistics about the input. Assuming the advice of size $O(\lg n)$, this setting of the problem is closely related to the streaming model [10] and map-reduce model [105]. We define the almost-online square packing problem as follows:

Definition 7. *In the* almost online square packing problem*, the input is a sequence of squares (items) with sizes $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ revealed in an online manner ($0 < \sigma_i \leq 1$). The goal is to pack these squares into a minimum number of squares of unit size (bins). At time step t, an online algorithm should pack square σ_t into a bin. The decision of the algorithm to select the target bin is a function of $\Phi, \sigma_1, \ldots, \sigma_{t-1}$, where Φ is advice of size $O(\lg n)$ generated by an offline oracle that runs in linear time (linear in the size of the input).*

We introduce an almost-online algorithm for the square packing problem which achieves a competitive ratio of at most 1.84. The offline oracle simply counts the number of squares whose sizes lie in different intervals defined by the algorithm. The online algorithm uses the advice to pack squares in an efficient way that ensures a good competitive ratio. The algorithm is quite simple and runs as quickly as its online counterparts. In some sense, the algorithm can be seen as a generalization of the algorithm presented in Section 4.3 for the classic bin packing problem; that algorithm achieved a competitive ratio of 1.5 when provided with advice of logarithmic size.

Our algorithm indicates that advice of logarithmic size is sufficient to outperform the existing online algorithms. Note that the competitive ratio 1.84 of this algorithm is significantly better than 2.1187 of the best existing algorithm [88]. The algorithm can also be regarded as a streaming algorithm with two passes. Although the algorithm does not perform as well as the offline algorithms (in particular the APTAS algorithm), its simple and fast nature makes it useful even in the offline setting.

5.2 An Algorithm with Sublinear Advice

In this section we introduce our square packing algorithm. The algorithms define *types* for squares based on their sizes. A square has type i if its size is in the interval $(\frac{1}{i+1}, \frac{1}{i}]$ for $1 \leq i \leq 14$. Squares of size smaller than $1/15$ have type 15 and are referred to as *tiny* squares. Squares of type 1 are called *large* squares and are further divided into types 1a, 1b, 1c, and 1d with sizes in the intervals $(4/5, 1]$, $(2/3, 4/5]$, $(3/5, 2/3]$, and $(1/2, 3/5]$, respectively. Similarly, squares of type 2 are divided into types 2a and 2b with sizes in the intervals $(2/5, 1/2]$ and $(1/3, 2/5]$, respectively. We refer to items of type 2 as *medium* items and items of types $3, 4, \ldots, 14$ as *small* items.

In total, there is a constant number of item types. The offline oracle simply scans the input and counts the number of items for each type except the last type associated with tiny items. These numbers are encoded as advice of size $\Theta(\lg n)$. The online algorithm makes use of this advice to achieve a competitive ratio of at most 1.84. To describe the algorithm, we start with the following two lemmas (note the distinction between the *size* and the *area* of a square).

Lemma 18. *[71] Consider the square packing problem in which all items are smaller than or equal to $1/M$ for some integer $M \geq 2$. There is an online algorithm that creates a packing in which all bins, except possibly one, have an occupied area at least $(M^2 - 1)/(M + 1)^2$.*

Lemma 19. *There is an online square packing which creates packings in which all bins, except possibly a constant number of them, have an occupied area more than $1/4$.*

Proof. Consider an online algorithm that places each large square in a separate bin. Since these items have size large than $1/2$, the occupied area in each bin is more than $1/4$. For squares in the interval $(1/3, 1/2]$, the algorithm places four squares in the same bin; the total occupied area will at least $4/9 > 1/4$. For squares that are no larger than $1/3$, the algorithm of Lemma 18 is applied which ensures that the total volume of each bin is at least $(9 - 1)/(3 + 1)^2 = 8/16 = 1/2$. □

Given the advice, the online algorithm knows upper and lower bounds for the size of all items (except tiny items). Before packing the sequence in an online manner, the algorithm creates an *approximate packing* in which there is a *reserved* area for each non-tiny item. The reserved area for an item x an upper

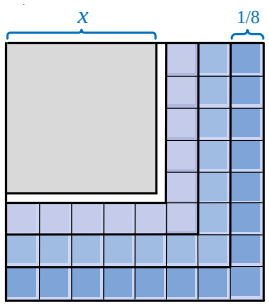


Figure 5.1: L-shape tiling for small items of type 8.

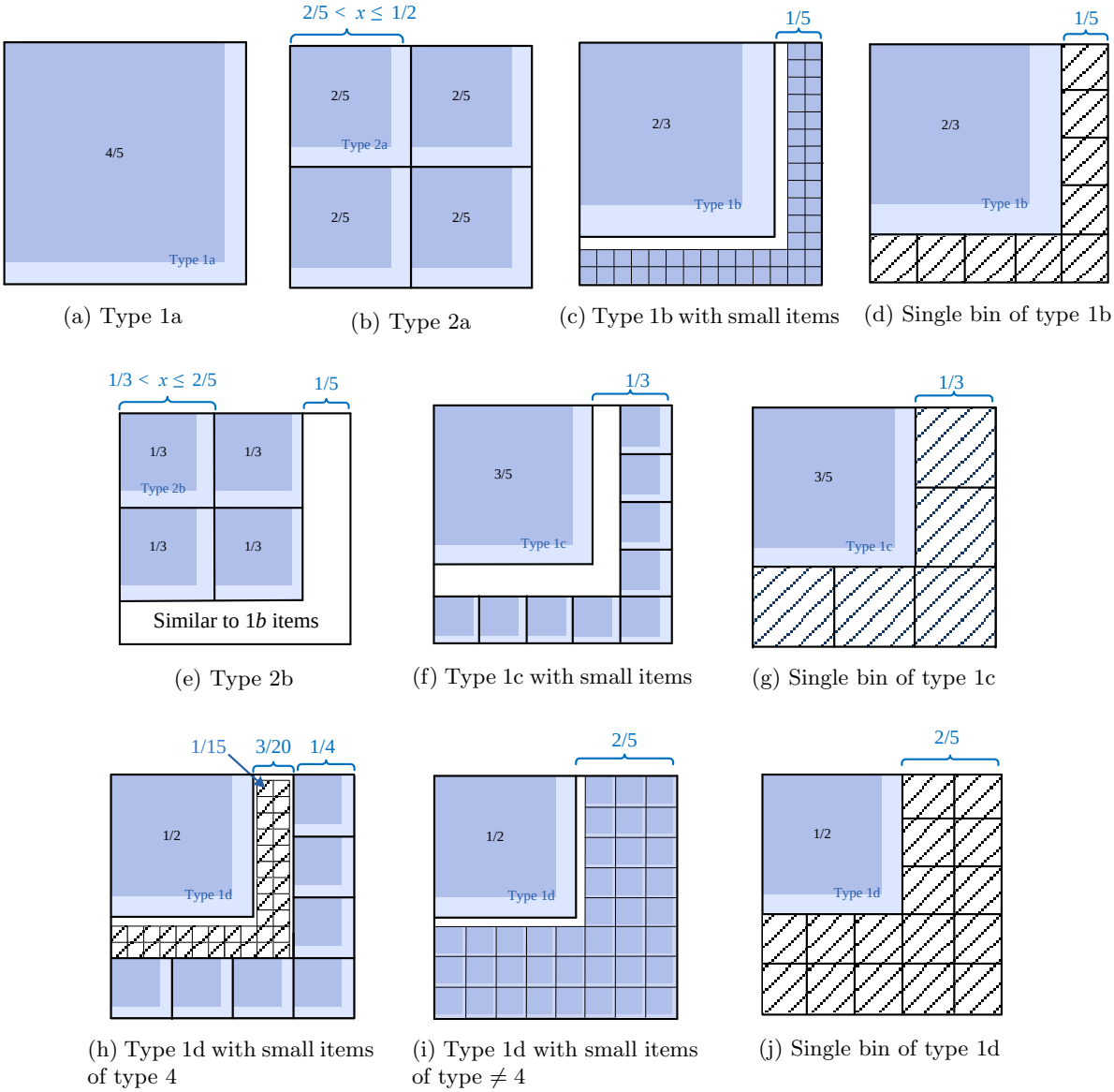
bound for the actual area of the square (which is revealed later). To create the approximate packing, the algorithm opens a new bin for each large item. Note that no two large squares can fit in the same bin. Moreover, the algorithm opens a new bin for each four squares of type 2. In doing so, it treats squares of type 2a and 2b separately, i.e., it does not place a 2a item and a 2b item in the same bin.

Depending on the type of the large and medium items, there might be enough space for small items in the opened bins. Assume a square of size x is reserved for a large item or a group of medium items in a bin B. Also, assume the algorithm places x on the top-left corner of the bin. We use an *L-shape tiling* to place small items of the same types into B. As before, by 'placing' a small item we mean reserving a sufficient space for the item in the approximate packing. Consider items of type $i \geq 3$. These items have their sizes in the range $(1/(i+1), 1/i]$. Hence, $2i-1$ items can be placed on the right and bottom sides of the bin; these squares form an L shape set of tiles. In case $1 - x \geq 1/i$, there is enough space for the L-shape tiling of another $2i-3$ items of type i (see Figure 5.1). More generally, we prove the following lemma.

Lemma 20. *Given a bin in which an area for a square of size at most $x \geq 1/2$ is reserved, one can apply the L-shape tiling to place $2ki - k^2$ small items of type $i \in \{3, 4, \ldots, 14\}$ into the bin where $k = \lfloor (1-x)i \rfloor$.*

Proof. Since the small items are in the range $(1/(i+1), 1/i]$, the first L-shape includes $2i-1$ items. The second L-shape includes one less item on each side and includes $2i-3$ items. More generally, the jth L-shape includes $2i-(2j-1)$ items. The total number of items after k iterations of the L-shape tiling will be $\sum_{j=1}^{k} 2i-(2j-1) = 2ki-k^2$. Moreover, k items of type i require a width of k/i and we should have $k/i + x \leq 1$ which implies $k \leq (1-x) \times i$. □

As mentioned earlier, the algorithm opens a new bin for each large square as well as each four medium squares. The bins opened so far are called LM-bins (Large-Medium bins). Initially, all LM-bins are *single*, i.e., there is no reserved area for small items in them. An LM-bin with a square of type 1a or four squares of type 2a remains single, i.e., no other items will be placed there (Figures 5.2a,5.2b). An LM-bin with a large item of type 1b has enough space for small items of types 5 or larger and later the algorithm uses the L-shape tiling to fill the empty space with these items (Figures 5.2c). Any four items of type 2b form a single square whose size is in the interval $(2/3, 4/5]$ and will be treated like a 1b item, i.e., the algorithm later places small items of types 5 or larger in these bins (Figure 5.2e). Similarly, the algorithm opens a



(a) Type 1a

(b) Type 2a

(c) Type 1b with small items

(d) Single bin of type 1b

(e) Type 2b

(f) Type 1c with small items

(g) Single bin of type 1c

(h) Type 1d with small items of type 4

(i) Type 1d with small items of type $\neq 4$

(j) Single bin of type 1d

Figure 5.2: A summary of LM-bins in an approximate packing. The dark squares indicate the lower bound for the size of the squares of different types while the light parts indicate the upper bound (i.e., the reserved space). The striped squares indicate the live squares which are used for tiny items.

bin for each item of type 1c and 1d and the remaining area will be used to place items of types 3 or larger (Figures 5.2f,5.2i).

The algorithm uses an L-shape tiling to place small items of types 3 to 14 in increasing order by type in the following manner. To place items of type i where $3 \leq i \leq 14$), the algorithm selects LM-bins with maximum reserved space which have enough space for items of type i. For example, items of type 3 can be placed into LM bins with large items of type 1c or 1d. Among these two, the algorithm puts priority to bins with items of type 1c. If it runs out of these bins (after placing certain number of type-3 items), it applies the L-shape tiling to place items of type 3 with items of type 1d. If it also runs out of type 1d items, it places 9 items of type 3 into the same bin.

More generally, the algorithm uses the L-shape tiling to place small items of type i in the LM-bins with maximum reserved area. If there are not enough LM-bins for placing small items, it opens new bins for them. We call these bins *harmonic bins*. A harmonic bin of type $i \geq 3$ includes i^2 small items of type i. LM-bins and harmonic bins form the approximate packing of an input sequence.

Inside some LM-bins, there is an empty area which will be used for placing tiny items. If there is a single LM-bin in the approximate packing, all the available space (i.e., the area which is not reserved for large or medium items) can be partitioned into squares of size 1/5 or larger. We call these squares *large live squares*. Moreover, for LM-bins with 1d items accompanied by small items of type 4, the empty area can be used to reserve 40 squares of sizes 1/15 (figures 5.2h). We refer to these squares as *small live squares*.

Provided with the approximate packing, the online algorithm places items in the input sequence in the following manner. Any non-tiny item is placed in the reserved area for its type in the approximate packing. To place tiny items, the algorithm first uses the live squares. Note that the size of live squares are at least 1/15 which is the upper bound for the size of tiny items. Large live squares have size at least 1/5; hence, we can use Lemma 18 to place tiny items in the large squares so that at least half of their area be occupied by tiny items. We declare a large live square as closed if half of its area is occupied. If all large live squares are closed, we use the algorithm of Lemma 19 to place tiny items in the small live squares. We declare a small live square as closed if its occupied area is at least a quarter of its total area. If all large and small live squares are closed, a new bin is opened for the tiny items and again the algorithm of Lemma 18 is applied. We call these bins *tiny* bins. This way, there are possibly three types of bins in the final packing of the algorithm, namely, LM-bins, Harmonic bins, and tiny bins.

Analysis

In this section, we prove that our algorithm has a competitive ratio of at most 1.84. Consider an LM-bin in the approximate packing which has a reserved area x for large or medium items, while the remaining area is filled with (reserved for) small items of type i. Let m denote the number of small squares in the bin given by Lemma 20. The occupied area by these squares in the final packing is at least $m \times 1/(i+1)^2$ since the size of a small bin of type i is more than $1/(i+1)^2$. Table 5.1 indicates the minimum area covered area by small items of type i in the LM-bins which include large or medium items of type j. For example, when $j = 1c$ and $i = 4$, the reserved space for the large item is 2/3 and by lemma 20, 7 items of type 4 can be placed in the bin. These items occupy an area of at least $7 \times 1/25$ as indicated in the table. In our analysis, we are interested in the minimum covered area by all small items in an LM-bin, i.e., the minimum value in the rows of Table 5.1. For example, for LM-bins with a large item of type 1c, the

Type j	Reserved	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$	$i=11$	$i=12$	$i=13$	$i=14$	tiny
1a,2a	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1b,2b	4/5	0	0	1/4	11/49	13/64	15/81	**17/100**	36/121	40/144	44/169	48/196	52/225	9/50
1c	2/3	5/16	7/25	**1/4**	20/49	24/64	28/81	45/100	51/121	57/144	80/169	88/196	96/225	5/18
1d	3/5	5/16	**7/25**	16/36	20/49	24/64	39/81	45/100	64/121	72/144	80/169	105/196	115/225	16/50

Table 5.1: Lower bounds for the occupied area by the small items of type i in the LM-bins. The last column indicate the occupied area by the tiny items when there is no small item in the bin (i.e., the bin is single in the approximate packing). The highlighted numbers indicate the minimum covered area among all types of small and tiny items.

minimum occupied area is given by small items of type 5 where their total occupied area is at least 1/4. Table 5.1 also indicates the occupied area by tiny items when they are placed in the 'large' live squares of single bins. For example, for a single bin in the approximate packing that contains a 1c item, there are 5 live squares with total area of 5/9; at least half of this area (i.e., 5/18) is occupied by the tiny items, as indicated in Table 5.1.

To prove the upper bound for the competitive ratio, we consider a few cases separately in the following lemmas.

Lemma 21. *Assume there is a tiny bin in the final packing of the algorithm. Then the occupied area in all bins, except possibly a constant number of them, is more than* 9/16.

Proof. We prove the claim for tiny, harmonic, and LM-bins separately. Tiny bins are opened using the algorithm of Lemma 18; since the size of items is smaller than 1/15, the occupied area of all bins, except possibly one of them, is at least $(15^2-1)/(15+1)^2 = 224/256 > 9/16$. A harmonic bin of type i has place for i^2 items of type i ($i \geq 3$). Hence, the occupied area of such a bin is more than $i^2/(i+1)^2$. This value increases as i grows which implies that the minimum occupied area of a harmonic bin is at least 9/16.

Next, we consider LM-bins. The occupied area of bins which include an item of type 1a or four items of type 2a is at least $16/25 > 9/16$. We know that other LM-bins cannot be single; otherwise, the algorithm would have placed tiny items in those bins (instead of opening new bins).

Assume a bin includes an item of type 1b or four items of type 2b. Since the bin is not single, it is either accompanied by small or tiny items. In both cases, as Table 5.1 suggests, the occupied area by small or tiny items is at least 17/100. Hence, the occupied area of the bin is more than $4/9 + 17/100 > 9/16$. With a similar argument, the occupied area of bins with a 1c item is more than $9/25 + 1/4 > 9/16$.

Consider an LM-bin which includes a 1d item. If the bin includes small items of type $i \neq 4$ or tiny items, as Table 5.1 indicates, the small or tiny items occupy an area at least 16/50 and the occupied area of the bin will be more than $1/4 + 16/50 > 9/16$. If the bin includes small items of type 4, there are 40 live small squares in the bin (Figure 5.2h). Since the algorithm has opened tiny bins, by Lemma 19, at least 1/4 of the total area of the live squares is occupied. Also, by Table 5.1, the total occupied area of the small items is at least 7/25. In total, the occupied area of the bin is more than $1/4 + 7/25 + 1/4 \times 40/225 = 517/900 > 9/16$. □

The above lemma implies that if the algorithm opens a tiny bin, the number of opened bins by the algorithm is at most $Ar \times 16/9$ where Ar is total area of all items in the sequence. Since Opt opens at least Ar bins, the competitive ratio of the algorithm at most $16/9 < 1.84$.

Type	Size	Area	Weight	Density
1a	$s(x) \in (4/5, 1]$	$> 16/25$	1	< 1.5625
1b	$s(x) \in (2/3, 4/5]$	$> 4/9$	$1 - 0.17\alpha \approx 0.698$	< 1.57
1c	$s(x) \in (3/5, 2/3]$	$> 9/25$	$1 - 0.25\alpha \approx 0.556$	< 1.55
1d	$s(x) \in (1/2, 3/5]$	$> 1/4$	$1 - 7\alpha/25 \approx 0.503$	< 2.01
2a	$s(x) \in (2/5, 1/2]$	$> 4/25$	0.25	< 1.5625
2b	$s(x) \in (1/3, 2/5]$	$> 1/9$	$0.25 - 0.0425\alpha \approx 0.1745$	< 1.57
$3, \ldots, 14$	$s(x) \in (1/15, 1/3]$	-	$\alpha s(x)^2$	$\alpha \approx 1.78$
15	$s(x) \in (0, 1/15]$	-	0	0

Table 5.2: Characteristics of items of different types in Lemma 22.

Lemma 22. *Assume there is no tiny bin while there is a harmonic bin of type $i \geq 5$ in the final packing of the algorithm. Then the competitive ratio is no more than* 1.84.

Proof. We use a weighting function as follows. We define the weight of items of types 1a and 2a to be respectively 1 and 1/4. Tiny items have weight 0. Let α be a constant equal to 16/9. A small item x of type 3 or larger has weight $\alpha s(x)^2$. The weight of any other item y is defined in a way that the total weight of y and small or tiny items accompanied with y in the same bin be at least 1. Note that, since there is a harmonic bin of type ≥ 5, large and medium items like y are accompanied with some small or tiny items.

For bins with an item of type 1b, except possibly a constant number of them, the minimum area occupied by small or tiny items is at least 17/100 (see Table 5.1). So, the weights items of type 1b is defined as $1-0.17\alpha \approx 0.698$. Similarly, the weights of items of types 1c and 1d are defined as $1-\alpha/4 \approx 0.556$ and $1 - 7\alpha/25 \approx 0.503$, respectively. Items of type 2b have weight $1/4 - 17\alpha/400 < 0.1745$ which is the same as a 1b item when divided between four 2b items placed in the bin.

In our analysis, we refer to density of an item as the ratio between its size and area. Table 5.2 provides a summary of the weights and densities of items. To prove the lemma, we show the followings:

claim 1: All bins in the final packing, except possibly a constant number of them, have weight at least 1.
claim 2: It is not possible to place a set of items into a bin so that the total weight of the items exceeds 1.84.

Claim 1 implies that the number of bins used by the algorithm is at most equal to W, which is the total weight of items in the sequence. Claim 2 implies that the number of bins in an optimal packing is at least $W/1.84$. Hence, the two claims together prove the lemma.

For claim 1, note that there is no tiny bin in the final packing of the algorithm. The bins which include a 1a or four 2a items clearly have weight 1. For all other LM-bins, the weights of the large and medium items are defined in a way to ensure that the accumulated weight of the bin is no less than 1 (except possibly a constant number of bins). A harmonic bin of type $i \geq 3$ includes i^2 items of type i; these items occupy an area at least $1/(i+1)^2$. The total occupied area in these bins is then $i^2/(i+1)^2 \geq 9/16$. Hence, the total weight of these items is at least $9/16 \times \alpha = 1$.

For claim 2, note that if there is no item of type 1d in the bin, the density of all items will be less than 1.84 and consequently their total weight is no more than 1.84. Assume there is an item of type 1d with weight $1 - 0.28\alpha$. There is an available area less than 3/4 for other items. The maximum

Type	Size	Area	Weight	Density
1a	$s(x) \in (4/5, 1]$	$> 16/25$	1	< 1.5625
1b	$s(x) \in (2/3, 4/5]$	$> 4/9$	1	< 2.25
1c	$s(x) \in (3/5, 2/3]$	$> 9/25$	9/16	< 1.5625
1d	$s(x) \in (1/2, 3/5]$	$> 1/4$	9/16	< 2.25
2a	$s(x) \in (2/5, 1/2]$	$> 4/25$	0.25	< 1.5625
2b	$s(x) \in (1/3, 2/5]$	$> 1/9$	0.25	< 2.25
3	$s(x) \in (1/4, 1/3]$	$> 1/16$	1/9	< 1.78
4	$s(x) \in (1/5, 1/4]$	$> 1/25$	1/16	< 1.5625
$5, \ldots, 15$	$s(x) \in (0, 1/15]$	-	0	0

Table 5.3: Characteristics of items of different types in Lemma 23.

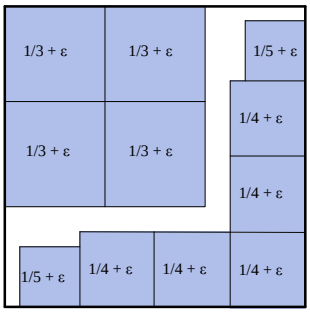
density of items in in this area is α. Consequently, the total weight of items in the bin is less than $1 - 0.28\alpha + 0.75\alpha = 1 + 0.47\alpha < 1.84$ □

Using a similar approach, we prove the following two lemmas.

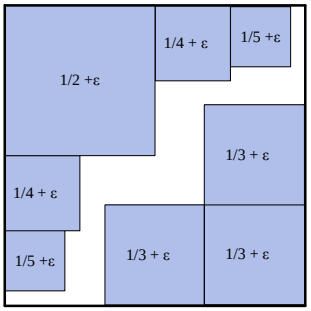
Lemma 23. *Assume there is no tiny bin or harmonic bin of type ≥ 5 in the final packing of the algorithm while there is a harmonic bin of type $i \in \{4, 5\}$. Then the competitive ratio is no more than* 1.84.

Proof. We use a weighting argument as before. The weights of items of types 1a and 1b are 1 while the weight of items of type 2 is 1/4 and the weights of items of type 3 and 4 are respectively 1/9 and 1/16. The weight of all tiny items and small items of types ≥ 5 is 0. For items of types 1c and 1d, we consider the minimum weight of small items accompanied with them in the same bins. Note that 1c and 1d items cannot be single since some harmonic bins of type 3 or 4 are opened. The contributed weight by small items is at least $\min(5 \times 1/9, 7 \times 1/16) = 7/16$. Hence, we define the weights of 1c and 1d items to be 9/16. This way, the weights of all bins in the final packing of the algorithm is at least 1 (see Table 5.3 for a summary of weights and densities).

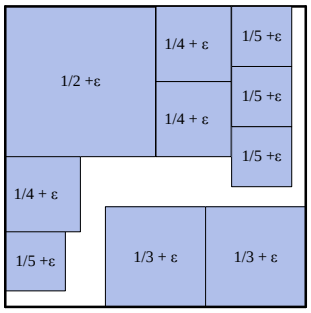
Next, we show that no bin in the packing of Opt can have weight more than 1.84. Assume there is a bin with weight more than 1.84. As entries in Table 5.3 indicate, the bin should contain items of type 1b, 1d, or 2b; otherwise, the density of all items and consequently the weight of the bin will be less than 1.84. First, assume there is no item of type 1d in the bin. Note that items of types 1b and 2b do not fit in the same bin (they have sizes more than 2/3 and 1/3, respectively). Only one item of type 1b or 4 items of type 2b fit in the same bin. In both cases, the contributed weight of non-small items is 1. Moreover, these items occupy an area more than 4/9; hence, there is enough space for at most 5 items of type 3 and two items of type 4 (Figure 5.3a); the weight of the bin will be $1 + 5 \times 1/9 + 2 \times 1/16 \approx 1.68 < 1.84$. Next, assume there is an item of type 1d in the bin. Intuitively, to achieve the maximum total weight, we need to fill the bin with items of high density; however, this results in a lot of empty space (since items with high density are large and do not fit each other in a bin). To be more precise, if there are three items of type 2b in the bin, then there is space for at most two items of type 3 and two items of type 4; the total weight will be $9/16 + 3 \times 0.25 + 2 \times 1/9 + 2 \times 1/16 < 1.84$ (Figure 5.3b). Similarly, when there are respectively two and one items of type 2b in the bin, the total weight of the bin will be at most $9/16 + 2 \times 0.25 + 3 \times 1/9 + 4 \times 1/16 < 1.84$ (Figure 5.3c) and $9/16 + 0.25 + 4 \times 1/9 + 6 \times 1/16 < 1.84$ (Figure 5.3d). If there is no item of type 2b in the bin, there can be at most 5 items of type 3 and 7 items of type 4 (Figure 5.3e). The total weight of the bin will be $9/16 + 5 \times 1/9 + 7 \times 1/16 < 1.84$. □



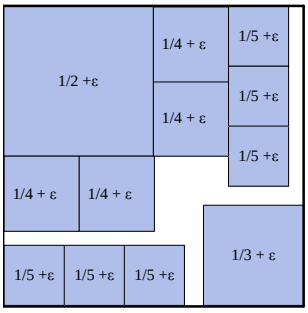
(a) There are items of type 1b or 2b.



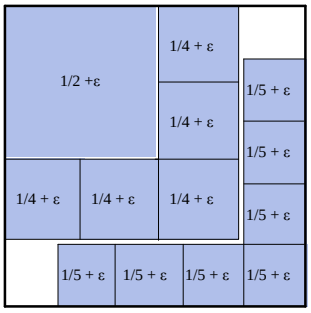
(b) An item of type 1d accompanied by three items of type 2b.



(c) An item of type 1d accompanied by two items of type 2b.



(d) An item of type 1d accompanied by an item of type 2b.



(e) There is an item of type 1d and no item of type 1b.

Figure 5.3: Packings which result in the maximum total weight in a bin of Opt in different cases.

Lemma 24. *Assume there are no tiny or harmonic bins in the final packing of the algorithm. Then the competitive ratio is no more than* 1.75*.*

Proof. We define the weight of items of types 1 and 2 to be respectively 1 and 1/4, while the weights of all other items (i.e., small and tiny items) are 0. Since there is no harmonic or tiny bin, the weights of all bins is 1. Also, no bin in the offline packing has weight more than 1.75. This is because if a bin contains an item of type 1 (size more than 1/2), then it cannot contain more than 3 items of type 2 (size more than 1/3); in this case, the total weight of items is $1 + 3 \times 1/4 = 1.75$. Note that a bin that only contains items of one type (1 or 2) has weight 1. □

From Lemmas 21, 22,23, and 24, we conclude the following theorem:

Theorem 11. *There is an almost online algorithm for the square packing problem which receives advice of size* $\Theta(\lg n)$ *for a sequence of length* n *and achieves a competitive ratio of at most* 1.84*.*

5.3 Remarks

The algorithm introduced in this chapter is expected to be generalized to the cube packing problem with d-dimensional cubes ($d \geq 2$). However, providing almost-online algorithms for the box packing problems seems to be more challenging and we leave it as a future work. Another promising direction is to investigate how many bits of advice are required and sufficient to achieve a 1-competitive algorithm for the square packing problem.

Chapter 6

List Update with Advice

In this chapter, we study the online list update problem under the advice model of computation. We show that advice of linear size is required and sufficient for a deterministic algorithm to achieve an optimal solution or even a competitive ratio better than 15/14. On the other hand, we show that surprisingly two bits of advice are sufficient to break the lower bound of 2 on the competitive ratio of deterministic algorithms and achieve a deterministic algorithm with a competitive ratio of $1.\bar{6}$. For this upper-bound argument, the bits of advice determine the algorithm with smaller cost among three classic online algorithms, TIMESTAMP and two members of the MTF2 family of algorithms. We also show that MTF2 algorithms are 2.5-competitive.

6.1 Introduction

Recall that in the list update problem, the input is a sequence of requests to items of a list which appear in an online manner. A request involves accessing an item in the list. To access an item, an algorithm should linearly probe the list; each probe has a cost of 1, and accessing an item in the ith position results in a cost of i. The goal is to maintain the list in a way to minimize the total cost. An algorithm can make a *free exchange* to move an accessed item somewhere closer to the front of the list. Further, it can make any number of *paid exchanges*, each having a cost of 1, to swap the positions of any two consecutive items in the list.

As mentioned in previous chapters, assuming a total lack of information about the future is unrealistic in many applications. This is particularly the case for the list update problem when it is used as a method for compression (this will be discussed in Chapter 8). Hence, it makes sense to study the problem under the advice framework.

Definition 8. *In the* online list update problem with advice*, the input consists of a list of l items and a sequence $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ of requests to items of the list which appear in an online manner. To serve a request to item x, an online algorithm $\mathbb{A}$ has to probe the list to access x; each probe has a cost of 1 and accessing x at index i has cost i. After accessing x, $\mathbb{A}$ can move it to closer to front using a free exchange. It can also applies any number of paid exchanges, each having cost 1, to swap the positions of any two*

consecutive items in the list. The goal of an online algorithm is to reorganize the list so that the total cost is minimized. The decision of an algorithm for reorganizing the list after accessing σ_t is a function of $\Phi, \sigma_1, \ldots, \sigma_t$, where Φ is the content of the advice tape. An algorithm $\mathbb{A}$ is c-competitive with advice complexity $s(n)$ *if there exists a constant c_0 such that, for all n and for all input sequences σ of length at most n, there exists some advice Φ such that $\mathbb{A}(\sigma) \leq c \ \textsc{Opt}(\sigma) + c_0$, and at most the first $s(n)$ bits of Φ have been accessed by the algorithm. If $c = 1$ and $c_0 = 0$, then $\mathbb{A}$ is* optimal.

Like any other problem, when studying list update under the advice model, the first question to answer is how many bits of advice are required to achieve an optimal solution. We show that advice of size $\textsc{Opt}(\sigma)$ is sufficient to optimally serve a sequence σ, where $\textsc{Opt}(\sigma)$ is the cost of an optimal offline algorithm for serving σ, and it is linear in the length of the sequence, assuming that the length of the list is a constant. We further show that advice of linear size is required to achieve a deterministic algorithm with a competitive ratio better than 15/14.

Another important question is how many bits of advice are required to break the lower bound on the competitive ratio of any deterministic algorithm. We answer this question by introducing a deterministic algorithm that receives two bits of advice and achieves a competitive ratio of at most $1.\bar{6}$. The advice bit for a sequence σ simply indicates the best option between three online algorithms for serving σ. These three algorithms are Timestamp, MTF-Odd (MtfO) and MTF-Even (MtfE). Recall that Timestamp inserts an accessed item x in front of the first item y (from the front of the list) that precedes x in the list and was accessed at most once since the last access to x. If there is no such item y or x is accessed for the first time, no items are moved. MtfO (respectively MtfE) moves a requested item x to the front on every odd (respectively even) request to x.

Our results indicate that if we dismiss Timestamp and take the better algorithm between MtfO and MtfO, the competitive ratio of the resulting algorithm is no better than 1.75. We also study the competitiveness of MtfE and MtfO, and more generally any algorithm that belongs to the family of Move-To-Front-Every-Other-Access (Mtf2) algorithms. We show that these algorithms have competitive ratios of 2.5.

6.2 Optimal Solution

In this section, we provide upper and lower bounds on the number of advice bits required to optimally serve a sequence. We start with an upper bound:

Theorem 12. *Under the advice model, $\textsc{Opt}(\sigma) - n$ bits of advice are sufficient to achieve an optimal solution for any sequence σ of length n, where $\textsc{Opt}(\sigma)$ is the cost of an optimal algorithm for serving σ.*

Proof. It is known that there is an optimal algorithm that moves items using only a family of paid exchanges called *subset transfer* [120]. In a subset transfer, before serving a request to an item x, a subset S of items preceding x in the list is moved (using paid exchanges) to just after x in the list, so that the relative order of items in S among themselves remains unchanged. Consider an optimal algorithm Opt which only moves items via subset transfer. Before a request to x at index i, an online algorithm can read $i - 1$ bits from the advice tape, indicating (bit vector style) the subset which should be moved to after x. Provided with this, the algorithm can always maintain the same list as Opt. The total number of bits read by the algorithm will be at most $\textsc{Opt}(\sigma) - n$. □

The above theorem implies that for lists of constant size, advice of linear size is sufficient to optimally serve a sequence. We show that advice of linear size is also required to achieve any competitive ratio smaller than 15/14. Consider instances of the list update problem on a list of two items x and y which are defined as follows. Assume the list is ordered as $[x, y]$ before the first request. Also, to make the explanation easier, assume that the length of the sequence, n, is divisible by 5. Consider an arbitrary bitstring B, of size $n/5$, which we refer to as the *defining bitstring*. Let σ denote the list update sequence defined from B in the following manner: For each bit in B, there are five requests in σ, which we refer to as a *round*. We say that a round in σ is of type 0 (respectively 1) if the bit associated with it in B is 0 (respectively 1). For a round of type 0, σ will contain the requests $yyyxx$, and for a round of type 1, the requests $yxxxx$. For example, if $B = 011\ldots$, we will have $\sigma = \langle yyyxx, yxxxx, yxxxx, \ldots\rangle$.

Since the last two requests in a round are to the same item x, it makes sense for an online algorithm to move x to the front after the first access. This is formalized in the following lemma.

Lemma 25. *For any online list update algorithm $\mathbb{A}$ serving a sequence σ created from a defining bitstring, there is another algorithm whose cost is not more than $\mathbb{A}$'s cost for serving σ and that ends each round with the list in the order $[x, y]$.*

Proof. Let R_t denote the first round such that the ordering of the list maintained by $\mathbb{A}$ is $[y, x]$ at the end of the round. So, $\mathbb{A}$ incurs a cost of 4 for the last two requests of the round (which are both to x) and a cost of 1 for the first request of the next round (which is to y). This sums to a cost of 5 for these three requests. Consider an alternative algorithm $\mathbb{A}'$ which moves x to the front after the first access to x in R_t. The cost of $\mathbb{A}'$ for the last two requests of R_t is 3. Also, $\mathbb{A}'$ incurs a cost of 2 to access the first request of the next round. Hence, $\mathbb{A}'$ incurs a cost of at most 5, equal to the cost of $\mathbb{A}$ for these three requests. After the access to y in the second position of the list, $\mathbb{A}'$ can re-establish the same ordering $\mathbb{A}$ uses from that point. Consequently, the cost of $\mathbb{A}'$ is not more than $\mathbb{A}$. Repeating this argument for all rounds completes the proof. □

Provided with the above lemma, we can restrict our attention to algorithms that maintain the ordering $[x, y]$ at the end of each round. In what follows, by an 'online algorithm' we mean an online algorithm with this property.

Lemma 26. *The cost of an optimal algorithm for serving a sequence of length n, where the sequence is created from a defining bitstring, is at most $7n/5$.*

Proof. Since there are $n/5$ rounds, it is sufficient to show that there is an algorithm which incurs a cost of at most 7 for each round. Consider an algorithm that works as follows: For a round of type 0, the algorithm moves y to the front after the first access to y. It also moves x to the front after the first access to x. Hence, it incurs a cost 2+1+1+2+1 = 7. For a round of type 1, the algorithm does not move any item and incurs a cost of 2+1+1+1+1 = 6. In both cases, the list ordering is $[x, y]$ at the end of the round and the same argument can be repeated for the next rounds. □

For a round of type 0 (with requests to $yyyxx$), if an online algorithm $\mathbb{A}$ moves each of x and y to the front after the first accesses, it has cost 7. If it does not move y immediately, it has cost at least 8. For a round of type 1 (i.e., a round of requests to $yxxxx$), if an algorithm does no rearrangement, its cost will be 6; otherwise its cost is at least 7. To summarize, an online algorithm should 'guess' the type of each round

and act accordingly after accessing the first request of the round. If the algorithm makes a wrong guess, it incurs a 'penalty' of at least 1 unit. This relates our problem to the Binary String Guessing Problem with Known History (2-SGKH) as defined in Section 1.2. Recall that in 2-SGKH problem, a bitstring is revealed in an online manner and an online algorithm has to guess the content of each bit before it is revealed. Any algorithm that correctly guesses more than half of the input bits must receive an advice of linear size (Lemma 1). We reduce the 2-SGKH problem to the list update problem as follows.

Theorem 13. *On an input of size n, any algorithm for the list update problem which achieves a competitive ratio of γ ($1 < \gamma \leq 15/14$) needs to read at least $(1 + (7\gamma - 7)\lg(7\gamma - 7) + (8 - 7\gamma)\lg(8 - 7\gamma))/5 \times n$ bits of advice.*

Proof. Consider the 2-SGKH problem for an arbitrary bitstring B. Given an online algorithm $\mathbb{A}$ for the list update problem, define an algorithm for 2-SGKH as follows: Consider an instance σ of the list update problem on a list of length 2 where σ has B as its defining bitstring, and run $\mathbb{A}$ to serve σ. For the first request y in each round in σ, $\mathbb{A}$ should decide whether to move it to the front or not. The algorithm for the 2-SGKH problem guesses a bit as being 0 (respectively 1) if, after accessing the first item requested in the round associated with the bit in B, $\mathbb{A}$ moves it to front (respectively keeps it at its position). As mentioned earlier, for each incorrect guess $\mathbb{A}$ incurs a penalty of at least 1 unit, i.e., $\mathbb{A} \geq \textsc{Opt} + w$, where w is the number of wrong guesses for critical requests. Since $\mathbb{A}$ has a competitive ratio of γ, we have $\mathbb{A} \leq \gamma \textsc{Opt}$. Consequently, we have $w \leq (\gamma - 1)\textsc{Opt}(\sigma)$ and by Lemma 26, $w \leq 7(\gamma - 1)/5 \times n$. This implies that if $\mathbb{A}$ has a competitive ratio of γ, the 2-SGKH algorithm makes at most $7(\gamma-1)/5 \times n$ mistakes for an input bitstring B of size $n/5$, i.e., at least $n/5 - 7(\gamma - 1)/5 \times n = (8 - 7\gamma) \times n/5$ correct guesses. Define $\alpha = 8 - 7\gamma$, and note that α is in the range $[1/2, 1)$ when γ is in the range stated in the lemma. By Lemma 1, at least $(1 + (1 - \alpha)\lg(1 - \alpha) + \alpha \lg \alpha)n/5$ bits of advice are required by such a 2-SGKH algorithm. Replacing α with $8 - 7\gamma$ completes the proof. □

Thus, to obtain a competitive ratio better than 15/14, a linear number of bits of advice is required. For example, to achieve a competitive ratio of 1.01, at least $0.12n$ bits of advice are required. Theorems 12 and 13 imply the following corollary.

Corollary 3. *For any fixed list, $\Theta(n)$ bits of advice are required and sufficient to achieve an optimal solution for the list update problem.*

6.3 An Algorithm with Two Bits of Advice

In this section we show that two bits of advice are sufficient to break the lower bound of 2 on the competitive ratio of deterministic algorithms and achieve a deterministic online algorithm with a competitive ratio of $1.\bar{6}$. The two bits of advice for a sequence σ indicate which of the three algorithms Timestamp, MTF-Odd (MtfO) and MTF-Even (MtfE), have the lower cost for serving σ. Recall that MtfO (respectively MtfE) moves a requested item x to the front on every odd (respectively even) request to x. We prove the following theorem:

Theorem 14. *For any sequence σ, we have either $\textsc{Timestamp}(\sigma) \leq 1.\bar{6}\,\textsc{Opt}(\sigma)$, $\textsc{MtfO}(\sigma) \leq 1.\bar{6}\,\textsc{Opt}(\sigma)$, or $\textsc{MtfE}(\sigma) \leq 1.\bar{6}\,\textsc{Opt}(\sigma)$.*

To prove the theorem, we show that for any sequence σ, $\textsc{Timestamp}(\sigma) + \textsc{MtfO}(\sigma) + \textsc{MtfE}(\sigma) \leq 5\,\textsc{Opt}(\sigma)$. We note that all three algorithms have the projective property (defined in Section 2.2.2). Recall that if an algorithm $\mathbb{A}$ has the projective property, then the relative order of any two items in the list maintained by $\mathbb{A}$ only depends on the requests to those items and their initial order in the list (and not on the requests to other items). MtfO (respectively MtfE) is projective since in its list an item y precedes x if and only if the last odd (respectively even) access to y is more recent than the last odd (respectively even) access to x. In the lists maintained by Timestamp, item y precedes item x if and only if in the projected sequence on x and y, y was requested twice after the second to last request to x or the most recent request was to y and x has been requested at most once. Hence, Timestamp also has the projective property.

Similar to most other work for analysis of projective algorithms, we consider the partial cost model in which accessing an item in position i is defined to have cost $i-1$. We say an algorithm is *cost independent* if its decisions are independent of the cost it has paid for previous requests. The cost of any cost independent algorithm for serving a sequence of length n decreases n units under the partial cost model when compared to the full cost model. Hence, any upper bound for the competitive ratio of a cost independent algorithm under the partial cost model can be extended to the full cost model.

To prove an upper bound on the competitive ratio of a projective algorithm under the partial cost model, it is sufficient to prove that the claim holds for lists of size 2. The reduction to lists of size two is done by applying a *factoring lemma* which ensures that the total cost of a projective algorithm $\mathbb{A}$ for serving a sequence σ can be formulated as the sum of the costs of $\mathbb{A}$ for serving projected sequences of two items. A projected sequence of σ on two items x and y is a copy of σ in which all items except x and y are removed. We refer the reader to [39, p. 16] for details on the factoring lemma. Since MtfO, MtfE, and Timestamp are projective and cost independent, to prove Theorem 14, it suffices to prove the following lemma:

Lemma 27. *Under the partial cost model, for any sequence σ_{xy} of two items, we have* $\textsc{MtfO}(\sigma_{xy}) + \textsc{MtfE}(\sigma_{xy}) + \textsc{Timestamp}(\sigma_{xy}) \leq 5 \times \textsc{Opt}(\sigma_{xy})$.

Before proving the above lemma, we study the aggregated cost of MtfO and MtfE on certain subsequences of two items. One way to think of these algorithms is to imagine they maintain a bit for each item. On each request, the bit of the item is flipped; if it becomes '0', the item is moved to the front. Note that the bits of MtfO and MtfE are complements of each other. Thus, we can think of them as one algorithm started on complementary bit sequences. We say a list is in state $[ab]_{(i,j)}$ if item a precedes b in the list and the bits maintained for a and b are i and j ($i, j \in \{0, 1\}$), respectively. To study the value of $\textsc{Opt}(\sigma_{xy})$, we consider an offline algorithm which uses a free exchange to move an accessed item from the second position to the front of the list if and only if the following request is to the same item. It is known that this algorithm is optimal for lists of two items [120].

Lemma 28. *Consider a subsequence of two items a and b of the form $\langle (ba)^{2i} \rangle$, i.e., i repetitions of $\langle baba \rangle$. Assume the initial ordering is $[ab]$. The cost of each of* MtfO *and* MtfE *for serving the subsequence is at most $3i$ (under the partial cost model). Moreover, at the end of serving the subsequence, the ordering of items in the list maintained by at least one of the algorithms is $[ab]$.*

Proof. We refer to repetition of *baba* as a *round*. We show that MtfO and MtfE have a cost of at most 3 for serving each round. Assume the bits associated with both items are '0' before serving *baba*. The

Bits for (a,b)	Cost for $\langle baba \rangle$	Orders before accessing items	Final order
$(0,0)$	$1+0+1+1=3$	$[a\overset{\downarrow}{b}]\ [\overset{\downarrow}{a}b]\ [a\overset{\curvearrowleft}{b}]\ [\overset{\curvearrowleft}{b}a]$	$[ab]$
$(0,1)$	$1+1+0+1=3$	$[\overset{\curvearrowleft}{a}b]\ [b\overset{\downarrow}{a}]\ [\overset{\downarrow}{b}a]\ [\overset{\curvearrowleft}{b}a]$	$[ab]$
$(1,0)$	$1+0+1+1=3$	$[a\overset{\downarrow}{b}]\ [\overset{\downarrow}{a}b]\ [\overset{\curvearrowleft}{a}b]\ [b\overset{\downarrow}{a}]$	$[ba]$
$(1,1)$	$1+1+1+0=3$	$[\overset{\curvearrowleft}{a}b]\ [\overset{\curvearrowleft}{b}a]\ [\overset{\downarrow}{a}b]\ [\overset{\downarrow}{a}b]$	$[ab]$

Table 6.1: Assuming the initial ordering of items is $[ab]$, the cost of a both MTFO and MTFE for serving subsequence $\langle baba \rangle$ is at most 3 (under the partial cost model). The final ordering of the items will be $[ab]$ in three of the cases.

first request has a cost of 1 and b remains in the second position, the second request has cost 0, and the remaining requests each have a cost of 1. In total, the cost of the algorithm is 3. The other cases (when items have different bits) are handled similarly. Table 6.1 includes a summary of all cases. As illustrated in the table, if the bits maintained for a and b before serving $baba$ are $(0,0)$, (0,1), or (1,1), the list order will be $[ab]$ after serving the round. Since both a and b are requested twice, the bits will be also the same after serving $baba$. Hence, in these three cases, the same argument can be repeated to conclude that the list order will be $[ab]$ at the end of serving $(ba)^{2i}$. Since the bits maintained for the items are complements in MTFE and MTFO, at least one of them starts with bits $(0,0)$, $(0,1)$, or $(1,1)$ for a and b; consequently, at least one algorithm ends up with state $[ab]$ at the end. □

Lemma 29. *Consider a subsequence of two items a and b which has form $\langle baa \rangle$. The total cost that* MTFE *and* MTFO *incur together for serving this subsequence is less than or equal to 4 (under the partial cost model).*

Proof. If the initial order of a and b is $[ba]$, the first request has no cost, and each algorithm incurs a total cost of at most 2 for the other two requests of the sequence. Hence, the aggregated cost of the two algorithms is 4. Next, assume the initial order is $[ab]$. Assume the bits maintained by one of the algorithms for a and b are (1,0), respectively. As illustrated in Table 6.2, this algorithm incurs a cost of 1 for serving baa; the other algorithm incurs a cost of 3. In total, the algorithms incur a cost of 4. In the other case, when bits maintained for a and b are both '0' in one algorithm (consequently, both are '1' in the other algorithm), the total cost of the algorithms for serving $\langle baa \rangle$ is 3. □

Using Lemmas 28 and 29, we are ready to prove Lemma 27:

Proof of Lemma 27, and consequently Theorem 14. Consider a sequence σ_{xy} of two items x and y. We use the *phase partitioning technique* as discussed in [39]. We partition σ_{xy} into *phases* which are defined inductively as follows. Assume we have defined phases up until, but not including, the tth request ($t \geq 1$) and the relative order of the two items is $[xy]$ before the tth request. Then the next phase is of *type 1* and is of one of the following forms ($j \geq 0$ and $k \geq 1$):

$$(a)\ x^j yy \quad (b)\ x^j (yx)^k yy \quad (c)\ x^j (yx)^k x$$

In case the relative order of the items is $[yx]$ before the tth request, the phase has type 2 and its form is exactly the same as above with x and y interchanged. Note that, after two consecutive requests to an

Initial order	Bits for (a,b)	Cost for $\langle baa \rangle$	Orders before accessing items	Bits and Costs (other algorithm)	Total cost (both algs.)
$[ab]$	(0,0)	$1+0+0=1$	$[\overset{\downarrow}{a}b]$ $[\overset{\downarrow}{a}b]$ $[\overset{\downarrow}{a}b]$	$(1,1) \to 2$	$1+2=3$
$[ab]$	(0,1)	$1+1+1=3$	$[\overset{\curvearrowleft}{a}b]$ $[b\overset{\downarrow}{a}]$ $[\overset{\curvearrowleft}{b}a]$	$(1,0) \to 1$	$3+1=4$
$[ab]$	(1,0)	$1+0+0=1$	$[\overset{\downarrow}{a}b]$ $[\overset{\downarrow}{a}b]$ $[\overset{\downarrow}{a}b]$	$(0,1) \to 3$	$1+3=4$
$[ab]$	(1,1)	$1+1+0=2$	$[\overset{\curvearrowleft}{a}b]$ $[\overset{\curvearrowleft}{b}a]$ $[\overset{\downarrow}{a}b]$	$(0,0) \to 1$	$2+1=3$
$[ba]$	(0,0) (0,1) (1,0) (1,1)	$\leq 0+1+1=2$	-	≤ 2	$2+2=4$

Table 6.2: The total cost of MTFO and MTFE for serving a sequence $\langle baa \rangle$ is at most 4 (under the partial cost model). Note that the bits of these algorithms for each item are complements of each other.

item, TIMESTAMP, MTFO and MTFE all have that item in the front of the list. So, after serving each phase, the relative order of items is the same for all three algorithms. This implies that σ_{xy} is partitioned in the same way for all three algorithms. To prove the lemma, we show that its statement holds for every phase.

Table 6.3 shows the costs incurred by all three algorithms as well as OPT for each phase. Note that phases of the form (b) and (c) are divided into two cases, depending on whether k is even or odd. We discuss the different phases of type 1 separately. Similar analyses, with x and y interchanged, apply to the phases of type 2. Note that before serving a phase of type 1, the list is ordered as $[xy]$ and the first j requests to x have no cost.

Consider phases of form (a), $x^j yy$. MTFO and MTFE incur a total cost of 3 for serving yy (one of them moves y to the front after the first request, while the other keeps it in the second position). TIMESTAMP incurs a cost of 2 for serving yy (it does not move it to the front after the first request). So, in total, the three algorithms incur an aggregated cost of 5. On the other hand, OPT incurs a cost of 1 for the phase. So, the ratio between the sum of the costs of the algorithms and the cost of OPT is 5.

Next, consider phases of the form (b). TIMESTAMP incurs a cost of $2k$ for serving the phase; it incurs a cost of 1 for all requests in $(yx)^{2i}$ except the very first one, and a cost of 1 for serving the second to last request to y. Assume k is even and we have $k = 2i$ for some $i \geq 1$, so the phase looks like $x^j(yx)^k yy$. By Lemma 28, the cost incurred by MTFO and MTFE is at most $3i$ for serving $(yx)^{2i}$. We show that for the remaining two requests to y, MTFO and MTFO incur an aggregated cost of at most 3. If the list maintained by any of the algorithms is ordered as $[yx]$ before serving yy, that algorithm incurs a cost of 0 while the other algorithm incurs a cost of at most 2 for these requests; in total, the cost of both algorithms for serving yy will be at most 2. If the lists of both algorithms are ordered as $[xy]$, one of the algorithms incurs a cost of 1 and the other incurs a cost of 2 (depending on the bit they keep for y). In conclusion, MTFO and MTFE incur a total cost of at most $6i+3$. TIMESTAMP incurs a cost of $2k = 4i$, while OPT incurs a cost of $2i+1$ for the phase. To conclude, the aggregated cost of all algorithms is at most $10i+3$ compared to $2i+1$ for OPT, and the ratio between them is less than 5.

Next, assume k is odd and we have $k = 2i-1$, i.e., the phase has the form $x^j(yx)^{2i-2}yxyy$. The total cost of MTFO and MTFE for $(yx)^{2i-2}$ is at most $2 \times (3(i-1))$ (Lemma 28), the total cost for the next request to y is at most 2, and the total cost for subsequent xyy is at most 4 (Lemma 29). In total, MTFO and MTFE incur a cost of at most $6i$ for the phase. On the other hand, TIMESTAMP incurs a cost of $4i-2$

Phase	AlgMin	AlgMax	Timestamp	Sum (AlgMin + AlgMax + Timestamp)	Opt'	$\frac{\text{Sum}}{\text{Opt}'}$
$x^j yy$	1	2	2	5	1	5
$x^j(yx)^{2i}yy$	$\leq 3i+1$	$\leq 3i+2$	$2\times 2i = 4i$	$\leq 10i+3$	$2i+1$	< 5
$x^j(yx)^{2i-2}yxyy$	$\leq 3(i-1)+1$ $+$ AlgMin$(\langle xyy\rangle)$	$\leq 3(i-1)+1$ $+$ AlgMax$(\langle xyy\rangle)$	$2\times(2i-1)$ $= 4i-2$	$\leq 6(i-1)+2+4$ $+(4i-2) = 10i-2$	$2i$	< 5
$x^j(yx)^{2i}x$	$\leq 3i$	$\leq 3i+1$	$2\times 2i-1$ $= 4i-1$	$\leq (6i+1)+(4i-1)$ $= 10i$	$2i$	≤ 5
$x^j(yx)^{2i-2}yxx$	$\leq 3(i-1)$ $+$ AlgMin$(\langle yxx\rangle)$	$\leq 3(i-1)$ $+$ AlgMax$(\langle yxx\rangle)$	$2\times(2i-1)-1$ $= 4i-3$	$\leq 6(i-1)+4$ $+(4i-3) = 10i-5$	$2i-1$	≤ 5

Table 6.3: The costs of MtfO, MtfE, and Timestamp for a phase of type 1 (the phase has type 1, i.e., the initial ordering of items is xy). The ratio between the aggregated cost of algorithms and the cost of Opt for each phase is at most 5. AlgMin (respectively AlgMax) is the algorithm among MtfO and MtfE, which incurs less (respectively more) cost for the phase. Note that the costs are under the partial cost model.

for the phase. The aggregated cost of the three algorithms is at most $10i-2$ for the phase, while Opt incurs a cost of $2i$. So, the ratio between sum of the costs of the algorithms and Opt is less than 5.

Next, consider phases of type 1 and form (c). Timestamp incurs a cost of $2k-1$ in this case. Assume k is even, i.e., the phase has the form $x^j(yx)^{2i}x$. By Lemma 28, MtfO and MtfE each incur a total cost of at most $3i$ for $(yx)^{2i}$. Moreover, after this, the list maintained for at least one of the algorithms is ordered as $[xy]$. Hence, the aggregated cost of algorithms for the next request to x is at most 1. Consequently, the total cost of MtfE and MtfO is at most $6i+1$ for the round. Adding the cost $2k-1 = 4i-1$ of Timestamp, the total cost of all three algorithms is at most $10i$. On the other hand, Opt incurs a cost of $2i$ for the phase. So, the ratio between the aggregated cost of all three algorithms and the cost of Opt is at most 5. Finally, assume k is odd, i.e., the phase has form $x^j(yx)^{2i-2}yxx$. By Lemma 28, MtfO and MtfE together incur a total cost of $2\times 3(i-1)$ for $x^j(yx)^{2i-2}$. By Lemma 29, they incur a total cost of at most 4 for yxx. In total, they incur a cost of at most $6(i-1)+4$ for the phase. Timestamp incurs a cost of $4i-3$; this sums up to $10i-5$ for all three algorithms. In this case, Opt incurs a cost of $2i-1$. Hence, the ratio between the sum of the costs of all three algorithms and Opt is at most 5. □

In fact, the upper bound provided in Theorem 3 for the competitive ratio of the better algorithm among Timestamp, MtfO and MtfE is tight under the partial cost model. To show this, we make use of the following lemma.

Lemma 30. *Consider a sequence $\sigma_\alpha = \langle x(yxxx\ yxxx)^k\rangle$, i.e., a single request to x, followed by k repetitions of $(yxxx\ yxxx)$. Assume the list is initially ordered as $[xy]$. We have $\text{MtfO}(\sigma) = \text{MtfE}(\sigma) = 4k$ while $\text{Opt}(\sigma) = 2k$ (under the partial cost model).*

Proof. We refer to each repetition of $(yxxx\ yxxx)$ as a round. Initially, the bits maintained by MtfO (respectively MtfE) for x, y are $(1,1)$ (respectively (0,0)). After the first request to x, the bits of MtfO (respectively MtfE) change to $(0,1)$ (respectively (1,0)) for x, y. MtfO incurs a cost of 3 for the first half of each round; it incurs a cost of 1 for all requests except the last request to x. MtfE incurs a cost of 1 for serving the first half of a round; it only incurs a cost of 1 on the first requests y. After serving the first half, the list for each algorithm will be ordered as $[xy]$ and the bits maintained by MtfO (respectively

MTFE) for x, y will be $(1, 0)$ (respectively (0,1)). Using a symmetric argument, the costs of MTFO and MTFE for the second half of a round are respectively 1 and 3. In total, both MTFO and MTFE incur a cost of 4 for each round. After serving the round, the list maintained by both algorithms will be ordered as $[xy]$ and the bits associated with the items will be the same as at the start of the first round. Thus, MTFO and MTFE each have a total cost of $4k$ on σ_α. A summary of actions and costs of MTFO and MTFE can be stated as follows (the numbers below the arrows indicate the costs of requests on top, and the numbers on top of x and y indicate their bits):

$$[\overset{0\,1}{xy}] \xrightarrow[1]{y} [\overset{0\,0}{yx}] \xrightarrow[1]{x} [\overset{0\,1}{yx}] \xrightarrow[1]{x} [\overset{0\,0}{xy}] \xrightarrow[0]{x} [\overset{1\,0}{xy}] \xrightarrow[1]{y} [\overset{1\,1}{xy}] \xrightarrow[0]{x} [\overset{0\,1}{xy}] \xrightarrow[0]{x} [\overset{1\,1}{xy}] \xrightarrow[0]{x} [\overset{0\,1}{xy}]$$
$$[\overset{1\,0}{xy}] \xrightarrow[1]{y} [\overset{1\,1}{xy}] \xrightarrow[0]{x} [\overset{0\,1}{xy}] \xrightarrow[0]{x} [\overset{1\,1}{xy}] \xrightarrow[0]{x} [\overset{0\,1}{xy}] \xrightarrow[1]{y} [\overset{0\,0}{yx}] \xrightarrow[1]{x} [\overset{0\,1}{yx}] \xrightarrow[1]{x} [\overset{0\,0}{xy}] \xrightarrow[0]{x} [\overset{1\,0}{xy}]$$

An optimal algorithm OPT never changes the ordering of the list and has a cost of 2 for the whole round, giving a cost of $2k$ for σ_α. □

Theorem 15. *There are sequences for which the costs of all of* TIMESTAMP, MTFE, *and* MTFO *are* $1.\bar{6}$ *times that of* OPT *(under the partial cost model).*

Proof. Consider a sequence $\sigma = \sigma_\alpha\sigma_\beta$ where $\sigma_\alpha = x(yxxx\ yxxx)^{k_\alpha}$ and $\sigma_\beta = (yyxx)^{k_\beta}$. Here, k_α is an arbitrary large integer and $k_\beta = 2k_\alpha$. By Lemma 30, we have $\text{MTFO}(\sigma_\alpha) = \text{MTFE}(\sigma_\alpha) = 4k_\alpha$ while $\text{OPT}(\sigma_\alpha) = 2k_\alpha$. We have $\text{TIMESTAMP}(\sigma_\alpha) = 2k_\alpha$, because it does not move y from the second position.

Next, we study the cost of MTFO and MTFE for serving σ_β. Note that after serving σ_α, the lists maintained by these algorithms are all ordered as $[xy]$ and the bits associated with x and y are respectively $(0, 1)$ for MTFO and $(1, 0)$ for MTFE (see the proof of Lemma 30).We show that for each round $yyxx$ of σ_β, the cost of each algorithm is 3. On the first request to y, MTFO moves it to the front (since the bit maintained for y is 1); so it incurs a cost of 1 for the first requests to y. On the first request to x, MTFO keeps x in the second position; hence, it incurs a cost of 2 for the requests to x. In total, it has a cost of 3 for the round. With a similar argument, MTFE incurs a cost of 2 for the requests to y and a cost of 1 for the requests to x and a total cost of 3. The list order and bits maintained for the items will be the same at the end of the round as at the start. Hence, the same argument can be extended to other rounds to conclude that the cost of both MTFE and MTFO for serving σ_β is $3k_\beta$. On the other hand, TIMESTAMP incurs a cost of 4 on each round as it moves items to the front on the second consecutive request to them; hence, the cost of TIMESTAMP for serving σ_β is $4k_\beta$. An algorithm that moves items in front on the first of two consecutive request to them will incur a cost of 2 on each round; hence, the cost of OPT for serving σ_β is at most $2k_\beta$.

To summarize, the cost of each of MTFO and MTFE for serving σ is $4k_\alpha + 3k_\beta = 10k_\alpha$ while the cost of TIMESTAMP is $2k_\alpha + 4k_\beta = 10k_\alpha$, and the cost of OPT is $2k_\alpha + 2k_\beta = 6k_\alpha$. As a consequence, all three algorithms have a cost which is $10/6 = 1.\bar{6}$ times that of OPT. □

The above lower bound cannot be easily extended to the full cost model. In what follows, we provide a lower bound of 1.6 for the competitive ratio of the better algorithm among TIMESTAMP, MTFO, and MTFE. We start with the following lemma:

Lemma 31. *Consider a list of l items which is initially ordered as $[a_1, a_2, \ldots, a_l]$. Consider the following sequence of requests:*

$$\sigma_\beta = \left\langle (a_1, a_2, ..., a_l, a_1^2, a_2^2, ..., a_l^2, a_l, a_{l-1}, ..., a_1, a_l^2, a_{l-1}^2, ..., a_1^2)^m \right\rangle$$

Assuming that l is sufficiently large, under the full cost model, we have:

$$\textsc{MtfO}(\sigma_\beta) = \textsc{MtfE}(\sigma_\beta) = m(3.5l^2 + o(l^2))$$

and

$$\textsc{Timestamp}(\sigma_\beta) = m(2l^2 + o(l^2)).$$

Proof. Define a phase to be a subsequence of requests which forms one of the m repetitions in σ_β. We calculate the costs of the algorithms for each phase. Note that each phase contains an even number of requests to each item. Also, if $i < j$, so item a_i precedes item a_j in the initial ordering of the list, then, in each phase, a_i is requested twice after the last request to a_j. Each algorithm moves a_i in front of a_j on the first or second of these requests. Thus, the state of the list maintained by all algorithms is the same as with the initial ordering after serving a phase.

Each of the three algorithms incurs a cost of $l(l+1)/2$ for serving $a_1, a_2, \ldots, a_l$ at the beginning of a phase. MtfO moves items to the front, reversing the list, but MtfE and Timestamp do not move the items. For serving the subsequent requests to $a_1^2, a_2^2, ..., a_l^2$, MtfO incurs a cost of $2l^2$ since it does not move items to the front on the first of the two consecutive requests to an item. MtfE and Timestamp move to the front at the first of the consecutive requests and incur a cost of $l(l+1)/2 + l$ (the second request is to front of the list). At this point, for all three algorithms, the list is in the reverse of the initial ordering since for $i < j$ there have been two consecutive requests to a_j after the last request to a_i. Also, the bits maintained by MtfE and MtfO are flipped compared to the beginning of the phase (since there have been three requests to each item). Thus, for the second half of the list, MtfE and MtfO reverse roles. For the next requests to $a_l, a_{l-1}, \ldots, a_1$, only MtfE reverses the list, and each of the three algorithms incurs a cost of $l(l+1)/2$. Consequently, for the remaining requests to $a_l^2, a_{l-1}^3, \ldots, a_1^2$, MtfE incurs a cost of $2l^2$, while MtfO and Timestamp each incur a cost of $l(l+1)/2 + 2l$.

To summarize, the costs of both MtfO and MtfE for each phase is $3.5l^2 + o(l^2)$, while the cost of Timestamp is $2l^2 + o(l^2)$. The actions and costs of the algorithms can be summarized as following (as before, the numbers below arrows indicate the cost for serving the sequence on top, and the numbers on top of items indicate the bits maintained by MtfO and MtfE). The three lines correspond to MtfE, MtfO, and Timestamp, respectively.

$$[\overset{0}{a_1} \ldots \overset{0}{a_l}] \xrightarrow[l^2/2+o(l^2)]{a_1 \ldots a_l} [\overset{1}{a_1} \ldots \overset{1}{a_l}] \xrightarrow[l^2/2+o(l^2)]{a_1^2 \ldots a_l^2} [\overset{1}{a_l} \ldots \overset{1}{a_1}] \xrightarrow[l^2/2+o(l^2)]{a_l \ldots a_1} [\overset{0}{a_1} \ldots \overset{0}{a_l}] \xrightarrow[2l^2+o(l^2)]{a_l^2 \ldots a_1^2} [\overset{0}{a_1} \ldots \overset{0}{a_l}]$$

$$[\overset{1}{a_1} \ldots \overset{1}{a_l}] \xrightarrow[l^2/2+o(l^2)]{a_1 \ldots a_l} [\overset{0}{a_l} \ldots \overset{0}{a_1}] \xrightarrow[2l^2+o(l^2)]{a_1^2 \ldots a_l^2} [\overset{0}{a_l} \ldots \overset{0}{a_1}] \xrightarrow[l^2/2+o(l^2)]{a_l \ldots a_1} [\overset{1}{a_l} \ldots \overset{1}{a_1}] \xrightarrow[l^2/2+o(l^2)]{a_l^2 \ldots a_1^2} [\overset{1}{a_1} \ldots \overset{1}{a_l}]$$

$$[a_1 \ldots a_l] \xrightarrow[l^2/2+o(l^2)]{a_1 \ldots a_l} [a_1 \ldots a_l] \xrightarrow[l^2/2+o(l^2)]{a_1^2 \ldots a_l^2} [a_l \ldots a_1] \xrightarrow[l^2/2+o(l^2)]{a_l \ldots a_1} [a_l \ldots a_1] \xrightarrow[l^2/2+o(l^2)]{a_l^2 \ldots a_1^2} [a_1 \ldots a_l]$$

□

The sequence σ_β of the above lemma shows that using one bit of advice to decide between using MTFE and MTFO gives a competitive ratio of at least 1.75, but TIMESTAMP serves σ_β optimally. Next, we introduce sequences for which TIMESTAMP performs significantly worse than both MTFO and MTFE.

Lemma 32. *Consider a list of l items which is initially ordered as $[a_1, a_2, \ldots, a_l]$. Consider the following sequence of requests:*

$$\sigma_\gamma = \left\langle (a_l^3, a_2^3, ..., a_1^3)^{2s} \right\rangle .$$

Assuming that l is sufficiently large, under the full cost model, we have:

$$\text{MTFO}(\sigma_\beta) = \text{MTFE}(\sigma_\beta) = s(3l^2 + o(l^2))$$

and

$$\text{TIMESTAMP}(\sigma_\beta) = s(4l^2 + o(l^2))$$

and

$$\text{OPT}(\sigma_\beta) = s(2l^2 + o(l^2)).$$

Proof. Define a phase to be two consecutive repetitions of the subsequence in parentheses. We calculate the costs of the algorithms for each phase. Note that there are an even number of requests in each phase, and for $i < j$, there are (actually more than) two consecutive requests to a_i after the last request to a_j. So the list orderings and bits maintained by MTFO and MTFE are the same for each algorithm before and after serving each phase. Similarly, after serving the first half of a phase (the subsequence in parentheses), the lists of all three algorithms are the same as the initial ordering.

An optimal algorithm applies the MTF strategy and incurs a cost of $2l^2+4l$. More precisely, for serving each half of the phase, it incurs a cost of l^2 for the first of three consecutive requests to each item, and a total cost of $2l$ for the second and third requests. TIMESTAMP moves items to the front on the second of three consecutive requests. In each half of a phase, it incurs a total cost of $2l^2$ for the first two requests to items and a cost of l for the third requests. In total, it incurs a cost of $4l^2 + 2l$ for each phase. For the first half of the phase, MTFO moves items to front on the first request to each item, while MTFE does so on the second requests. Hence, MTFO and MTFE respectively incur a cost of $l^2 + 2l$ and $2l^2 + l$ for the first half. For the second half, the bits maintained by the algorithms are flipped, while the list ordering is the same as the initial ordering. Hence, MTFO and MTFE respectively incur a cost of $2l^2 + l$ and $l^2 + 2l$ for the second half. In total the costs of each of MTFO and MTFE for each phase is $3l^2 + 3l$. Since the cost of all algorithms are the same for all phases, the statement of the lemma follows. The actions and costs of the algorithms for each phase can be summarized as follows:

$$\begin{aligned}
\text{MTFE}: &[\overset{0}{a_1} \ldots \overset{0}{a_l}] \xrightarrow[2l^2+o(l^2)]{a_l^3 \ldots a_1^3} [\overset{1}{a_1} \ldots \overset{1}{a_l}] \xrightarrow[l^2+o(l^2)]{a_l^3 \ldots a_1^3} [\overset{0}{a_1} \ldots \overset{0}{a_l}] \\
\text{MTFO}: &[\overset{1}{a_1} \ldots \overset{1}{a_l}] \xrightarrow[l^2+o(l^2)]{a_l^3 \ldots a_1^3} [\overset{0}{a_1} \ldots \overset{0}{a_l}] \xrightarrow[2l^2+o(l^2)]{a_l^3 \ldots a_1^3} [\overset{1}{a_1} \ldots \overset{1}{a_l}] \\
\text{TIMESTAMP}: &[a_1 \ldots a_l] \xrightarrow[2l^2+o(l^2)]{a_l^3 \ldots a_1^3} [a_1 \ldots a_l] \xrightarrow[2l^2+o(l^2)]{a_l^3 \ldots a_1^3} [a_1 \ldots a_l] \\
\text{OPT}: &[a_1 \ldots a_l] \xrightarrow[l^2+o(l^2)]{a_l^3 \ldots a_1^3} [a_1 \ldots a_l] \xrightarrow[l^2+o(l^2)]{a_l^3 \ldots a_1^3}
\end{aligned}$$

□

We use the above two lemmas to prove the following theorem:

Theorem 16. *The competitive ratio of the better algorithm between* MTFE, MTFO, *and* TIMESTAMP *is at least 1.6 under the full cost model.*

Proof. Consider the sequence $\sigma = \sigma_\beta\sigma_\gamma$, i.e., the concatenation of the sequences σ_β and σ_γ as defined in Lemmas 31 and 32. Recall that these sequences consist of m and s phases, respectively. In defining σ, consider values of s which are multiples of 3, and let $m = 2s/3$.

Assume the initial ordering is also the same as the one stated in the lemmas, and recall that the state of the algorithms (list ordering and bits of MTFO and MTFE) are the same at the end of serving σ_β. The costs of each of MTFE and MTFO for serving σ_β is $3.5l^2m + o(l^2m) = \frac{7}{3}l^2s + o(l^2s)$ (by Lemma 31), while they incur a cost of $3l^2s + o(l^2s)$ for σ_γ (by Lemma 32). In total, each of these two algorithms incurs a cost of $\frac{16}{3}l^2s + o(l^2s)$ for σ. On the other hand, TIMESTAMP incurs a cost of $2l^2m + o(l^2m) = \frac{4}{3}l^2s + o(l^2s)$ for σ_β and a cost of $4l^2s + o(l^2s)$ for σ_γ. In total, its cost for σ is $\frac{16}{3}l^2s + o(l^2s)$. Note that all three algorithms have the same costs for serving σ. The cost of OPT for serving σ_β is at most $2l^2m + o(l^2m) = \frac{4}{3}l^2s + o(l^2s)$ (by Lemma 31)), while it has a cost of $2l^2s + o(l^2s)$ for serving σ_γ. In total, the cost of OPT is $\frac{10}{3}l^2s + o(l^2s)$. Comparing this with the cost of $16/3l^2s + o(l^2s)$ of the three algorithms, we conclude that the minimum competitive ratio is at least 1.6. □

Thus, the competitive ratio of the best of the three algorithms, MTFO, MTFE, and TIMESTAMP, is at least 1.6 and at most $1.\bar{6}$. We concluded after Lemma 31 that the competitive ratio of the better of MTFO and MTFE is at least 1.75. Here, we show that the competitive ratio of the better algorithm among MTFO and MTFE is at most 2, using the potential function method.

Lemma 33. *For any sequence σ of length n, we have*

$$\text{MTFO}(\sigma) + \text{MTFE}(\sigma) \leq 4\,\text{OPT}(\sigma).$$

Proof. Consider an algorithm $\mathbb{A} \in \{\text{MTFO}, \text{MTFE}\}$. At any time t (i.e., before serving the tth request), we say a pair (a, b) of items forms an *inversion* if a appears before b in the list maintained by $\mathbb{A}$ while b appears before a in the list maintained by OPT. We define the *weight* of an inversion (a, b) to be 1, if the bit maintained by $\mathbb{A}$ for b is 1, and 2 otherwise. Intuitively, the weight of an inversion is the number of accesses to the latter of the two items in $\mathbb{A}$'s list before the item is moved to the front and the inversion disappears.

We define the potential, Φ_t, at each time t to be the total weight of the inversions in the list maintained by MTFO plus the total weight of the inversions in the list maintained by MTFE.

We consider the events that involve costs and change the potential function. An *online* event is the processing of a request by both MTFO and MTFE. An *offline* event is OPT making a paid exchange. The latter is not directly associated with a request, and we define the cost of MTFO and MTFE in connection with this event to be zero, but there may be a change in the potential function.

For an event at time t, we define the amortized cost a_t to be the total cost paid by MTFO and MTFE together for processing the request (if any), plus the increase in potential due to that processing, i.e., $a_t = \text{MTFO}_t + \text{MTFE}_t + \Phi_t - \Phi_{t-1}$. So the total cost of MTFO and MTFE for serving a sequence σ is $\sum_t a_t - (\Phi_{last} - \Phi_0)$. The maximum possible value of $\Phi_{last} - \Phi_0$ is independent of the length of the

sequence. Hence, to prove the competitiveness of MTFO and MTFE together, it is enough to bound the amortized cost relative to OPT's cost. Let OPT_t be the cost paid by OPT at event t. To prove the lemma, it suffices to show that for each event, we have $a_t \leq 4\,\text{OPT}_t$.

Note that one may assume that OPT only does paid exchanges, no free ones [120]. Consider MTFO and MTFE for an *online event*. Let $\mathbb{A}$ be the algorithm that moves y to the front, while A' is the other algorithm, i.e., the one that keeps it at its current position. Assume $\mathbb{A}$ accesses y at index k while A' finds it at index k'. Also, let j denote the index of y in the list maintained by OPT.

We first show that the contribution by $\mathbb{A}$ to the amortized cost is at most $k - (k - j) + 2j = 3j$. The first term (k) is the access cost for $\mathbb{A}$. Before moving y to front, there are at least $k - j$ inversions with y for $\mathbb{A}$ involving items which occur before y, each having a weight of 1 (since the bit of y in $\mathbb{A}$ has been 1 as it moves y to front). All these inversions are removed after moving y to front. This gives the second term in the amortized cost, i.e., $-(k - j)$. Moving y to the front creates at most j new inversions, each having a weight of at most 2, which results in a total increase of $2j$ in the potential. Next, we show that the contribution by A' to the amortized cost is at most $k' - (k' - j) = j$. This is because, after accessing y at index k', there are at least $k' - j$ inversions with y for A' involving items which occur before y. Since A' does not move y to the front, the bit of y was 0, i.e., all these inversions had weight 2. After the access, the bit of y becomes 1 and the weights of these inversions decreases 1 unit. To summarize, the amortized cost a_t is at most $3j + j = 4j$. Since OPT accesses y at index j, we have $\text{OPT}_j = j$ and consequently $a_t \leq 4\,\text{OPT}_t$.

Next, consider an *offline event* where OPT makes a paid exchange. In doing so, it incurs a cost of 1 and $\text{OPT}_t = 1$. This single exchange might create an inversion in the list of MTFO and an inversion in the list of MTFE. Each of these inversions have a weight of at most 2. So, the total increase in the potential is at most 4, i.e., $a_t \leq 4$. Consequently, $a_t \leq 4\,\text{OPT}_t$. □

The above lemma implies that the better algorithm between MTFO and MTFE has a competitive ratio of at most 2. By Lemma 31, such an algorithm has a competitive ratio of at least 1.75.

Theorem 17. *The competitive ratio of the better algorithm between* MTFO *and* MTFE *is at least 1.75 and at most 2.*

6.4 Analysis of Move-To-Front-Every-Other-Access

In the previous sections, we have used MTFE and MTFO to devise algorithms with better competitive ratios. Recall that these algorithms are instances of MOVE-TO-FRONT-EVERY-OTHER-ACCESS (MTF2) algorithms. In this section, we study the competitive ratio of these algorithms. In [39, Exercise 1.5], it is stated that any MTF2 algorithm is 2-competitive. The same statement is repeated in [21, 97]. It was first observed in [82] that MTF2 algorithms are in fact *not* 2-competitive. There, the author proves a lower bound of 7/3 for the competitive ratio of any MTF2 algorithm, and claims that an upper bound of 2.5 can be achieved. Here, we show that the competitive ratio of MTF2 is 2.5, and it is tight.

Lemma 34. *The competitive ratio of any* MOVE-TO-FRONT-EVERY-OTHER-ACCESS *algorithm is at least* 2.5.

Proof. We prove the lemma for MTFO and later extend it to other Move-To-Front-Every-Other-Access algorithms.

Consider a list of l items, initially ordered as $[a_1, a_2, \ldots, a_l]$. Consider the following sequence of requests:

$$\sigma_\delta = \left\langle (a_1, a_2, ..., a_l, a_1^3, a_2^3, ..., a_l^3, a_l, a_{l-1}, ..., a_1, a_l^3, a_{l-1}^3, ..., a_1^3)^m \right\rangle .$$

We show that asymptotically, the cost of MTFO is 2.5 times the cost of OPT. Similar to our other lower bound proofs, we define a phase as a subsequence of requests which forms one of the m repetitions in σ_δ. Note that each phase contains an even number of requests to each item. Also, if $i < j$, meaning that item a_i precedes item a_j in the initial ordering of the list, then, in each phase, a_i is requested three times after the last request to a_j. MTFO moves a_i in front of a_j due to these requests. Thus, the state of the list maintained by the algorithm is the same as the initial ordering after serving a phase.

Both MTFO and OPT incur a cost of $l(l+1)/2$ for serving $a_1, a_2, \ldots, a_l$ at the beginning of a phase. MTFO moves items to the front and reverses the list, but OPT does not move the items. For serving the subsequent requests to $a_1^2, a_2^2, ..., a_l^2$, MTFO incurs a cost of $2l^2 + l$ since it moves items to the front on the second of the three consecutive requests to an item. OPT moves to the front at the first of the consecutive requests and incurs a cost of $l(l+1)/2 + 2l$ (the second and third requests are to the front of the list). At this point, for both algorithms, the list is in the reverse of the initial ordering, while the bits maintained by MTFO are the same as in the beginning of the phase, since there have been four requests to each item, i.e., they are all 1.

For the next requests to $a_l, a_{l-1}, \ldots, a_1$, MTFO reverses the list and incurs a cost of $l(l+1)/2$. OPT has the same cost and does not move the items. Consequently, for the remaining requests to $a_l^2, a_{l-1}^3, \ldots, a_1^2$, MTFO incurs a cost of $2l^2 + l$, while OPT incurs a cost of $l(l+1)/2 + 2l$.

To summarize, in each phase, the cost of MTFO is $5l^2 + o(l^2)$, while the cost of OPT is $2l^2 + o(l^2)$. The actions and costs of the algorithms can be summarized as follows. The two lines correspond to MTFO, and OPT, respectively.

$$[\overset{1}{a_1} \ldots \overset{1}{a_l}] \xrightarrow[l^2/2+o(l^2)]{a_1 \ldots a_l} [\overset{0}{a_l} \ldots \overset{0}{a_1}] \xrightarrow[2l^2+o(l^2)]{a_1^3 \ldots a_l^3} [\overset{1}{a_l} \ldots \overset{1}{a_1}] \xrightarrow[l^2/2+o(l^2)]{a_l \ldots a_1} [\overset{0}{a_1} \ldots \overset{0}{a_l}] \xrightarrow[2l^2+o(l^2)]{a_l^3 \ldots a_1^3} [\overset{1}{a_1} \ldots \overset{1}{a_l}]$$

$$[a_1 \ldots a_l] \xrightarrow[l^2/2+o(l^2)]{a_1 \ldots a_l} [a_1 \ldots a_l] \xrightarrow[l^2/2+o(l^2)]{a_1^3 \ldots a_l^3} [a_l \ldots a_1] \xrightarrow[l^2/2+o(l^2)]{a_l \ldots a_1} [a_l \ldots a_1] \xrightarrow[l^2/2+o(l^2)]{a_l^3 \ldots a_1^3} [a_1 \ldots a_l]$$

We can extend the above lower bound to show that MTFE is at least 2.5-competitive. In doing so, consider the sequence $\langle (a_1, a_2, \ldots, a_l)\sigma_\delta \rangle$. Note that after serving the subsequence in parentheses, all bits maintained by MTFE become 1, and the same analysis as above holds for serving σ_δ. More generally, for any initial setting of the bits maintained by a MOVE-TO-FRONT-EVERY-OTHER-ACCESS algorithm, we can start a sequence with a single request to each item having bit 0. After this subsequence, all bits are 1 and we can continue the sequence with σ_δ to prove a lower bound of 2.5 for the competitive ratio of these algorithms. □

As mentioned earlier, an upper bound of 2.5 for the competitive ratio of MTF2 was claimed earlier [82]. Here we include the proof for completeness since it does not appear to have ever been published.

Lemma 35. *The competitive ratio of any online list update algorithm $\mathbb{A}$ that belongs to the family of* MOVE-TO-FRONT-EVERY-OTHER-ACCESS *algorithms is at most 2.5.*

Proof. We prove the statement for the partial cost model. Since $\mathbb{A}$ has the projective property and is cost independent, the upper bound argument extends to the full cost model. Consider a sequence σ_{xy} of two items x and y. As before, we use the phase partitioning technique and partition σ_{xy} into phases as in the proof of Lemma 27. Recall that a phase ends with two consecutive requests to the same item in σ_{xy}. A phase has type 1 (respectively 2) if the relative order of x and y is $[xy]$ (respectively $[yx]$) at the beginning of the phase. Recall that a phase of type 1 has one of the following three forms ($j \geq 0$ and $k \geq 1$):

$$(a)\ x^j yy \quad (b)\ x^j (yx)^k yy \quad (c)\ x^j (yx)^k x$$

A phase of type 2 has exactly the same form as above with x and y interchanged. To prove the lemma, we show that its statement holds for every *two consecutive* phases. First, we consider each phase separately and show that the cost of $\mathbb{A}$ is at most 2 times that of Opt for all phases except a specific phase type that we call a *critical phase*. Table 6.4 shows the costs incurred by $\mathbb{A}$ and Opt for each phase. Note that phases of the form (b) and (c) are divided into two and three cases, respectively. The last row in the table corresponds to a critical phase. We discuss the different phases of type 1 separately. Similar analyses, with x and y interchanged, apply to the phases of type 2.

Note that before serving a phase of type 1, the list is ordered as $[xy]$ and the first j requests to x have no cost. Consider phases of the form (a), $x^j yy$. $\mathbb{A}$ incur a total cost of at most 2 for serving yy and Opt incurs a cost of 1. So, the ratio between the costs of $\mathbb{A}$ and Opt is at most 2.

Next, consider phases of the form (b) with $k = 2i$ (i is a positive integer). By Lemma 28, the cost incurred by $\mathbb{A}$ is at most $3i$ for serving $(yx)^{2i}$. For the remaining two requests to y, $\mathbb{A}$ incurs a cost of at most 2. In total, the cost of $\mathbb{A}$ is at most $3i + 2$ compared to $2i + 1$ for Opt, and the ratio between them is less than 2.

Next, assume k is odd and $k = 2i - 1$, i.e., the phase has the form $x^j (yx)^{2i-2} yxyy$. The total cost of $\mathbb{A}$ for $(yx)^{2i-2}$ is at most $3(i - 1)$ (Lemma 28), and its cost for the next requests to $yxyy$ is at most 4. In total, it incurs a cost of at most $3i + 1$ for the phase, which is no more than twice the cost $2i$ of Opt.

Next, consider phases of the form (c). Assume k is even, i.e., the phase has the form $x^j (yx)^{2i} x$. By Lemma 28, $\mathbb{A}$ incurs a cost of at most $3i$ for $(yx)^{2i}$ and a cost of at most 1 for the single request to x. The cost of the algorithm will be $3i + 1$ compared to $2i$ of Opt, and the ratio between them is no more than 2. Next, assume k is odd, i.e., the phase has the form $x^j (yx)^{2i} yxx$ or $x^j yxx$ (as before, i is a positive integer). In the first case, by Lemma 28, $\mathbb{A}$ incurs a cost of $3i$ for $x^j (yx)^{2i}$ and a cost of at most 3 for yxx. This sums to $3i + 3$ while Opt incurs a cost of $2i + 1$; the ratio between these two is no more than 2. If the phase has the form $x^j yxx$, we refer to it as a critical phase. The cost of $\mathbb{A}$ for such a phase can be as large as 3 while Opt incurs a cost of 1. However, we show that the cost of $\mathbb{A}$ in two consecutive phases is no more than twice the cost of Opt.

Consider two consecutive phases in σ_{xy}. If none of the phases are critical, the cost of $\mathbb{A}$ is at most twice that of Opt in both phases and we are done. Assume one of the phases is critical while the other phase is not. Let Opt_1 and Opt_2 denote the cost of Opt for the critical and non-critical phases, respectively. We have $\text{Opt}_1 \leq \text{Opt}_2$ because Opt incurs a cost of 1 for critical phases and a cost of at least 1 for other phases (see Table 6.4). The cost of $\mathbb{A}$ for serving the two phases is at most $3\,\text{Opt}_1 + 2\,\text{Opt}_2$ which is no more than $2.5(\text{Opt}_1 + \text{Opt}_2)$ (since $\text{Opt}_1 \leq \text{Opt}_2$). This implies that the cost of $\mathbb{A}$ is no more than 2.5 more than that of Opt for the two phases. Finally, assume both phases are critical. Thus, they form a subsequence $(x^j yxx)(x^{j'} yxx)$ in σ_{xy}. $\mathbb{A}$ moves y to the front for exactly one of the two requests to y. Thus, it incurs a cost of 1 for one of the phases and a cost of at most 3 for the other phase. In total, its cost is no more than 4, while Opt incurs a cost of 2 for the two phases. □

Phase	MTF2 ($\mathbb{A}$)	OPT'	ratio
$x^j yy$	≤ 2	1	≤ 2
$x^j (yx)^{2i} yy$	$\leq 3i + 2$	$2i + 1$	< 2
$x^j (yx)^{2i-2} yxyy$	$\leq 3(i-1) + 4$	$2i$	≤ 2
$x^j (yx)^{2i} x$	$\leq 3i + 1$	$2i$	≤ 2
$x^j (yx)^{2i} yxx$	$\leq 3i + 3$	$2i + 1$	≤ 2
$x^j yxx$	3	1	3

Table 6.4: The costs of a MTF2 algorithm $\mathbb{A}$ and OPT for a phase of type 1 (i.e., the initial ordering of items is xy). The ratio between the cost of MTF2 and OPT for each phase, except the critical phase (the last row), is at most 2.

From Lemmas 34 and 35, we conclude the following theorem:

Theorem 18. *The competitive ratio of* MOVE-TO-FRONT-EVERY-OTHER-ACCESS *algorithms is 2.5.*

6.5 Remarks

Recall that the offline version of the list update problem is known to be NP-hard [12]. In this sense, our algorithm can be seen as a linear-time approximation algorithm with an approximation ratio of at most $1.\bar{6}$; this is, to the best of our knowledge, the best deterministic offline algorithm for the problem. It should be mentioned that the randomized online algorithm BIT also has a competitive ratio of $1.\bar{6}$ against an oblivious adversary [121]. Recall that BIT maintains a bit for each item and flips the bit on each access; whenever the bit becomes '0' it moves the item to the front. The bits are initially set uniformly at random; hence, BIT uses l bits of advice for lists of length l. COMB is another randomized algorithm which makes use of a linear number of random bits and improves the competitive ratio to 1.6 [8]. We can conclude that there are online algorithms which achieve a competitive ratio of at most 1.6 when provided a linear (in the length of the list) number of advice bits. However, from a practical point of view, it is not clear how an offline oracle can smartly generate such bits of advice. Moreover, our results (Theorem 18) indicate that, regardless of how the initial bits are generated, algorithm BIT has a competitive ratio of 2.5 against adaptive adversaries. This follows since an adaptive adversary can learn the original random bits from the behavior of BIT by requesting all items once, and then give requests to change 0-bits to 1-bits. This initial subsequence has constant length (proportional to the length of the list). After this, with a renaming of items based on their current order in the list, the adversary can treat BIT as MTFO and give σ_δ from the proof of Lemma 34.

We proved that the competitive ratio of the better algorithm between MTFE, MTFO, and TIMESTAMP is at least 1.6 and at most $1.6\bar{6}$. Similarly, for the better algorithm between MTFE and MTFO the competitive ratio is between 1.75 and 2. It would be interesting to close these gaps.

Part IV

Applications

Chapter 7

Fault-Tolerant Bin Packing (Server Consolidation)

Server consolidation is an important application of the bin packing problem in which the goal is to minimize the number of servers needed to host a set of clients. The clients appear in an online manner and each of them has a certain load. The servers have uniform capacity and the total load of clients assigned to a server must not exceed this capacity. Additionally, to have a fault-tolerant solution, the load of each client should be distributed between at least two different servers so that failure of one server avoids service interruption by migrating the load to the other servers hosting the respective second loads. In a simple setting, upon receiving a client, an online algorithm needs to select two servers and assign half of the load of the client to each server. In this chapter, we analyze this problem under the framework of competitive analysis. First, we provide upper and lower bounds for the competitive ratio of two well known heuristics which are introduced in the context of tenant placement in the cloud. In particular, we show their competitive ratios are no better than 2. We then present a new algorithm called Horizontal Harmonic and show that it has an improved competitive ratio which converges to 1.59. The simplicity of this algorithm makes it a good choice for use by cloud service providers. Finally, we prove a general lower bound that shows any online algorithm for the online fault-tolerant server consolidation problem has a competitive ratio of at least 1.42.

7.1 Introduction

Server consolidation is an essential concept for efficient use of computer resources by means of reducing the number of required active servers. Certain applications involve a large number of *clients* which appear in a sequential, online manner. Each client requires a certain amount of resources which is referred to as the *load* of the client. Servers have uniform *capacity*, which is the maximum load that they can supply so that their performance is not compromised. It is naturally assumed that the load of any client is no larger than the server capacity. At the time a client appears, one or more servers should be selected to host the client. If more than one server is selected, the load of the client is evenly split between them

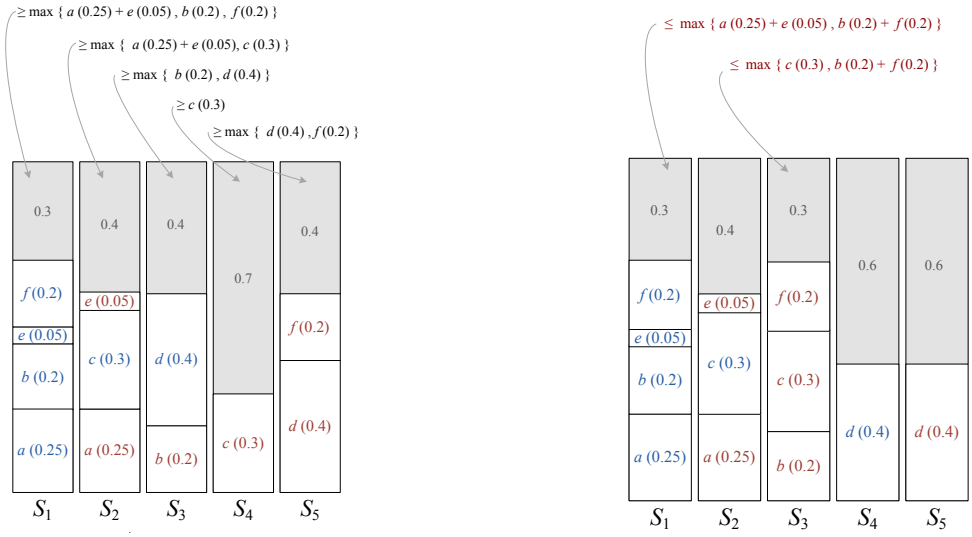
[127]. An efficient algorithm reduces the number of active servers by smart selection of the hosting servers for each client. Reducing the number of servers is essential to avoid *server sprawl*, namely the situation in which there are numerous under-utilized active servers which consume more resources than required by the workload. Preventing server sprawl is particularly important for saving on energy-related costs which account for 70 to 80 percent of a data centre's ongoing operational costs [59]. In a dynamic setting, the server consolidation problem is online in the sense that at the time of selecting servers for a client, an algorithm does not know the load of future clients.

The server consolidation problem is closely related to the bin packing problem. The only difference is that in the bin packing problem, in contrast to the server consolidation problem, items cannot be divided between two bins and each item is 'packed' on a single bin. In this chapter, we interchangeably use terms 'client' and 'item', as well as terms 'server' and 'bin'.

One application area of server consolidation is in multi-tenant systems [68, 127]. Here, each client represents a *tenant* which is an application or a process. Each tenant has a load and is hosted by one or multiple servers in a way that the total load of servers do not exceed their capacity. Typical examples of tenants are enterprise databases, websites, and on-demand media (e.g., Netflix movies) which are placed and run on powerful cloud servers like those of Amazon EC2 [11]. Depending on the application, the load of a tenant can be a function of its GPU consumption (e.g., online games), its bandwidth consumption (e.g., on-demand media), its size (e.g., enterprise databases) or a function of all. In cloud systems, the service provider (e.g, Amazon) has service contracts with customers that are owners of the tenants. These contracts define some *service-level-agreement constraints* (SLA constraints) which determine the cost of the service and the performance guarantees provided by the service provider. Based on the SLA constraints, the service provider assigns a load to each tenant so that it ensures that the SLA constraints are satisfied. In what follows, we assume the loads of tenants are computed upon their arrival and focus on how tenants should be placed on servers with regard to the loads. A related concept are cloud *jobs* which are similar to tenants except that they have lifetimes after which they leaves the system (while tenants stay permanently). When the goal is to assign jobs to servers, the problem becomes similar to the dynamic bin packing where the goal is to minimize the bin usage rather than the number of bins (see [109] for details).

In multi-tenant systems, similar to many other server consolidation applications, it is desirable to have a *fault-tolerant* solution so that failing or removing a single server does not interrupt the service. To achieve such a guarantee, there should be more than one copy or *replica* of each tenant in the system. When a tenant is replicated in k servers, its load is uniformly distributed among the k replicas, assuming the workload has read-mostly characteristics [127]. It is preferable to have a small number of replicas for each tenant since more replicas require complex management and the problem becomes more constrained [127]. Assuming a single server can handle the entire load of a tenant, having two replicas for a tenant is sufficient to protect the system against failure of a single server. In this case, the 1/2 load residing in the failed server migrates to the other server which must have space capacity for this transfer. In what follows, taking the same approach of [127], we assume each tenant has two replicas, which we refer to as *blue* and *red* replicas. The blue and red replicas of an item x both have size $s(x)/2$.

The two replicas associated with an item should be hosted on different servers so that in case of one server's failure, the tenant's load can be directed to the tenant replica on the other hosting server. This requires a reserved capacity in each server S_x so that in case of a failure of any other server $S_{x'}$ ($x' \neq x$), the additional load imposed on S_x (due to the items which are hosted by both servers) does not cause an overflow in S_x. To be more precise, let T denote the total load of items which have one replica in S_x and one in $S_{x'}$. There is a load of $T/2$ on each server associated with these items which we denote by $L_{(x,x')}$.



(a) A valid packing

(b) An invalid packing

Figure 7.1: Two packings of the sequence $\sigma = \langle a = 0.5, b = 0.4, c = 0.6, d = 0.8, e = 0.1, f = 0.4 \rangle$. Each item has a blue and a red replica. The packing on the left is a valid packing; if any server fails, the load redirected to other servers does not cause an overflow. The packing on the right is not valid since if server S_3 fails, the shared items between S_1 and S_3 (i.e., b and f) will add an extra load of size $0.2 + 0.2 = 0.4$ to S_1. The total size of replicas in S_1 will be $0.7 + 0.4 = 1.1$ which is more that the unit capacity of servers. Similarly, if server S_1 fails, the redirected load causes an overflow in S_3.

In case $S_{x'}$ fails, this load is redirected to S_x, i.e., the load of S_x is increased by $L_{(x,x')}$. Consequently, to maintain all items (tenants) after a failure, it is required to have a reserved capacity of $L_{(x,x')}$ in S_x. This reserved capacity can be shared by more than two clients if they reside in different servers since at most one server fails. This can be translated into the bin packing language as follows:

Definition 9. *An instance of the online fault-tolerant server consolidation problem (alternatively called online fault-tolerant bin packing problem) is defined by a sequence $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ of items (clients) which should be placed in a minimum number of bins (servers). Bins have uniform capacity of 1 and each item has a size $s(a_i)$ in the range $(0, 1]$. Items are revealed in an online manner. Upon receiving an item $a_i (1 \leq i \leq n)$, an algorithm should place two replicas of a_i, each having a size of $s(a_i)/2$, into two different bins. These two replicas are 'partners' of each other and are referred to as blue and red replicas. An online algorithm needs to maintain a 'valid packing' after packing each item. In a valid packing, no two replicas of the same item are placed in the same bin. Moreover, if we let L_x be the total size of replicas placed in a bin S_x, i.e., $L_x = \sum_{a_j \in S_x} s(a_j)/2$, and $L_{x,x'}$ be the total size of replicas in S_x which have their partner in $S_{x'}$, i.e., $L_{x,x'} = \sum_{a_j \in S_x \cap S_{x'}} s(a_j)/2$, we should have $L_x + L_{x,x'} \leq 1$ for all $x' \neq x$ (see Figure 7.1).*

The bin packing problem has been previously used as a model for server consolidation [85, 135, 137]. However, these approaches do not provide fault-tolerant solutions; moreover, the proposed strategies are

offline and not suitable for dynamic environments where clients are coming in an online manner. The fault-tolerant server consolidation problem as defined above was recently introduced by Schaffner et al. [127]. In the same paper, two strategies were introduced for the problem and their average performance was evaluated in a real-world system. The first strategy is referred to as the MIRRORING algorithm. This algorithm treats blue and red replicas separately using the BEST FIT strategy. Since two replicas of the same item have equal sizes, the packings associated with blue and red replicas are the same, i.e., they mirror each other. To achieve a valid packing, the capacity of each bin is assumed to be 0.5. This is because when a server (bin) fails, its entire load is redirected to the mirrored bin and the total load is doubled. We theoretically analyze the MIRRORING algorithm and show an upper bound of 3 and a lower bound of $2.\bar{6}$ for the competitive ratio of this algorithm.

The second algorithm introduced in [127] is what we shall call the INTERLEAVING algorithm. This algorithm also applies the BF strategy to place replicas. In doing so, it considers a *legal capacity* for each bin as the total capacity of the bin (i.e., 1) minus the level of the bin and the maximum load redirected to the bin in case of another bin's failure. More precisely, the legal capacity of a bin S_x is $1-L_x-\max_{x'} L_{(x,x')}$, where L_x is the total size of replicas in S_x, and $L_{(x,x')}$ is the total size of replicas in S_x which have their partner in bin $S_{x'}$ (see Definition 9). For placing the first replica of an item (the blue replica), the algorithm considers a fraction μ of the legal capacity of each bin to be 'available' and applies the BF strategy to place the replica in a bin with the smallest available capacity large enough to contain it (if any such bin exists). Here, μ is a parameter of the algorithm which is a positive value no more than 1. For placing the red replica, the actual legal capacity is considered and again the BF strategy is applied to place the replica in a bin other than that of its partner. We provide upper and lower bounds for the competitive ratio of the INTERLEAVING algorithm. Our results indicate that, in terms of competitive ratio, INTERLEAVING algorithm does not provide a big improvement over the MIRRORING algorithm. In particular, for the suggested value of $\mu = 0.85$ in [127], we show that the competitive ratio of the algorithm is in the range $(2, 3.71)$.

We introduce a new algorithm called HORIZONTAL HARMONIC *(*HH*)*. The algorithm is inspired by the HARMONIC algorithm for the bin packing problem [106] and defines K classes for replicas based on their sizes; here, K is a parameter of the algorithm. We show that the competitive ratio of this algorithm converges to 1.59 for large values of K. For small values of K, the competitive ratio of HH is still better than the existing algorithms, e.g., when $K = 30$, the competitive ratio of HH is 1.625. Hence, in the worst case, HORIZONTAL HARMONIC outperforms its counterparts. The algorithm is simple and runs in linear time.

We also prove a general lower bound on the competitive ratio of any online algorithm for the fault-tolerant server consolidation problem. We show the competitive ratio of any online algorithm is at least $10/7 > 1.428$.

7.1.1 A Shifting Technique

We introduce a *shifting lemma* which will be used in a few occasions in our lower bound arguments. An offline algorithm (particularly OPT) can use this technique to achieve a packing in which each pair of bins host replicas of at most one shared item. Consider a long sequence of replicas with a bounded number of different sizes, and assume item sizes are larger than a constant value. Consequently, each bin contains a bounded number of replicas and there are a bounded number of *bin types*. We say that two bins have the same type if the multi-sets formed by the sizes of the hosted replicas are the same for both bins.

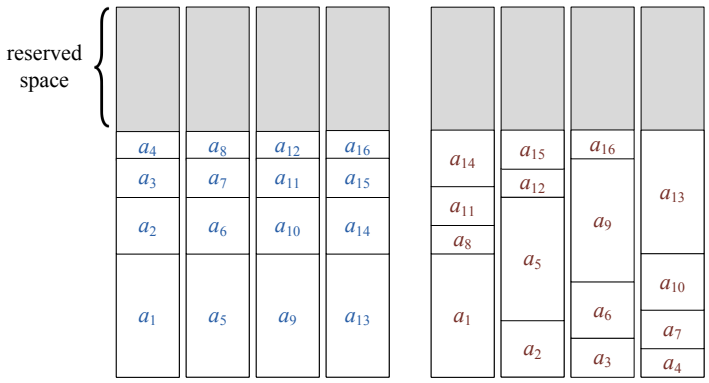


Figure 7.2: The shifting technique results in the same packings for the blue and the red replicas in a way that any two bins share replicas of at most one item. In this example, a bin with four different item sizes is considered. For each set of four bins hosting the blue replicas, four bins are opened for placing the red replicas. The red partners of the replicas in the first blue bin are distributed among these four bins; the same holds for the red replicas of other bins. In the resulting packing, the reserved space in each bin is equal to the size of the largest replica hosted on the bin.

Lemma 36. *Consider a packing of the blue replicas of a long sequence into X bins so that each bin contains a bounded number of replicas, and there are a bounded number of bin types. Assume there is an empty space of size greater than or equal to the size of the largest replica in each bin. To achieve a valid packing, an offline algorithm* Off *can place the red replicas into $X + c$ bins, where c is a constant integer.*

Proof. Note that if the empty space of a bin is less than the size of the largest replica in the bin, the packing is not valid (in case of the failure of the bin hosting the partner of the largest replica, the bin will be overloaded). To achieve a valid packing, when placing the red replicas, Off ensures that any two bins share replicas of at most one item. As a result, if any bin fails, the redirected load to any other bin B is smaller than the size of the largest replica hosted on B, which is indeed smaller than the reserved space in B.

To place the red replicas, Off considers a fixed ordering of the blue replicas inside all bins, e.g., assume the blue replicas are placed in decreasing order of their sizes. Consider a bin type u and let c_u denote the number of replicas in such a bin type. Off partitions the bins of type u into groups of size c_u; the last group might include less bins. The bins in each group include ${c_u}^2$ blue replicas (except potentially the last group). Off opens c_u new bins to place the red partners of these replicas in the following manner: if a blue replica x is placed as the jth replica in the ith bin in a group ($0 \leq i, j \leq c-1$), then Off places the red partner of x as the ith replica in the $(j+i)$ mod c_u-th bin among the bins opened for the red partners (see Figure 7.2.)

We show that in the resulting packing, no two bins share replicas of more than one item. Assume otherwise, i.e., assume two blue replicas hosted by a blue bin have their red replicas placed in the same red bin. This implies $(i+j)$ mod $c_u = (i+j')$ mod c_u, where i is the index of the blue bin in its group and j and j' are the indices of the two replicas inside the bin (w.l.o.g assume $j' > j$). So we will have

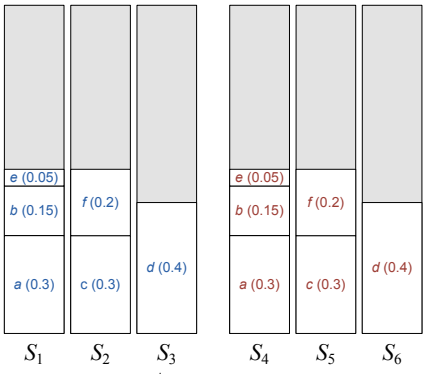


Figure 7.3: The packing of the Best-Fit Mirroring algorithm when applied on sequence $\langle\, a = 0.6, b = 0.3, c = 0.6\ , d = 0.8, e = 0.1, f = 0.2\,\rangle$.

$i + j = i + j' - k\ c_u$ for some positive integer k. This implies $j' \geq k\ c_u$ which contradicts the fact that there are c_u replicas in each bins.

For each group of c_u bins of type u in the packing of blue replicas, c_u bins are opened for the red replicas. The only exception is the last group of blue bins which might include as few as one bin while Off opens c_u red bins for this group. Since c_u is a constant, for each bin type, a bounded number of extra bins are opened. There are a bounded number of bin types; hence, the total number of opened bins for the red replicas is at most a constant value more than the number of bins in the packing of blue replica.

□

7.2 Analysis of the Existing Algorithms

In this section, we provide upper and lower bounds for the competitive ratio of existing algorithms for the fault-tolerant server consolidation problem.

7.2.1 Mirroring Algorithm

The Mirroring algorithm places blue and red replicas of all items separately using the Best Fit strategy. In case of a bin's failure, its entire load goes to the mirrored server. To achieve a valid packing, the capacity of each bin is assumed to be 0.5. Figure 7.3 shows a packing associated with the Mirroring algorithm. We first provide an upper bound for the competitive ratio of the Mirroring algorithm. In the following lemma, we consider a sequence of *replicas* in which each item is presented as two replicas of the same size.

Lemma 37. *For any sufficiently long sequence σ of replicas, the number of bins used by the* Mirroring *algorithm is at most* 3 *times more than that of* Opt *(within an additive constant).*

Proof. Let σ_1 denote a subsequence of σ formed by replicas smaller than or equal to $1/4$ and σ_2 denote the set of replicas larger than $1/4$. Also, let W_1 denote the total size of replicas in σ_1 and L_2 denote the number of replicas in σ_2, i.e., the length of σ_2. To prove the lemma we show that $\text{OPT}(\sigma) > W_1 + 3L_2/8$ and $\text{MIR}(\sigma) \leq 3W_1 + L_2 + c$, where $\text{MIR}(\sigma)$ is the number of bins used by the MIRRORING algorithm to pack σ and c is a constant value. Note that these two inequalities guarantee that $\text{MIR}(\sigma) < 3 \times \text{OPT}(\sigma) + c$.

The number of bins in an optimal packing is no less than the total size of all replicas in the sequence plus the required reserved space in the bins of an optimal packing. Let X denote the number of bins in an optimal packing which include a replica in σ_2 (i.e., a replica larger than $1/4$). Since at most two replicas of σ_2 can be hosted on the same bin (otherwise the reserved space will be less than the size of any of these replicas), we have $X \geq L_2/2$. For each bin which includes a replica of size larger than $1/4$, a reserved space of size larger than $1/4$ is required. Hence, the total reserved space in an optimal packing is more than $X/4 \geq L_2/8$. The total size of replicas in σ_1 is W_1 and the total size of replicas in σ_2 is at least $L_2/4$. Hence, the total size of replicas in σ is at least $W_1 + L_2/4$. Consequently, we have $\text{OPT}(\sigma) > W_1 + L_2/4 + L_2/8 = W_1 + 3L_2/8$.

Let m_1 (respectively m_2) denote the number of bins in the packing of the MIRRORING algorithm which do not include (respectively include) a replica of size larger than $1/4$. So, we have $\text{MIR}(\sigma) = m_1 + m_2$. Clearly $m_2 \leq L_2$ since the number of bins opened for replicas of size larger than $1/4$ cannot be more than the number of these replicas. We show $m_1 \leq 3W_1$. Let σ_s denote the set of blue replicas in σ which are placed in bins without a replica larger than $1/4$, and let Y denote the number of such bins. Since the red replicas are excluded, we have $Y = m_1/2$. Let W_s denote the total size of replicas in σ_s; note that $W_s \leq W_1/2$. The algorithm applies the BEST FIT strategy to place the replicas in σ_s into bins of size $1/2$. The number of opened bins does not change if we double the size of replicas and capacity of bins at the same time; hence, the number of bins opened for replicas in σ_s (i.e., Y) is equal to the number of bins that the BEST FIT algorithm opens for the same sequence as σ_s in which replicas sizes are doubled. Doubling replicas' sizes in σ_s results in a sequence in which each replica has a size at most equal to $1/2$. The number of bins used by BEST FIT to pack such a sequence is at most 1.5 times the total size of the sequence (within an additive constant) [49]. Hence, $Y \leq 1.5 \times 2W_s + c' \leq 3 \times W_1/2 + c'$ for some constant c'. So, we have $m_1 = 2Y \leq 3W_1 + c$ and $\text{MIR}(\sigma) = m_1 + m_2 \leq 3W_1 + L_2 + c$, where $c = 2c'$. □

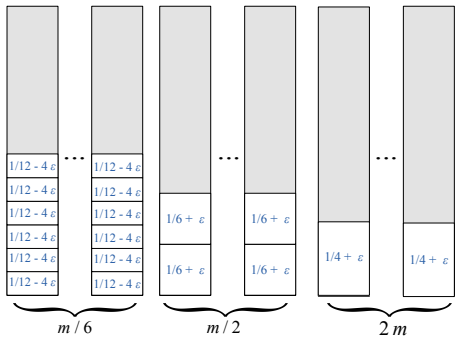
Next, we provide a lower bound for the competitive ratio of the MIRRORING algorithm algorithm:

Lemma 38. *There are arbitrary long sequences for which the number of bins used by the* MIRRORING *algorithm is at least* $8/3$ *times more than that of* OPT *(within an additive constant).*

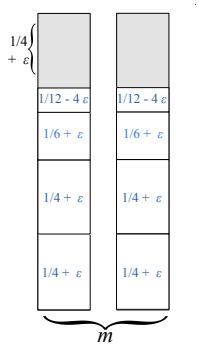
Proof. Consider the following sequence σ of $n = 4m$ items where m is a large integer. The sequence starts with m items of size $\frac{1}{6} - 8\epsilon$, followed by m items of size $\frac{1}{3} + 2\epsilon$, and ends with $2m$ items have size $\frac{1}{2} + 2\epsilon$. So, the sequence of blue (and red) *replicas* has the following sizes:

$$\sigma_{blue} = \langle \underbrace{\frac{1}{12} - 4\epsilon, \ldots, \frac{1}{12} - 4\epsilon}_{\text{m}}, \underbrace{\frac{1}{6} + \epsilon, \ldots, \frac{1}{6} + \epsilon}_{\text{m}}, \underbrace{\frac{1}{4} + \epsilon, \ldots, \frac{1}{4} + \epsilon}_{\text{2m}} \rangle$$

We show that the number of bins used by the MIRRORING algorithm to pack σ is $16m/3$ while that of OPT is at most $2m + c$ where c is a non-negative constant independent of m. The ratio between these values approaches $8/3$ for large values of m.



(a) Packing of the MIRRORING algorithm



(b) Packing of OFF

Figure 7.4: Packings of the MIRRORING algorithm and OFF for placing the blue replicas of σ. The packing of the MIRRORING algorithm for the red replicas is a mirror of its packing for the blue replicas, while OFF applies the shifting technique for placing the red replicas.

For the MIRRORING algorithm, we only consider the number of bins in the packing associated with the blue replicas; the actual number of bins used by the algorithm is twice this value. To pack the first m blue replicas, the algorithm places 6 replicas of size $1/12 - 4\epsilon$ in the same bin. The level of each bin will be $1/2 - 24\epsilon$ and the remaining capacity would be 24ϵ (recall that the level of a bin cannot be more than $1/2$ in a valid packing for the MIRRORING algorithm). Hence, no other replica will be placed in these bins and they can be thought as being closed; the algorithm opens $m/6$ bins for placing the first m replicas. Similarly, the algorithm places two replicas of size $1/6 + \epsilon$ in the same bin and opens $m/2$ bins for these replicas. Finally, the algorithm opens a bin for each replica of size $1/4 + \epsilon$ (having two replicas of that size results in a level larger than $1/2$). In total, the number of opened bins for the blue replicas is $m/6 + m/2 + 2m = 8m/3$. Adding to this the number of mirrored bins for the red replicas, the total number of bins used by the MIRRORING algorithm will be $16m/3$ (see Figure 7.4a).

Next, we describe an offline algorithm OFF which places blue and red replicas separately. For placing the blue replicas, OFF places two replicas of size $1/4 + \epsilon$ together with one replica of size $1/6 + \epsilon$ and one replica of size $1/12 - 4\epsilon$ in the same bin. Note that the level of such a bin will be $3/4 - \epsilon$, and there is an empty space of size equal to the largest replica placed in the bin. OFF opens m bins to place the blue replicas. Since there is only one bin type with four replicas in each bin in the packing of blue replicas, OFF can apply the shifting technique (Lemma 36) to place the red replicas in $m + c$ bins for some constant c. Consequently, the number of bins used by OPT is no more than $2m + c$ (see Figure 7.4b).

□

From Lemmas 37 and 38 we get the following theorem:

Theorem 19. *The competitive ratio of the* MIRRORING *algorithm for the fault-tolerant server consolidation problem is at least* $8/3 = 2.\bar{6}$ *and at most 3.*

7.2.2 Interleaving Algorithm

In the INTERLEAVING algorithm, in contrast to the MIRRORING algorithm, the blue and red replicas of different items are 'mixed', i.e., a bin can include blue replicas of some items and red replicas of some other items. To place the blue replicas, the algorithms considers a fraction $\mu \leq 1$ of the legal capacity and applies the BEST FIT strategy (the suggested parameter in [127] is $\mu = 0.85$). Recall that the legal capacity of a bin is its actual capacity (its left space) minus the maximum load that might be redirected to the bin in case of another bin's failure. For placing the red replicas, the entire legal capacity of the bins is considered and the BEST FIT strategy is applied to place the replica in an bin which does not include its partner. This way, more replicas are assigned to a single bin and the average number of opened bins decreases when compared to the MIRRORING algorithm. However, for many sequences, the MIRRORING algorithm and the INTERLEAVING algorithm result in almost similar packings. This is particularly the case when replicas' sizes are relatively small. In what follows, we provide upper and lower bound for the competitive ratio of the INTERLEAVING algorithm.

Lemma 39. *For any sufficiently long sequence σ, the number of bins used by the* INTERLEAVING *algorithm is at most $4/\mu - 1$ times that of* OPT *(within an additive constant).*

Proof. The structure of the proof is similar to that of Lemma 37. Let σ_1 denote a subsequence of σ formed by replicas smaller than or equal to $\mu/4$ and σ_2 denote the set of replicas larger than $\mu/4$. Also, let W_1 denote the total size of σ_1 and L_2 denote the number of replicas in σ_2 (i.e., the length of σ_2). We prove that $\text{OPT}(\sigma) > W_1 + L_2\mu/(4-\mu)$ while $\text{IA}(\sigma) \leq 8W_1/3 + L_2 + c$, where $\text{IA}(\sigma)$ is the number of bins used by the INTERLEAVING algorithm and c is a constant value. These two prove a competitive ratio of at most $\max(4/\mu - 1, 8/3)$ for the INTERLEAVING algorithm. Note that for $\mu \leq 1$, we have $4/\mu - 1 \geq 8/3$.

The number of bins used by OPT is no less than the total size of all replicas in the sequence plus the required reserved space in the bins of an optimal packing. Let X denote the number of bins in an optimal packing which include a replica of σ_2. Also, let $i \geq 5$ denote an integer so that $1/i < \mu/4 \leq 1/(i-1)$. Hence, at most $(i-2)$ replicas of σ_2 can be hosted on the same bin; otherwise, the reserved space will be less than the size of any of these replicas. So we have $X \geq L_2/(i-2)$. For each bin which includes a replica of size larger than $\mu/4$, a reserved space of size more than $\mu/4$ is required. Hence, the total reserved space in an optimal packing is more than $X \times (\mu/4) \geq L_2\mu/(4(i-2)) > L_2\mu^2/(16-4\mu)$. The last inequality holds because $i - 2 \leq (4-\mu)/\mu$. The total size of replicas in σ_1 is W_1 and the total size of replicas in σ_2 is more than $L_2 \times (\mu/4)$. Hence, the total size of replicas in σ is lower bounded by $W_1 + L_2 \times \mu/4$. Consequently, we have $\text{OPT}(\sigma) > W_1 + L_2\mu/4 + L_2\mu^2/(16-4\mu)) = W_1 + L_2\mu/(4-\mu)$.

Let m_1 (respectively m_2) denote the number of bins in the packing of the INTERLEAVING algorithm which do not include (respectively include) a replica of size larger than $\mu/4$. So, we have $IA(\sigma) = m_1 + m_2$. Clearly $m_2 \leq L_2$. We show $m_1 \leq 8W_1/3 + c$. Among the bins which include only replicas of size smaller than or equal to $\mu/4$, consider the last bin opened by the INTERLEAVING algorithm. If the first replica in such a bin is a blue replica, by definition of the INTERLEAVING algorithm, for any previously opened bin B we have $\mu \times cap(B) < \mu/4$ where $cap(B)$ is the legal capacity of B. If the first replica in the last bin is a red replica, we will have $cap(B) < \mu/4$ which also gives $\mu \times cap(B) < \mu/4$ (the only exception might be the bin which includes the partner of the red replica). Recall that the legal capacity of B is the remaining space of B minus the maximum total size of replicas which are shared between B and any other bin B'. From this definition we get $cap(B) \geq 1 - 2\ level(B)$; consequently, we have $\mu \times (1 - 2\ level(B)) < \mu/4$ and $level(B) \geq 3/8$. So, the level of all bins which include only replicas of size at most $\mu/4$, except potentially

a constant number of them, is at least 3/8. The total size of all replicas in these bins is at most W_1; hence, the number of these bins (m_1) is at most $8W_1/3+c$. Consequently, the number of bins used by the INTERLEAVING algorithm is at most $8W_1/3 + L_2 + c$.

□

Note that when μ converges to zero, the above upper bound does not provide a non-trivial worst-case guarantee for the performance of the INTERLEAVING algorithm. In fact, for small values of μ, the algorithm is not competitive at all (i.e., does not have a constant competitive ratio). This is because, when μ is sufficiently small, the algorithm opens a new bin for placing each blue replica while an optimal offline algorithm can efficiently place these replicas together in same bins. In what follows, we prove a general lower bound for the competitive ratio of the INTERLEAVING algorithm which holds for all values of μ.

Lemma 40. *There are arbitrary long sequences for which the number of bins used by the* INTERLEAVING *algorithm is at least $2-\epsilon^*$ times more than that of* OPT*, in which ϵ^* is a small constant positive value.*

Proof. Consider an input sequence σ with the following subsequence of *replica* sizes in which $\epsilon_m = \epsilon^*/8$ and $\epsilon_i = \epsilon_{i+1} \times \frac{\mu}{2\mu+1}$ $(1 \leq i \leq m-1)$. Here, m is an arbitrary integer which defines the length of the sequence.

$$\sigma = \langle \underbrace{\epsilon_1, \ldots, \epsilon_1}_{n_1}, \underbrace{\epsilon_2, \ldots, \epsilon_2}_{n_2}, \ldots, \underbrace{\epsilon_m, \ldots, \epsilon_m}_{n_m} \rangle$$

We define the values of n_i $(1 \leq i \leq m)$ to be $\lfloor \frac{\mu-\epsilon_i}{2\mu\epsilon_i} \rfloor$. Let W denote the total size of replicas in σ. To prove the lemma, we show that in the packing of σ by the INTERLEAVING algorithm, the level of all bins is smaller than or equal to 1/2, while there is an offline packing in which the level of all bins is at least $1-4\epsilon_m$. This implies that the number of bins used by the INTERLEAVING algorithm is at least $2W$ while that of OPT is at most $W/(1-4\epsilon_m)$. Consequently, the competitive ratio of the INTERLEAVING algorithm is at least $2(1-4\epsilon_m) = 2-\epsilon^*$.

To place the first two replicas of size ϵ_1, the INTERLEAVING algorithm opens 2 bins. We argue that it places the rest of replicas with size ϵ_1 in these two bins. After placing t items of size ϵ_1 in each of these bins $(t \leq n_1 - 1)$, the level of the bins will be $t \times \epsilon_1$ and their legal capacity will be $1-2t\epsilon_1$. For placing the next two replicas (the next item), the INTERLEAVING algorithm compares ϵ_1 with either a fraction μ of the legal capacity (for the blue replica) or the actual legal capacity (for the red replica). In both cases, ϵ_1 is smaller because we have $\mu \times (1-2t\epsilon_1) > \mu(1-2n_1\epsilon_1) \geq \epsilon_1$. Consequently, the first n_1 replicas will be placed into two bins. Next, we show that after placing n_1 replicas, there is no enough space for any other replica in these bins. For any consequent replica ϵ_j $(j > 1)$, we have $\epsilon_j \geq \epsilon_1(2 + 1/\mu)$. This gives the following:

$$\epsilon_1(2+\frac{1}{\mu}) \leq \epsilon_j \Rightarrow 1 - 2\epsilon_1(\frac{1}{2\epsilon_1} - \frac{1}{2\mu} - 1) \leq \epsilon_j \Rightarrow$$
$$1 - 2\epsilon_1 \lfloor \frac{1}{2\epsilon_1} - \frac{1}{2\mu} \rfloor < \epsilon_j \Rightarrow 1 - 2\epsilon_1 n_1 < \epsilon_j$$

Consequently, ϵ_j is larger than the legal capacity of the two bins and placing it in any of these two bins results in an invalid packing. Hence, the two bins opened for ϵ_1 can be assumed as being closed after

placing the n_1 replicas of this size. Consequently the INTERLEAVING algorithm opens a new pair of bins for the next replicas which have size ϵ_2.

Replacing ϵ_1 and n_1 with respectively ϵ_i and n_i in the above argument, one can show that the INTERLEAVING algorithm opens a pair of bins for each group of replicas of size ϵ_i $(1 \leq i \leq m)$ and closes the bins after placing these replicas. Consequently, in the packing of the INTERLEAVING algorithm, each bin has a mirrored bin. The level of these bins cannot be more than 1/2 in a valid packing. As a result, the number of bins used by the INTERLEAVING algorithm bins for placing σ cannot be less than $W/2$.

Consider an offline algorithm OFF that places blue and red replicas separately using the NF strategy. For blue replicas, the algorithm assumes a capacity of $1 - 2\epsilon_m$ for each bin. This way, the level of each bin (except possibly one) will be more than $1 - 3\epsilon_m$ and OFF opens M bins for the blue replicas where $M < W/(2 \times (1 - 3\epsilon_m)) + 1$. Note that M grows with m. To place the red replicas, OFF again assumes a capacity of $1 - 2\epsilon_m$ for each bin and applies NF on a different permutation of input. The red partners of blue replicas that are placed in the same bin are partitioned into a set of *multi-replicas*. Each multi-replica includes a multiset of red replicas whose total size is a constant value between ϵ_m and $2\epsilon_m$. The permutation is defined in rounds. Each round includes exactly one multi-replica from red replicas of each of the M blue bins. Since the size of multi-replicas is constant and the value of M grows with m, each round involves opening more than one bin (assuming m is large). Hence, no two multi-replicas of the same blue bin are placed in the same red bin. Moreover, the total size of shared replicas between two bins is no more than the size of multi-replicas, which is no more than $2\epsilon_m$. Hence, the total redirected load in case of a bin's failure is no more than the reserved space. Finally, the level of all red bins (except possibly one) is more than $1 - 4\epsilon_m$ as the size of multi-replicas is at most $2\epsilon_m$. Consequently, the number of bins used by OFF (and hence, OPT) is at most $W/(1 - 4\epsilon_m) + c$ where c is a constant. To conclude, the competitive ratio of the INTERLEAVING algorithm is at least $2(1 - 4\epsilon_m) = 2 - \epsilon^*$.

□

From Lemmas 39 and 40, we get the following result:

Theorem 20. *The competitive ratio of the* INTERLEAVING *algorithm with parameter μ for the fault-tolerant server consolidation problem is at least $2 - \epsilon$ and at most $4/\mu - 1$, where ϵ is a small constant positive value.*

Note that for the suggested value of $\mu = 0.85$, the competitive ratio of INTERLEAVING algorithm is in the range $(2 - \epsilon, 3.71)$.

7.3 Horizontal Harmonic Algorithm

In this section we introduce the HORIZONTAL HARMONIC (HH) algorithm for the fault-tolerant server consolidation problem which is inspired by the classic HARMONIC algorithm for the bin packing problem. Similar to the HARMONIC algorithm, HH is based on placing replicas of almost equal sizes in the same bins. It define *classes* for replicas based on their sizes and treats replicas of each class separately. The algorithm has a parameter K which defines the number of classes. We assume K is a constant around 30.

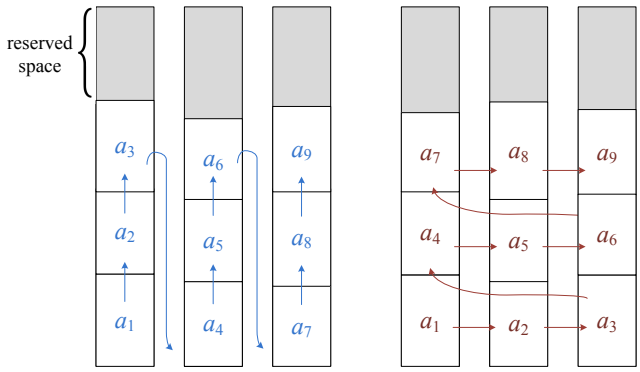


Figure 7.5: The main idea behind the HORIZONTAL HARMONIC algorithm is to apply HARMONIC algorithm on the blue replicas, while horizontally placing the red replicas of type i in i different bins. This ensures that no two bins share replicas of more than one item. In this example, it is assumed that items arrive as $\langle a_1, a_2, \ldots, a_9 \rangle$. The replicas have type 3, i.e., their size is in the range $(\frac{1}{5}, \frac{1}{4}]$. Three replicas are placed in each bin while an empty space of size $\frac{1}{4}$ is reserved in each bin. The size of the reserved space is an upper bound for the size of replicas in this class.

The replicas with sizes in the range $(\frac{1}{i+2}, \frac{1}{i+1}]$ belong to class i, where $1 \leq i < K$ (Note that the size of a replica is at most 1/2). The replicas which have size in the range $(0, \frac{1}{K+1}]$ belong to class K.

For placing the blue replicas from class $i < K$, HORIZONTAL HARMONIC places i replicas in the same bin; this way, an empty space of size $\frac{1}{i+1}$ is reserved for the load of one replica of the same class in case of another bin's failure. One can think of placing the blue replicas as vertically stacking them into bins, one bin after another. For placing the red replicas of class i, the algorithm opens i bins. If the blue replica of an items x is placed as the jth replica in its bin, the red replica is placed in the jth bin among the i open bins for the red replicas. This ensures that two bins share replicas of at most one item. Consequently, the reserved space for one bin is sufficient for having a valid packing (see Figure 7.5).

For placing the replicas in class K, the algorithm considers the largest integer α_K so that $\alpha_K^2 + \alpha_K \leq K$, i.e., $\alpha_k = \lfloor \frac{\sqrt{4K+1}-1}{2} \rfloor$. This ensures that $\frac{1}{\alpha_K} - \frac{1}{\alpha_K+1} \geq \frac{1}{K}$; consequently, the algorithm can group sets of replicas of class K into *multi-replicas* with total size in the range $(\frac{1}{\alpha_K+1}, \frac{1}{\alpha_K}]$. The algorithm treats these multi-replicas similar to the way that it treats replicas of class $\alpha_K - 1$, i.e., it places $\alpha_K - 1$ multi-replicas in the same bin. In what follows, when there is no risk of confusion, we replace α_K with α. Algorithm 4 illustrates the details of the algorithm.

HORIZONTAL HARMONIC Algorithm guarantees that two bins do not share replicas of more than one item. At the same time, it guarantees that each bin has a certain level (used space). These properties intuitively justify the advantage of the algorithm over the algorithms which are based on the BEST FIT strategy. HORIZONTAL HARMONIC is simple and runs in linear time. This gives another advantage to the algorithm compared to the existing algorithms which are based on the BF and run in time $\Theta(n \lg n)$.

Algorithm 4: HORIZONTAL HARMONIC with parameter K

input : A sequence $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ of items (clients)
output: A fault-tolerant packing of σ

$\alpha \leftarrow \lfloor(\sqrt{4K+1}-1)/2\rfloor$; `// used for the replicas of class` K
$mrSize \leftarrow 0$; `// multi-replica size`
for $j \leftarrow 1$ *to* K **do**
  $blueBins_j$, $redBins_j \leftarrow$ arrays of j empty bins
  / / $bIndex_j$ (resp. $rIndx_j$) is the index of the current blue (red) bin among the j open blue (red) bins of type j
  $bIndx_j$, $rIndx_j \leftarrow 1$
  / / set the number of replicas (multi-replicas) that fit in a bin of type j (capacity of the bin)
  if $j < K$ **then** $cap_j \leftarrow j$;
  else $cap_j \leftarrow \alpha - 1$;
end
for $i \leftarrow 1$ *to* n **do**
  $repSize \leftarrow s(a_i)/2$; `// size of the replicas (`$a_{\texttt{blue}}, a_{\texttt{red}}$`)`
  $j \leftarrow \lfloor 1/repSize \rfloor - 1$; `// the class of the current replica`
  if $rIndx_j > cap_j$ **then**
    $rIndx_j \leftarrow 1$; $bIndx_j \leftarrow bIndx_j + 1$
    if $bIndx_j > cap_j$ **then**
      $blueBins_j$, $redBins_j \leftarrow$ arrays of j empty bins
      $bIndx_j$, $rIndx_j \leftarrow 1$
  place a_{blue} (the blue replica of a_i) into bin $blueBins_j[\text{bIndx}_j]$
  place a_{red} (the red replica of a_i) into bin $redBins_j[\text{bIndx}_j]$
  if $j < K$ **then** $rIndx_j \leftarrow rIndx_j + 1$;
  else
    $mrSize \leftarrow mrSize + repSize$
    if $mrSize > 1/(\alpha+1)$ **then**
      $rIndx_j \leftarrow rIndx_j + 1$; $mrSize \leftarrow 0$
  end
end

Lemma 41. *The competitive ratio of* HH *with* $K \geq 30$ *classes is at most*

$$\chi = 1.5 + 1/12 \times \max\left(\frac{\alpha+1}{\alpha-1}, \frac{13}{11}\right)$$

Proof. Consider a sequence σ of sufficiently large length. We would like to show $\text{HH}_K(\sigma) \leq \chi\, \text{OPT}(\sigma)$. To do so, we assign weights to replicas in σ based on their class. The weight of a replica which belongs to class $i < K$ is equal to $1/i$; the weight of a replica x in class K is $\frac{s(x)(\alpha+1)}{\alpha-1}$. Recall that $\alpha = \lfloor \frac{\sqrt{4K+1}-1}{2} \rfloor$. Furthermore, we define *density* of a replica as the ratio between the weight and the size of the replica. See Table 7.1 for a summary of weight and density of replicas in different classes.

To prove the lemma we show the followings:

(I) The total weight of replicas in any bin of HH, except possibly a constant number of them, is at least 1.

(II) The total weight of replicas in a bin of OPT is at most χ.

The above statements respectively imply that $\text{HH}(\sigma) \leq W(\sigma) + c$ and $\text{OPT}(\sigma) \geq W(\sigma)/\chi$ in which $W(\sigma)$ is the total weight of replicas in σ. We will have $\text{HH}(\sigma) \leq \chi\,\text{OPT}(\sigma) + c$ which completes the proof.

Proving (I) is relatively easy. Consider a bin of HH associated with replicas of class j ($j < K$). HH places j replicas of this class in the same bin (except possibly the $2j$ most recently opened bins of the class). Consequently, the total weight of replicas is $j \times 1/j = 1$ for all bins of class j (except the mentioned ones). The replicas of class K are accumulated and the associated multi-replicas are treated like replicas of class $\alpha - 1$. The level of these bins is at least $\frac{\alpha-1}{\alpha+1}$ and consequently the total weight of replicas in such bins is 1 (again, with the exception of the last $2\alpha-2$ most recently opened bins of class K). To summarize, the weight of all bins of HH, except a constant number of them in each class, is at least 1. Since K is a constant, statement (I) follows.

To prove (II), we show that to achieve maximum weight, a bin B_{opt} should include a replica of class 1 with weight $1/3 + \epsilon$ and a replica of class 2 with weight $1/4 + \epsilon$, where ϵ is a sufficiently small positive value. Let B_1 denote a bin which includes three replicas of sizes $1/3 + \epsilon$, $1/4 + \epsilon$, and $1/13 + \epsilon$. Note that there is enough space for the largest replica (i.e., $1/3 + \epsilon$) in case of a bin's failure. The total weight of replicas in B_1 is $\omega(B_1) = 1 + 1/2 + 1/11 > 1.59$.

Assume B_{opt} does not include a replica of class 1. Consider the case that the largest replica in B_{opt} belongs to class $i \in \{2, 3, 4\}$. The level of the bin is less than $(i + 1)/(i + 2)$ (otherwise, there will not be enough space in case of failure of the bin hosting the partner of largest replica). Since $K \geq 30$, we have $\alpha \geq 5$ and the density of replicas in class K will be less than $3/2$. Since the items of class i (or higher) have density larger than $(i + 2)/i$, the density of all replicas in the bin is less than $(i + 2)/i$; this implies that the total weight of the bin cannot be more than $(i+1)/(i+2) \times (i+2)/i = (i+1)/i < \omega(B_1)$. If the largest replica belongs to class 5 or higher, the density of all replicas in the bin will be less than $3/2$ and consequently the weight of the bin is less than $3/2 < \omega(B_1)$.

Hence, to achieve the maximum weight, B_{opt} should include a replica of class 1. If such a replica has size more than $1/3+\epsilon$, one can replace it with $1/3+\epsilon$ and fill the resulting space with replicas of size ϵ to achieve a new bin with weight more than B_{opt}. Hence, to achieve the maximum weight, a bin b_{opt} should include a replica of size $1/3 + \epsilon$. This implies that there should be an empty space of size $1/3 + \epsilon$ in B_{opt}, i.e., the level of B_{opt} cannot be more than $2/3 - \epsilon$ and there will be an empty space of size $1/3 - 2\epsilon$ to be filled with other replicas. We claim that a replica of class two with size $1/4+\epsilon$ should be in B_{opt}. Consider otherwise; then the density of replicas (except the one with type 1) in B_{opt} is less than $5/3$ and the total weight of all replicas in B_{opt} will be less than $1 + (1/3 - 2\epsilon) \times 5/3 < \omega(B_1)$. So, there is a replica of class two in B_{opt}. As before, this replica cannot have size more than $1/4 + \epsilon$ (otherwise, it can be reduced to a replica of size $1/4 + \epsilon$).

So, to achieve maximum weight, B_{opt} should have replicas of sizes $1/3 + \epsilon$ and $1/4 + \epsilon$. Note that the weight of these two replicas is $1+1/2 = 1.5$. There will be an available space of size $1/12-3\epsilon$. Only replicas of class greater or equal to 11 can fill this empty space (i.e., replicas with size smaller than $1/12$). As Table 7.1 indicates, all these replicas have density smaller than $\max\{\frac{\alpha+1}{\alpha-1}, \frac{13}{11}\}$. Consequently, the weight of B_{opt} cannot be more than $\chi = 1.5 + 1/12 \times \max\left(\frac{\alpha+1}{\alpha-1}, \frac{13}{11}\right)$.

□

Class	Replica Size	No. Replicas in a Bin	Bin Level	Replica Weight	Replica Density
1	$s(x)/2 \in (\frac{1}{3}, \frac{1}{2}]$	1	$> \frac{1}{3}$	1	< 3
2	$s(x)/2 \in (\frac{1}{4}, \frac{1}{3}]$	2	$> \frac{1}{2}$	$\frac{1}{2}$	< 2
...	...	...	...	...	...
$i < K$	$s(x)/2 \in (\frac{1}{i+2}, \frac{1}{i+1}]$	i	$> \frac{i}{i+2}$	$\frac{1}{i}$	$< \frac{i+2}{i}$
...	...	...	...	...	...
$K-1$	$s(x)/2 \in (\frac{1}{K+1}, \frac{1}{K}]$	$K-1$	$> \frac{K-1}{K+1}$	$\frac{1}{K-1}$	$< \frac{K+1}{K-1}$
K	$s(x)/2 \in (0, \frac{1}{K+1}]$	N/A	$> \frac{\alpha-1}{\alpha+1}$	$\frac{x(\alpha+1)}{\alpha-1}$	$< \frac{\alpha+1}{\alpha-1}$

Table 7.1: Characteristics of replicas and bins for each class of Horizontal Harmonic.

The following two lemmas provide lower bounds for the competitive ratio of Hh. To analyze the algorithm, define $\beta_1 = 3$, $\beta_2 = 4$ and $\beta_{i+1} = \beta_i(\beta_i - 1) + 1$ $(i \geq 2)$.

Lemma 42. *The competitive ratio of* Hh *with* $K \geq 5$ *classes is at least* $\sum_{i=1}^{K-1} 1/(\beta_i - 2) > 1.597$.

Proof. Consider a sequence of replicas with the following sizes (blue and red replicas are included in the sequence and n is an even integer):

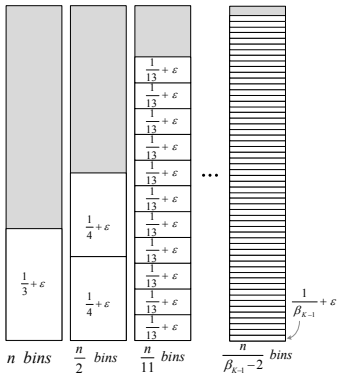
$$\sigma = \langle \underbrace{\tfrac{1}{3}+\epsilon, \ldots, \tfrac{1}{3}+\epsilon}_{n}, \underbrace{\tfrac{1}{4}+\epsilon, \ldots, \tfrac{1}{4}+\epsilon}_{n}, \ldots, \underbrace{\tfrac{1}{\beta_K}+\epsilon, \ldots, \tfrac{1}{\beta_K}+\epsilon}_{n} \rangle$$

Hh classifies replicas by their sizes and places $\beta_i - 2$ replicas of size $\frac{1}{\beta_i} + \epsilon$ in the same bin (see Figure 7.6a). Consequently, the number of opened bins will be $n \times \sum_{1}^{K-1} 1/(\beta_i - 2) + c$, where c is a constant.

To place the blue replicas, an offline algorithm Off includes all replicas of different sizes in the same bin. As illustrated in Figure 7.6b, there will be an empty space of size larger than $1/3 + \epsilon$ in such a bin. The total size of replicas plus the reserved space will be $\frac{2}{3} + \frac{1}{4} + \ldots + \frac{1}{\beta_K} < 1$. This way, Off opens $n/2$ bins for placing the blue replicas. Note that there is a constant number of replica sizes and bin types in the packing of the blue replicas. Hence, Off can apply the shifting lemma (Lemma 36) to place the red replicas in $n/2 + c'$ bins. In total, Off opens $n + c'$ bins for packing σ. Consequently, the ratio between the number of bins used by Hh and that of Opt for packing σ is at least $\sum_{i=1}^{K-1} 1/(\beta_i - 2)$ for large values of n.

□

Lemma 43. *The competitive ratio of* Hh *with* K *classes is at least* $1.5 + \frac{\alpha+1-3\epsilon'}{(\alpha-1)(12+\epsilon')}$ *in which* ϵ' *is an arbitrary small constant value.*



(a) The packing of the HH (b) The packing of OFF

Figure 7.6: Lower bound argument for the HORIZONTAL HARMONIC algorithm.

Proof. Consider a sequence of replicas with the following sizes (blue and red replicas are included in the sequence and n is an even integer):

$$\sigma = \langle \underbrace{\frac{1}{3}+\epsilon, \ldots, \frac{1}{3}+\epsilon}_{n}, \underbrace{\frac{1}{4}+\epsilon, \ldots, \frac{1}{4}+\epsilon}_{n}, \underbrace{\epsilon, \ldots, \epsilon}_{n\times(\frac{1}{12\epsilon}-3)} \rangle$$

Here, we have $\epsilon = \epsilon'/(12\alpha+12)$. HH opens one bin for each replica of size $1/3+\epsilon$, and one bin for each two replicas of size $1/4+\epsilon$. To place replicas of size ϵ, it accumulates them to form multi-replicas of size no more than $\frac{1}{\alpha+1}+\epsilon$. The number of replicas in a multi-replica is upper-bounded by $\frac{1}{(\alpha+1)\epsilon}+1$ and there will be at least $n\times(\frac{1}{12\epsilon}-3)/(\frac{1}{(\alpha+1)\epsilon}+1) = n\times\frac{\alpha+1-3\epsilon'}{12+\epsilon'}$ multi-replicas. The algorithm places $\alpha-1$ multi-replicas in the same bin; hence, it opens at least $n\times\frac{\alpha+1-3\epsilon'}{(12+\epsilon')(\alpha-1)}$ bins for replicas of size ϵ. In total, HH opens at least $n\times(1.5+\frac{\alpha+1-3\epsilon'}{(12+\epsilon')(\alpha-1)})$ for packing σ.

An offline algorithm OFF can place one blue replica of size $1/3+\epsilon$, one blue replica of size $1/4+\epsilon$, and $1/12\epsilon-3$ blue replicas of size ϵ in the same bin. Note that there is an empty space of size $\frac{1}{3}+\epsilon$ in each bin. The total size of the replicas plus the reserved space will be $\frac{2}{3}+2\epsilon+\frac{1}{4}+\epsilon+(\frac{1}{12\epsilon}-3)\times\epsilon = 1$. Consequently, OFF places all blue replicas in at most $n/2$ bins. Since ϵ is a constant, there will be a constant number of replicas in each bin; hence, OFF can apply the shifting lemma to place the red replicas in $n/2+c$ bins for some constant c. In total, OFF opens $n+c$ bins for placing σ; consequently, the ratio between the number of bins used by the two algorithms is at least $1.5+\frac{\alpha+1-3\epsilon'}{(\alpha-1)(12+\epsilon')}$.

□

From Lemmas 41, 42, and 43 we get the following theorem.

Theorem 21. *The competitive ratio of* HH *with K classes ($K \geq 30$) is at least* $\max(1.597, l^*-\epsilon)$ *and at most* $\max(1.599, l^*)$*, where* $l^* = 1.5+\frac{\alpha+1}{12(\alpha-1)}$*, ϵ is an arbitrary small constant value, and* $\alpha = \lfloor\frac{\sqrt{4K+1}-1}{2}\rfloor$.

The bounds in the above theorem are tight for small values of K, e.g., for the suggested value of $K = 30$, the upper and lower bounds for the competitive ratio of HH almost match at 1.625. Note that when $K \to \infty$ we have $\alpha \to \infty$ and consequently the competitive ratio of HH converges to a value between 1.597 and 1.599.

7.4 General Lower Bound

In this section, we show that the competitive ratio of any online algorithm for the fault-tolerant server consolidation problem is at least 1.37. In the proof, we build sequences that contain only items of sizes $1/6 - 6\epsilon$, $1/2 + 2\epsilon$, and $2/3 + 2\epsilon$, where ϵ is a sufficiently small constant. The replicas for these items have sizes $x = 1/12 - 3\epsilon$, $y = 1/4 + \epsilon$, and $z = 1/3 + \epsilon$, respectively. In what follows, we consider a sequence of replicas rather than items. Consider a sequence $\sigma = \sigma_1\sigma_2\sigma_3$ in which σ_1, σ_2, and σ_3 are composed of n replicas of respectively sizes x, y, and z. Here, n is a sufficiently large even integer. We compare the number of bins used by any online algorithm $\mathbb{A}$ with that of OPT after packing sequences σ_1, $\sigma_1\sigma_2$, and $\sigma_1\sigma_2\sigma_3$.

Lemma 44. *Consider the sequence $\sigma = \sigma_1\sigma_2\sigma_3$ as defined above. We have $\text{OPT}(\sigma_1) = n/11 + c_1$, $\text{OPT}(\sigma_1\sigma_2) \leq n/2 + c_2$, and $\text{OPT}(\sigma_1\sigma_2\sigma_3) \leq n + c_3$, where c_1, c_2, and c_3 are constants.*

Proof. We present an offline algorithm OFF which places blue and red replicas separately. The algorithm places the blue replicas in a way that the size of the reserved space in each bin is just equal to the size of largest replica in that bin. To ensure a valid packing, OFF applies the shifting lemma (Lemma 36) to place the red replicas.

For packing σ_1, OFF places 11 blue replicas of size x in each bin. Hence, it opens at most $n/2 \times 1/11 + 1$ bins for placing the blue replicas of σ_1 (Note that there are $n/2$ blue replicas in σ_1). There will be an empty space of size larger than $1/12$ in case of a bin's failure. Using the shifting technique, OFF places the red replicas in the same number of bins (within an additive constant). Consequently, we have $\text{OPT}(\sigma_1) \leq n/11 + c_1$ for some constant c_1. For packing $\sigma_1\sigma_2$, OFF places two blue replicas of size x with two blue replica of size y in the same bin. The total size of the replicas in the bin will be $2(1/12 - 3\epsilon) + 2(1/4 + \epsilon) = 2/3 - 4\epsilon$; hence, there is enough space for another replica of size y. OFF opens at most $n \times 1/4 + 1$ bins for the blue replicas. Again, it applies the shifting technique for placing the red replicas in the same number of bins (within an additive constant). The total number of bins in such an offline solution will be $n/2 + c_2$ for some constant c_2. Finally, to place the blue replicas of $\sigma_1\sigma_2\sigma_3$, OFF places one replica of size x, one replica of size y, and one replica of size z in each bin. The size of the replicas in each bin will be $1/12 - 3\epsilon + 1/4 + \epsilon + 1/3 + \epsilon = 2/3 - \epsilon$; hence, there is an empty space of size $1/3 + \epsilon$ in case of a bin's failure. The number of opened bins for the blue replicas will be $3n/2 \times 1/3 = n/2$. Using the shifting technique, the red replicas can be packed in the same number of bins (again, within an additive constant). Hence, the total number of bins will be $n + c_3$ for some constant c_3.

□

The above lemma helps us prove the following theorem:

Theorem 22. *The competitive ratio of any online algorithm $\mathbb{A}$ for the fault-tolerant server consolidation problem is at least* 10/7.

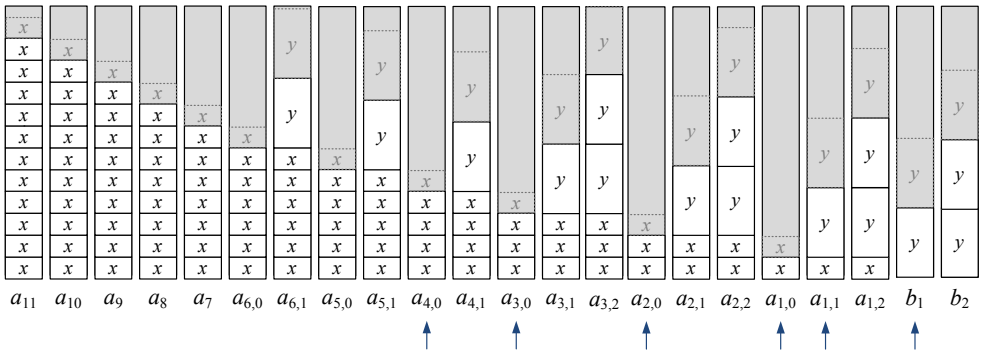


Figure 7.7: Potential bins after packing $\sigma_1\sigma_2$. Note that in a fault-tolerant packing, each bin needs to have an empty space of size at least equal to the largest hosted replica. Bins which have enough space for a replica of size z are indicated by arrows.

Proof. Consider the packing of $\mathbb{A}$ after packing $\sigma_1\sigma_2$. At this point the algorithm has placed n replicas of size x and n replicas of size y. Figure 7.7 shows all possible bins which include replicas of sizes x and y. Note that if a bin includes more than 6 replicas of size x, then there is no space for a replica of size y (to host a replica of size y, a bin requires a space of size $2y > 1/2$). Similarly, if a bin contains more than three replicas of size x, then it includes no more than one replica of size y. For $1 \leq i \leq 6$, let $a_{i,j}$ denote the number of bins which include i replicas of size x and j replicas of size y ($j \in \{0, 1, 2\}$). Also, for $7 \leq i \leq 11$, let a_i denote the number of bins which only include i replicas of size x. Finally, let b_1 (respectively b_2) denote the number of bins which include only one (respectively two) replicas of size y (and no replica of size x). Define the following variables:

$$\begin{cases} S_1 = a_{1,1} \\ S_2 = a_{1,2} + a_{2,2} + a_{3,2} \\ S_3 = a_{1,0} + a_{2,0} + a_{3,0} + a_{4,0} \\ S_4 = \quad a_{2,1} + a_{3,1} + a_{4,1} \\ S_5 = \quad\quad\quad\quad a_{5,1} + a_{6,1} \\ S_6 = \quad\quad\quad\quad a_{5,0} + a_{6,0} + \sum_{i=7}^{12} a_i \end{cases}$$

The number of bins used by $\mathbb{A}$ to pack σ_1 is equal to the number of bins which include a replica of size x. We have:

$$\mathbb{A}(\sigma_1) = \sum_{i=1}^{6} S_i \tag{7.1}$$

Counting the number of replicas of size x we get:

$$n \leq S_1 + 3S_2 + 4S_3 + 4S_4 + 6S_5 + 11S_6 \tag{7.2}$$

Similarly, for packing $\sigma_1\sigma_2$ we have:

$$\mathbb{A}(\sigma_1\sigma_2) = \sum_{i=1}^{6} S_i + b_1 + b_2 \tag{7.3}$$

And counting the number of replicas of size y we get:

$$n = S_1 + 2S_2 + S_4 + S_5 + b_1 + 2b_2 \tag{7.4}$$

Next, we count the number of opened bins that potentially can host a replica of size z. First, if a bin contains more than 4 replicas of size x, it cannot include a replica of size z; otherwise, its level will be at least $5/12+1/3-14\epsilon$ and its empty space will be at most $1/4+14\epsilon$ which is not enough for another replica of size z. With similar arguments, a bin that contains two replicas of size y cannot host a replica of size z and the same holds for a bin with one replica of size y and more than one replica of size x. Furthermore, no two replicas of size z can be placed in the same bin. Hence, the number of bins which can host a replica of size z is at most $a_{1,0} + a_{1,1} + a_{2,0} + a_{3,0} + a_{4,0} + b_1 = S_1 + S_3 + b_1$. Except those replicas which can be placed in these bins, for any other replica in σ_3 a new bin should be opened. Hence, we have:

$$\mathbb{A}(\sigma_1\sigma_2\sigma_3) \geq S_2 + S_4 + S_5 + S_6 + b_2 + n \tag{7.5}$$

Let r_A denote the competitive ratio of $\mathbb{A}$. By Lemma 44, we have $\mathbb{A}(\sigma_1) \leq r_A \times \textsc{Opt}(\sigma_1) = r_A \times (n/11 + c_1)$. Similarly, $\mathbb{A}(\sigma_1\sigma_2) \leq r_A \times (n/2 + c_2)$ and $\mathbb{A}(\sigma_1\sigma_2\sigma_3) \leq r_A \times (n + c_3)$. Here, c_1, c_2 and c_3 are some constant values. From Equations (1),(2),(3),(4), and (5), we respectively get the system of equations depicted in Figure 7.8. Since all values of S_i $(1 \leq i \leq 6)$ are positive, summing all equations in that system, we get $35/22 \times r_A \geq 25/11 - c/n$ where c is a constant. This implies that r_A is lower bounded by $10/7$ for large values of n.

□

7.5 Remarks

There is a gap between the lower bound of $10/7 \approx 1.42$ for the competitive ratio of any online algorithm for fault-tolerant server consolidation and the best upper bound of 1.625 given by the Horizontal Harmonic algorithm. Closing this gap seems to be difficult considering the fact that there is still a gap between the best upper and lower bound for the classic online bin packing problem. We conjecture that both upper

$(n/11 + c_1) \times r_A$	$\geq$	S_1	$+S_2$	$+S_3$	$+S_4$	$+S_5$	$+S_6$			
$(n/2 + c_2) \times r_A$	$\geq$	S_1	$+S_2$	$+S_3$	$+S_4$	$+S_5$	$+S_6$	$+b_1$	$+b_2$	
$(n + c_3) \times r_A$	$\geq$		S_2		$+S_4$	$+S_5$	$+S_6$		$+b_2$	$+n$
$-\frac{3}{11}n$	$\geq$	$-\frac{3}{11}S_1$	$-\frac{9}{11}S_2$	$-\frac{12}{11}S_3$	$-\frac{12}{11}S_4$	$-\frac{18}{11}S_5$	$-3S_6$			
$-n$	$=$	$-S_1$	$-2S_2$		$-S_4$	$-S_5$		$-b_1$	$-2b_2$	
$\frac{35}{22}n \times r_A - \frac{14}{11}n + r_A(c_1 + c_2 + c_3)$	$\geq$	$\frac{8}{11}S_1$	$+\frac{2}{11}S_2$	$+\frac{10}{11}S_3$	$+\frac{10}{11}S_4$	$+\frac{4}{11}S_5$				$+n$

Figure 7.8: Summary of the inequalities in the lower bound argument.

and lower bounds for the fault-tolerant server consolidation problem can be improved and leave it as a future work.

An extension of the bin packing problem, called *vector packing* problem, has been used to address the server consolidation problem where the load of a client is defined by a vector rather than a single value [135]. Fields of this vector represents the amount of different resources required by the client. In order to maintain high performance, a server should not be overload with respect to any of these fields. The ideas presented in this chapter might be extended to provide a fault-tolerant vector packing algorithm. Other directions for future work include considering variable sized bins (i.e., different types of servers) and studying settings in which there are incompatibility constraints where some items are not allowed to be placed together in the same bin.

Chapter 8

List Update and Compression

In this chapter, we consider an application of list update in the context of compression. In this application, a list update algorithm is used to encode a given string σ. To increase the locality of the sequence, the Burrows-Wheeler transform (BWT) can be applied before using the list update algorithm. Previous work has shown (e.g., [29, 5, 21, 44, 1]) that careful study of the list update step leads to better BWT compression. Surprisingly, the theoretical study of list update algorithms for compression has lagged behind its use in real practice. To be more precise, the standard model for list update considers a linear cost-of-access model while compression incurs a *logarithmic cost of access*, i.e., accessing item i in the list has cost $\Theta(i)$ in the standard model but $\Theta(\lg i)$ in compression applications. These models have been shown, in general, not to be equivalent [65]. In this chapter, we give the first theoretical proof that the commonly used Move-To-Front (MTF) has good performance under the compression logarithmic cost-of-access model. This has long been known in practice but a formal proof under the logarithmic cost model was missing. We also refine the online compression model to reflect its use for compression under the advice framework. The advice model was initially a purely theoretical construct; however, we show that surprisingly, this seemingly unrealistic model can be used to produce better multi-pass compression algorithms. More precisely, we introduce an 'almost-online' list update algorithm, called BIB, which results in a compression scheme which is superior to schemes using standard online algorithms, in particular those of MTF and TIMESTAMP. For example, for the files in the standard Canterbury Corpus [42], the compression ratio of the scheme that uses BIB is 33.66 on average, while the compression ratios for the schemes that use MTF and TIMESTAMP are respectively 34.25 and 36.30.

8.1 Introduction

List update algorithms have been extensively used for compression purposes, both directly and as a post-processing step of the Burrows-Wheeler Transform (BWT) compression. In this application, each character of a text is treated as an item in the list, and the text as the input sequence which is parsed (revealed) in a sequential manner. A compression algorithm can be devised from a list update algorithm $\mathbb{A}$ by writing the access cost of $\mathbb{A}$ for serving each character in the compressed file. Hence, the size of the compressed file is proportional to the sum of the logarithm of the access costs of the list update algorithm. List

update algorithms have smaller costs when the input sequence has a high level of *locality* (i.e., recently requested items are expected to be requested soon again). In this context, many compression schemes (e.g., the bZip family) apply the BWT to the input file. The result will be a reversible sequence which has a high level of locality. The next stage is to apply a list update algorithm–in particular MTF–on the BWT sequence. This way, a compression algorithm encodes a set of small numbers (instead of characters) using a self-delimiting binary code, e.g., the Elias Gamma code, or an entropy code, e.g., Huffman code. A decompressor algorithm decodes a sequence by simply reading the access costs from the compressed file and applying the same online algorithm used for compression to maintain the list of items. Here, the online constraint plays a critical role since for decoding the tth item in a sequence, an algorithm can only look at the previously decoded items. This implies that, in its most basic form, an offline list update algorithm cannot be used for compression purposes.

Although there has been a great deal of interest in theoretical analysis of the list update problem, it is known that the standard model is not suitable for practical scenarios which include maintaining a self-adjusting linked list [115, 112] and compression [65]. In the context of compression, as mentioned earlier, a compressor encodes the index of an accessed item in the compressed file. Assuming that it uses a self-delimiting binary code, it encodes an index i in $\Theta(\lg i)$ bits. However, under the standard model, the algorithm is charged i units for accessing index i. This discrepancy was first reported in [65] where the authors introduced a 'logarithmic model' in which accessing an item in the ith position has a cost of $c \lg i + b$, where c and b are positive integers. There they observed that there are competitive online algorithms under the standard model which are non-competitive under the logarithmic model. For example, the algorithm MF2 which moves an item half way towards the front is known to be 4-competitive under the standard model, while it is not constant competitive under the logarithmic model [65]. This suggests that not all good list update algorithms are good for compression.

In their seminal paper, Sleator and Tarjan [134] proved that the competitive ratio of MTF is at most 2 for any model in which the cost of accessing an item in the ith position is a convex function of i. The status of the problem is open when the access cost is concave. This is particularly the case when the access cost is a logarithmic function of the accessed index. Although many algorithms are empirically compared when used for compression (where encoding an index i costs $\Theta(\lg i)$), as stated by Dorrigiv et al. [65], *'it remains an open question to determine the competitive ratios of the various list update algorithms under the $c \lg i + b$ cost of access model'*. In Section 8.2, we partially answer this question by showing that MTF has a competitive ratio of at most 2 under the logarithmic model. This can be seen as a justification of the empirically-observed advantage of MTF over other online list update algorithms for compression.

Recall that under the advice model for analysis of online problems, the 'online constraint' is relaxed and an online algorithm receives partial information about the input sequence in the form of some advice generated by an offline oracle. In the context of compression, the advice is included in the compressed file to provide some hints for the list update when it is used for decompressing. This concept has been implicitly studied in [44], where it is shown that, for context-independent sequences, switching between members of 'Best x of $2x$' family of algorithms and a variant of MTF might result in better compression schemes than MTF schemes. In these schemes, the indices in which the compression algorithm is alternated between, referred as 'switching points', are included in the compressed file. Providing the right switching points which guarantees a good compression scheme is a bottleneck of that approach. In Section 8.3, we provide a simple and fast online list update algorithm, called BIB, which receives sparse advice on the compressed file. We empirically compare the BWT compression schemes resulting from BIB, MTF, and TIMESTAMP algorithms on the standard corpora. We observe that the compression scheme resulting from BIB is generally better than the schemes of MTF and TIMESTAMP. For example, for the files in

the Canterbury Corpus, the compression ratio of the scheme that uses BIB is 33.66 on average, while the compression ratios for the schemes that use MTF and TIMESTAMP are respectively 34.25 and 36.30.

8.2 MTF under the Logarithmic Cost Model

In this section, we show that the competitive ratio of MTF is at most 2 when accessing the item in the ith position has a cost of $c \lg i + b$.

Theorem 23. *Consider the list update problem under the* logarithmic model *in which accessing an item in the ith position costs $c \lg i + b$, and a paid exchange has a cost of c. Here, b and c are constant positive integers. The competitive ratio of* MTF *is at most 2 under this model.*

To prove the theorem, we use the potential function method. At any time t, we say a pair (a, b) of items form an *inversion* if a appears before b in the list maintained by MTF while b appears before a in the list maintained by OPT. Let $I_{\text{MTF}}(x)$ and $I_{\text{OPT}}(x)$ respectively denote the index of x in the lists maintained by MTF and OPT. An inversion (a, b) has type 1 if $I_{\text{MTF}}(a) > I_{\text{OPT}}(b)$ and type 2 otherwise (see Figure 8.1). Define the *weight* of an inversion (a, b) to be $w(a, b) = \frac{\tau}{r+1}$, where $r = \max\{I_{\text{MTF}}(a), I_{\text{OPT}}(b)\}$ and $\tau = \frac{c}{\ln 2}$. In other words, an inversion (a, b) has weight $\frac{\tau}{I_{\text{MTF}}(a)+1}$ if it has type 1 and weight $\frac{\tau}{I_{\text{OPT}}(b)+1}$ otherwise.

We define the potential at each time t as the summation of the weights of all inversions at that time, namely, $\Phi_t = \sum_{(x,y)} w(x, y)$. Consider a sequence of *events* where each event is defined as a set of operation performed by MTF and/or OPT. For each event at time t we define the amortized cost a_t as the cost paid by MTF for the event plus the increase in potential after the event, i.e., $a_t = \text{MTF}_t + \Phi_t - \Phi_{t-1}$. So the total cost of MTF for serving a sequence σ is $\sum_t a_t - (\Phi_{last} - \Phi_0)$. The value of $\Phi_{last} - \Phi_0$ is independent of the length of sequence. Hence, to prove the competitiveness MTF, it is enough to bound the amortized cost. Let OPT_t be the cost paid by OPT for event t. To prove Theorem 23, it suffices to show that for each event we have $a_t \leq 2\,\text{OPT}_t$. There are four types of events, which are listed below.

- Event 1: MTF accesses an item y at index k, OPT access y at index $j < k$, and MTF moves y to the front.
- Event 2: MTF accesses an item y at index k, OPT access y at index $j \geq k$, and MTF moves y to the front.
- Event 3: After an access to an item in index j, OPT makes a free exchange.[1]
- Event 4: OPT makes a paid exchange.

We separately address any of the above events. We start with the following lemma.

Lemma 45. *Assume* MTF *moves an item y to the front. The total weight of newly created inversions, plus the total increase in the weight of the old inversions is at most $\tau \times \sum_{i=2}^{j} \frac{1}{i}$, where j is the index of y in the list maintained by* OPT.

[1] Recall that, under the standard model, there are optimal offline algorithms which only make use of paid exchanges [120]. Therefore, it is usually assumed that OPT does not make a free exchange. It is not clear if the same statement holds for the logarithmic model. Hence, we consider the event in which OPT makes a free exchange.

(a) An inversion of type 1 has weight $\tau/(I_{\text{MTF}(a)}+1)$ (b) An inversion of type 2 has weight $\tau/(I_{\text{OPT}(b)}+1)$

Figure 8.1: The weights associated with inversions of different types

Proof. Assume there are $p < j$ items which are before y in the OPT list (i.e., before index j) and do not form an inversion with y before moving y to the front. Moving y to the front creates p new inversions of the form (y, x_i), each having weight $\frac{\tau}{I_{opt}(x_i)}$. Moreover, consider the inversions of the form (y, z_j) which have type 1 before moving y to the front. Since z_j appears before y in the OPT list, there are exactly $j-p-1$ inversions of this form. After moving y to the front, the weight of these inversions will increase from $\frac{\tau}{k}$ to $\frac{\tau}{I_{opt}(z_j)}$. So, the new inversions of the form (y, x) and those of the form (y, z) which had type 1 before moving y have a total weight of $\tau \times (\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{j})$ after moving y to the front. For the inversions of the form (y, z') which have type 2 before moving y, the weight is defined by the index of z' in the OPT list. Hence, their type and weight remain unchanged when MTF moves y to the front. In sum, the total increase in the weight of inversions which involve y will be no more than $\tau \times \sum_{i=2}^{j} \frac{1}{i}$. For the inversions of the form (x, z) which do not involve y, moving y to the front does not increase the weight. For these inversions, the weight is defined either via the index of x in the MTF's list or the index of z in the OPT list. In the first case, moving y to the front decreases the weight of the inversion (it increases the index of x by 1 unit). In the second case, moving y to the front does not change the weight of the inversion—although its type might change from 2 to 1, if x is located at index j of the MTF's list before the access. □

Using the above lemma, we are ready to prove Theorem 23:

Proof of Theorem 23. For each event type, we show that $a_t \leq 2\,\text{OPT}_t$. Recall that a_t and OPT_t are respectively amortized cost of MTF and cost of OPT for the tth event.

Event 1: Assume there is an access to item y and we have $k > j$. MTF incurs a cost of $c \lg k + b$ to access y and move it to the front. We have:

$$c \lg k + b \leq \frac{c}{\ln 2} \ln k + b = \tau \ln k + b \leq \tau(1 + 1/2 + \ldots + 1/k - \gamma) + b$$

Here, γ is the Euler constant and we have $\gamma \approx 0.557$. Let v denote the number of inversions in the form (x, y). Among the $k-1$ items which are in front of y in the MTF's list, at most $j-1$ of them are also in front of y in the OPT list. Hence, at least $k-j$ items give inversions of the form (x, y), i.e., $v \geq k-j$. An inversion of the form (x_i, y) has a weight $\frac{\tau}{I_{\text{MTF}}(x_i)}$ if it has type 1, and weight $\frac{\tau}{j}$ if it has type 2. Hence, the total weight of the inversions in the form (x_i, y) is at least $\tau \times (\frac{1}{j} + \frac{1}{j+1} + \ldots + \frac{1}{k})$. After moving y to the front, all these inversions and consequently the potential decreases by a value of at least $\tau \times (\frac{1}{j} + \frac{1}{j+1} + \ldots + \frac{1}{k})$. Be Lemma 45, after moving y to the front, the total increase in the amortized

cost will be at most $\tau \times \sum_{i=2}^{j} \frac{1}{i}$. So, the amortized cost of the event will be at most:

$$\begin{aligned}
a_t \leq & \tau(1 + 1/2 + \ldots + 1/k - \gamma) + b && \text{[upper bound for access cost]} \\
& - \tau(1/j + 1/(j+1) \ldots + 1/k) && \text{[lower bd. for decrease in potential]} \\
& + \tau(1/2 + 1/3 + \ldots + 1/j) && \text{[upper bd. for increase in potential]} \\
& = \tau\left(2(1 + 1/2 + \ldots + 1/j) - 1 - \gamma - 1/j\right) + b \\
& < \tau(2 \ln j + \gamma - 1) + b < 2c \lg j + b = 2\,\textsc{Opt}_t
\end{aligned}$$

In the above inequalities, we made use of the following inequalities that holds for all values of n:

$$\ln n + \gamma < 1 + 1/2 + \ldots + 1/n < \ln n + \gamma + 1/(2n)$$

Event 2: Assume there is an access to item y, we have $k \leq j$. MTF incurs a cost of $c \lg k + b \leq c \lg j + b$ units to access y and move it to the front. By Lemma 45, after moving y to the front, the total increase in the amortized cost will be at most $\tau \times \sum_{i=2}^{j} \frac{1}{i} < c \lg j$. So, the amortized cost of the event will be at most $2c \lg j + b \leq 2\,\textsc{Opt}_t$.

Event 3: Assume OPT makes a free exchange to move y closer to the front. Since OPT and MTF make their moves simultaneously, y is in front of the MTF's list. Hence, the free exchange by OPT does not create new inversions, and there are no inversions of the form (x, y). All inversions of the form (y, z) have type 2 and their weight is $\frac{\tau}{I_{opt}(z)+1}$. If the OPT's free exchange moves y in front of z, the inversion is removed. Otherwise, the weight of the inversion remains unchanged. In both cases, the contribution to the amortized cost is non-positive. For the inversions of the form (x, z) which do not involve y, moving y closer to the front does not increase their weight. The weight is defined either via the index of x in the MTF's list or the index of z in the OPT list. In the first case, the weight of the inversion does not change; but its type might change from 1 to 2 (if x is located at index $j - 1$ of the OPT list before the move). In the second case, moving y to the front decreases the weight of the inversion (the index of z in the OPT list increases 1 unit if y is moved to the front of z). To summarize, when OPT makes a free exchange, the amortized cost will be non-positive, i.e., $a_t \leq 0 = \textsc{Opt}_t$.

Event 4: Assume OPT makes a paid exchange to swap the position of two items (y, z) in the list. Assume y and z are located at indices k and $k + 1$ before the swap. The swap causes the indices of all items except z to increase or remain unchanged; hence, the weight of all inversions decrease or remain unchanged, except the inversions of the form (x, z) $(x \notin \{y, z\})$, for which the weight might increase from $\frac{\tau}{k+2}$ to $\frac{\tau}{k+1}$. This implies that the inversion has type 2 after the swap (otherwise the weight of inversion is defined by the index of x in the MTF's list and remains unchanged). An inversion (x, z) of type 2 implies that x is located before k in the MTF's list. Hence, there are at most $k - 1$ such inversions. The swap might create a new inversion of weight $\frac{\tau}{k+1}$. In total, the amortized cost will be at most:

$$a_t \leq \tau \times \left(\frac{1}{k+1} + (k-1)\left(\frac{1}{k+1} - \frac{1}{k+2}\right)\right) \;<\; \frac{2\tau}{k+2} \;<\; 2c = 2\,\textsc{Opt}_t$$

Recall that OPT incurs a cost of c for making a paid exchange. Consequently, for any event t, we have $a_t \leq 2\,\textsc{Opt}_t$ which completes the proof. □

8.3 Compression Model

In the logarithmic model for the list update problem, to rearrange the list, an algorithm has to use paid exchanges. In the context of compression, however, an algorithm can rearrange the list free of charge. This is because the size of the compressed file is equal to the total access cost, and not the cost involved in paid exchanges. To resolve this issue, one might define a compression model under which an algorithm can rearrange the whole list, free of charge. However, it can be easily verified that all online algorithms are non-competitive under such model: An adversary can always ask for the last item in the list maintained by an online algorithm so that it incurs a cost of l on each request. At the same time Opt can rearrange the list, free of charge, so that the next requested item is in the front of its list. So Opt incurs a cost of 1 on each request. This argument gives a competitive ratio of l for lists of size l, for any online algorithm which is the worst ratio for any algorithm. In other words, all online algorithms are equally bad under such a model. This is because, in practice, no online algorithm can make use of the extra power provided in the form of free rearrangements of the whole list. In fact, most existing online algorithms for the list update problem only make use of free exchanges. To analyze these algorithms for compression purposes, we can compare them under a model which does not allow rearrangement of the list, or allows it only though paid exchanges. In both cases, the proof of Theorem 23 can be applied to state that Mtf is a constant competitive algorithm.

One way to leverage the true power of list update algorithms for compression purposes is to provide them with some insight about the structure of the input sequence. This can be done by including a set of advice bits in the compressed file which encode some information about the encoded sequence (and, consequently, how it should be decompressed). As an example, consider a compression scheme that runs both Mtf and Timestamp on an input sequence and selects the better algorithm to compress the file (i.e., writes the access costs of that algorithm in the file). To be able to decompress, it is sufficient to include a single bit of advice in the compressed file which indicates the algorithm used to compress the file. Based on this idea, we introduce a simple yet effective compression algorithm, called *Best-In-Block* (Bib).

8.3.1 BIB Algorithm

It is well-know that, for sequences with high levels of locality (e.g., a BWT transformation of a text file), Mtf generally outperforms other online algorithms [17, 4, 65]. However, even for these sequences, in many occasions Timestamp is better than Mtf. Based on these observations, Bib applies the better algorithm (between Mtf and Timestamp) for subsequences of a large sequence.

To compress a given sequence (file) σ, Bib divides the sequence into blocks[2] of fixed size β. For each block, it computes the cost of both Mtf and Timestamp for compressing that block, and uses the better algorithm to compress it. More precisely, the access cost of the better algorithm is added to the compressed file. Moreover, for each block, a single bit is added to the file to indicate which algorithm has been used to compress the block. This way, the overhead of including the advice in the compressed file resulted by Bib is $\lceil n/\beta \rceil$, with the size of the block also included as a part of the advice string.

The performance of Bib is dependant on the block size (β). When compressing a file, in theory, we can test all values of β and select the value which results in the smallest file size. However, this results in a slow

[2]This notion of block should not be confused by the blocks defined for BWT transform. In our experiments, the blocks for BTW transform have a size of 9×10^5 bytes; consequently, there is a single BWT block for all studied files.

compression algorithm. We cannot use a straightforward binary-search method to find the 'best' value of β since the size of the compressed file is not a unimodal function of β, i.e., there is a chance of selecting local minimum values for β which are far from the best achievable value. We suggest the value of β be selected through a linear sampling of all values in the range $(1, n)$, where n is the number of characters in the file. The number of samples gives a compromise between the size of the compressed file and the speed of the compressor. In our experiments, we took a small number of candidate values which were linearly distributed in the domain of β, i.e., numbers smaller than n. This was followed by a quick sequential linear search for improving the selected candidate.

8.3.2 Experimental Results

In this section, we compare the performance of Bib with that of Mtf and Timestamp through an experimental study. In the context of compression, it has been observed that Mtf and Timestamp outperform other list update algorithms. Hence, we do not include other algorithms in our comparison. We test algorithms on Calgary and Canterbury corpora, which are the standard benchmark for comparing compression algorithms. As is the case for practical compression schemes, we run our list update algorithms as a secondary stage after applying BWT on the input files. We treat the resulting files as a sequence of ASCII characters, and assume the list of the characters is initially sorted in the order of appearance in the ASCII table. When applying Mtf, Timestamp, and Bib on the input sequence, we include the access cost (i.e., the index of the accessed item) in the compressed file. In doing so, we use the Elias Gamma coding to encode an index i using $2\lfloor \lg i \rfloor + 1$ bits. For the Bib algorithm, we also encode the value of β (block size) using the Elias Gamma coding. To select the block size (β) for the Bib algorithms, we used a fast linear sampling. Depending on the file size, we took 5 to 10 samples in the range $(5, n)$, where n is the length of the input sequence. This was followed by a local search close to the best observed value of β (5 random samples). It should be mentioned that our focus is on comparing the effect of different list update algorithms for compression; consequently, we have not applied any optimization used *after* applying the list update algorithms, in the presumption that all schemes equally benefit from these post-optimization techniques.

Table 8.1 gives a summary of the results. It can be seen that the compression ratio (the ratio between the compressed and the original files, scaled by 100) is smaller (i.e., better) when the Bib algorithm is used compared to when Mtf or Timestamp are used. The only exceptions are 'progp' and 'fields.c' for which the overhead of including the advice bits results in slightly worse compression schemes. On average, for the files in the standard Canterbury Corpus, the compression ratio is 33.66 when Bib is used, while the average compression ratios of Mtf and Timestamp schemes are respectively 34.25 and 36.30. Similarly, for the Calgary Corpus, on average, the scheme that uses Bib has a better compression ratio of 36.54 compared to respectively 36.99 and 39.31 ratios of Mtf and Timestamp. For perspective, we have also included compression ratios of a few of some commonly used compression algorithms in Table 8.1. These are bzip-6, gzip-b, and huffword2. The entries are taken from the Canterbury corpus website [42] (note that some entries are not reported). bzip6 is a successor of bzip2 and is based on BWT and Mtf. gzip-b is a combination LZ77 (Lampel-Ziv) and Huffman coding. huffword2 is a word-based model based on Huffman coding. Note that even without any optimization, the compression schemes which are based on list update algorithms outperform huffword2. It is expected that, replacing the Mtf with Bib in bzip6, the resulting compression scheme would outperform bzip6.

Recall that we have used a relatively fast compression algorithm which only takes a small number of

file name	original file size (bytes)	MTF	TS	BIB	bzip-6	gzip-b	huffword2	block Size	compressed file size (bytes)	advice cost (bits)
Canterbury Corpus										
alice29.txt	152089	33.1365	33.4278	**32.2351**	28.13	35.63	38.63	49	49026	3115
asyoulik.txt	125179	36.9751	37.0797	**35.8295**	31.38	39.00	44.88	32	44851	3923
cp.html	24603	36.1582	39.3692	**35.955**	30.50	32.38	63.13	76	8846	337
fields.c	11150	**30.0717**	34.9686	30.0987	26.13	28.00	52.75	1628	3356	28
grammar.lsp	3721	35.125	41.1986	**35.0175**	31.88	33.13	59.25	73	1303	64
kennedy.xls	1029744	39.2399	38.5065	**38.4435**	11.25	20.38	61.63	1035	395870	1016
lcet10.txt	426754	30.7627	31.0118	**29.9367**	25.00	33.88	33.50	42	127756	10172
plrabn12.txt	481861	36.1451	35.4152	**34.7567**	29.88	40.38	40.75	38	167479	12692
ptt5	513216	20.156	19.5808	**19.5797**	9.63	10.25	25.38	519	100486	1008
sum	38240	37.5732	41.8279	**37.0868**	32.63	33.38	86.88	62	14182	628
xargs.1	4227	41.4242	46.96	**41.4242**	39.13	41.38	70.88	55	1751	88
Calgary Corpus										
bib	111261	30.5013	32.3195	**30.1948**	24.38	31.38	50.00	117	33595	964
book1	768771	35.7117	34.6887	**34.1462**	31.13	40.63	38.75	39	262506	19724
book2	610856	31.1388	31.4832	**30.5859**	25.75	33.75	37.13	507	186836	1222
geo	102400	79.251	78.4229	**77.8457**	56.00	66.75	113.25	211	79714	501
news	377109	36.2137	38.6721	**35.6995**	31.38	38.25	51.38	38	134626	9935
obj1	21504	57.2359	59.8726	**56.5895**	48.38	48.00	99.88	46	12169	479
obj2	246814	37.9043	41.9093	**37.8098**	30.75	32.88	71.50	121	93320	2053
paper1	53161	34.7191	37.6855	**34.388**	30.75	34.88	53.00	59	18281	913
paper2	82199	34.869	36.0369	**34.2303**	30.25	36.13	45.88	88	28137	948
paper3	46526	37.7724	39.7176	**37.076**	-	-	-	52	17250	906
paper4	13286	41.3367	44.6937	**40.9303**	-	-	-	34	5438	402
paper5	11954	42.3624	46.863	**42.2118**	-	-	-	17	5046	713
paper6	38105	35.2552	38.8558	**35.1371**	-	-	-	84	13389	467
pic	513216	20.156	19.5808	**19.5797**	9.63	10.25	25.38	519	100486	1008
progc	39611	35.0711	38.5247	**34.9221**	31.25	33.50	56.88	85	13833	480
progl	71646	26.3295	29.4308	**26.2974**	21.50	22.50	37.75	221	18841	340
progp	49379	**26.0313**	30.2193	26.0394	21.38	22.63	38.50	6030	12858	34
trans	93695	24.1176	28.6867	**24.0984**	18.75	20.13	46.13	475	22579	215

Table 8.1: The compression ratios (percentage) for different compression schemes for Canterbury and Calgary corpora. Except for two files, BIB outperforms MTF and TIMESTAMP. The blocks size, compressed file size, and the length of advice string for BIB are also included. Note that the advice cost is relatively small compared to the total size of the compressed file.

samples to find an appropriate value of β. Spending more time on finding better values for β (by taking more samples) slightly improves the performance ratio of BIB. Also, note that the sampling algorithm has selected relatively small values for β. This implies that BIB frequently changes strategy from MTF to TIMESTAMP and vice versa. The only exceptions are 'progp' and 'fields.c', for which MTF has a big advantage over TIMESTAMP, and it is very unlikely that TIMESTAMP offers any advantages over MTF for any of the blocks; consequently, a large value of β is selected to decrease the overhead in including advice bits. Yet even that overhead causes BIB to have a slightly worse compression ratio than MTF though the difference in practice is negligible.

8.4 Remarks

We proved formally that MTF is competitive under the logarithmic model. It would be interesting to investigate if the same holds for other list update algorithms but this remains as future work. We conjecture that TIMESTAMP and its related family of algorithms are also constant competitive under the logarithmic model.

BIB might be the simplest algorithm which makes use of some advice included in the compressed file. Even this simple algorithm results in a compression scheme which is better than MTF and TIMESTAMP schemes. Although the improvement is small, it is on par with other engineering improvements for known compression schemes and it might shine light towards providing more sophisticated algorithms with advice that offer more significant improvements.

Part V

Conclusions

Chapter 9

Conclusions

In this thesis, we considered some alternative methods for finer analysis of online bin packing and list update problems. We theoretically and also experimentally analyzed these problems to close the gap between the existing theory and observations made in practice. We also studied the problems under new models and assumptions which are more realistic for certain applications of these problems.

In Part II, we introduced bin packing algorithms that have better competitive ratios than Best Fit and First Fit algorithms. At the same time, they have comparable average-case behavior to Best Fit and First Fit. In other words, we show that the competitive ratio of major bin packing algorithms can be improved without giving up on the average-case performance. The current champion among the bin packing algorithms is Harmonic++ with a competitive ratio of 1.588. Whether this algorithm can be modified to boost its average-case performance remains an open problem.

In Part III, we considered bin packing and list update problems under the advice model of complexity. Our interest in studying the advice framework is mainly theoretical. However, this model is expected to find applications in practice; we observed such an application in the context of list update and compression in Chapter 8. We answered several questions about the advice complexity of bin packing and list update. In both cases, we proved (almost) tight upper and lower bounds to achieve optimal solutions. We also presented algorithms that break the lower bounds on competitive ratio of any online algorithm by receiving advice of sub-linear size (logarithmic-size advice for bin packing and constant-size advice for list update). In the case of the bin packing problem, we proved that the competitive ratio can be further improved with advice of linear size.

In Part IV, we considered the real-world applications of the two problems. In Chapter 7, we studied the online server consolidation problem as a fault-tolerant variant of the bin packing problem. An application of this problem is earlier studied for tenant placement in the Cloud [127]. We investigated the theoretical aspects of the problem under the framework of competitive analysis and presented upper and lower bounds for the competitive ratios of two existing heuristics for the problem. As an alternative to these heuristics, we presented a simple algorithm that runs in linear time and is faster than its counterparts. The competitive ratio of the new algorithm is no more than 1.625 which is better than the other algorithms whose competitive ratios are lower-bounded by 2. Finally, we proved a general lower bound of $10/7 > 1.42$ for the competitive ratio of any online algorithm.

In Chapter 8, we considered an application of list update algorithms for compression purposes. We observed that the standard model for the list update is not suitable for studying the existing algorithms in the context of compression. We studied the compression model in which accessing an item at index i has a logarithmic cost rather than linear cost. We showed that Move-To-Front is competitive under this model; this indicates that MTF works well for compressing all files, even those generated adversarially. Whether the same statement holds for other list update algorithms remains an open question. We also introduced a new compression scheme which is based on including some bits of advice in the compressed file. We experimentally observed that this scheme outperforms other compression schemes which are based on the list update algorithms. Further investigation on how advice can be helpful for compression remains as a future work.

References

[1] Jürgen Abel. Post BWT stages of the burrows-wheeler compression algorithm. *Softw. Pract. Exper.*, 40(9):751–777, 2010. 102

[2] Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.*, 27:682–693, 1998. 13, 14, 15

[3] Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *J. Comput. Systems Sci.*, 70(2):145–175, 2005. 17

[4] Susanne Albers and Sonja Lauer. On list update with locality of reference. In *Proc. 35th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 5125 of *Lecture Notes in Comput. Sci., Springer*, pages 96–107, 2008. 18, 107

[5] Susanne Albers and Michael Mitzenmacher. Average case analyses of list update algorithms, with applications to data compression. In *Proc. 23rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1099 of *Lecture Notes in Comput. Sci., Springer*, pages 514–525, 1996. 102

[6] Susanne Albers and Michael Mitzenmacher. Average-case analyses of First Fit and Random Fit bin packing. In *Proc. 9th Symp. on Discrete Algorithms (SODA)*, pages 290–299. Society for Industrial and Applied Mathematics, 1998. 34

[7] Susanne Albers and Michael Mitzenmacher. Average case analyses of list update algorithms, with applications to data compression. *Algorithmica*, 21(3):312–329, 1998. 16

[8] Susanne Albers, Bernhard von Stengel, and Ralph Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Inform. Process. Lett.*, 56:135–139, 1995. 14, 16, 80

[9] Susanne Albers and Jeffery Westbrook. Self-organizing data structures. In *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 13–51, 1996. 3, 13

[10] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Systems Sci.*, 58(1):137–147, 1999. 55

[11] Amazon EC2. http://aws.amazon.com/ec2/. Accessed: 2014-07-23. 83

[12] Christoph Ambühl. Offline list update is NP-hard. In *Proc. 8th European Symp. on Algorithms (ESA)*, volume 1879 of *Lecture Notes in Comput. Sci., Springer*, pages 42–51, 2000. 14, 80

[13] Christoph Ambühl, Bernd Gärtner, and Bernhard von Stengel. A new lower bound for the list update problem in the partial cost model. *Theoret. Comput. Sci.*, 268:3–16, 2001. 16

[14] Christoph Ambühl, Bernd Gärtner, and Bernhard von Stengel. Optimal projective algorithms for the list update problem. *CoRR*, abs/1002.2440, 2010. 16

[15] Christoph Ambühl, Bernd Gärtner, and Bernhard von Stengel. Optimal lower bounds for projective list update algorithms. *ACM Trans. Algorithms*, 9(4):31, 2013. 16

[16] Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. On the separation and equivalence of paging strategies. In *Proc. 18th Symp. on Discrete Algorithms (SODA)*, pages 229–237, 2007. 4

[17] Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. List update with locality of reference. In *Proc. 8th Latin American Theoretical Informatics Symp. (LATIN)*, volume 4957 of *Lecture Notes in Comput. Sci., Springer*, pages 399–410, 2008. 4, 17, 107

[18] Spyros Angelopoulos and Pascal Schweitzer. Paging and list update under bijective analysis. In *Proc. 20th Symp. on Discrete Algorithms (SODA)*, pages 1136–1145, 2009. 4, 17

[19] David Applegate, Luciana S, Bernard L. Dillard Buriol, David S. Johnson, and Peter W. Shor. The cutting-stock approach to bin packing: Theory and experiments. In *Proc. 5th Meeting on Algorithm Engineering and Experiments (ALENEX)*, 2003. 35

[20] Eyjólfur I. Ásgeirsson, Urtzi Ayesta, Edward G. Coffman, Jonathan Etra, Petar Momcilovic, David J. Phillips, Visda Vokhshoori, Zhenyu Wang, and Jon Wolfe. Closed on-line bin packing. *Acta Cybernet.*, 15(3):361–367, 2002. 10

[21] Ran Bachrach, Ran El-Yaniv, and M. Reinstadtler. On the competitive theory and practice of online list accessing algorithms. *Algorithmica*, 32(2):201–245, 2002. 77, 102

[22] János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. *Theoret. Comput. Sci.*, 440–441:1–13, 2012. 11, 41

[23] János Balogh, József Békési, Gábor Galambos, and Gerhard Reinelt. On-line bin packing with restricted repacking. *J. Comb. Optim.*, 27(1):115–131, 2014. 3

[24] Nikhil Bansal, José R. Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: Inapproximability results and approximation schemes. *Math. Oper. Res.*, 31(1):31–49, 2006. 12

[25] Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *Proc. 25th Symp. on Discrete Algorithms (SODA)*, pages 13–25, 2014. 12

[26] Kfir Barhum. Tight bounds for the advice complexity of the online minimum steiner tree problem. In *Proc. 40th International Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 77–88, 2014. 5

[27] Shai Ben-David, Allan Borodin, Richard M. Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994. 15

[28] Jon L. Bentley, David S. Johnson, Frank T. Leighton, Catherine C. McGeoch, and Lyle A. McGeoch. Some unexpected expected behavior results for bin packing. In *Proc. 16th Symp. on Theory of Computing (STOC)*, pages 279–288, 1984. 9, 11, 21

[29] Jon L. Bentley, Daniel Sleator, Robert E. Tarjan, and Victor K. Wei. A locally adaptive data compression scheme. *Commun. ACM*, 29:320–330, 1986. 16, 102

[30] Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovic, and Lucia Keller. Online coloring of bipartite graphs with and without advice. In *Proc. 18th Computing and Combinatorics Conf. (COCOON)*, volume 7434 of *Lecture Notes in Comput. Sci., Springer*, pages 519–530, 2012. 5

[31] Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovic, Sacha Krug, and Björn Steffen. On the advice complexity of the online $L(2,1)$-coloring problem on paths and cycles. In *Proc. 19th Computing and Combinatorics Conf. (COCOON)*, volume 7936 of *Lecture Notes in Comput. Sci., Springer*, pages 53–64, 2013. 5

[32] Jacek Błażewicz and Klaus Ecker. A linear time algorithm for restricted bin packing and scheduling problems. *Oper. Res. Lett.*, 2(2):80–83, 1983. 43

[33] David Blitz, Andre van Vliet, and Gerhard J. Woeginger. Lower bounds on the asymptotic worst-case ratio of online bin packing algorithms. Unpublished manuscript, 1996. 12

[34] Avrim Blum, Shuchi Chawla, and Adam Kalai. Static optimality and dynamic search-optimality in lists and trees. *Algorithmica*, 36(3):249–260, 2003. 4

[35] Hans-Joachim Böckenhauer, Juraj Hromkovic, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. In *Proc. 19th Computing and Combinatorics Conf. (COCOON)*, volume 7936 of *Lecture Notes in Comput. Sci., Springer*, pages 493–505, 2013. 5, 6

[36] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On the advice complexity of the k-server problem. In *Proc. 38th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 6755 of *Lecture Notes in Comput. Sci., Springer*, pages 207–218, 2011. 5, 40, 41

[37] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *Proc. 20th International Symp. on Algorithms and Computation (ISAAC)*, volume 5878 of *Lecture Notes in Comput. Sci., Springer*, pages 331–340, 2009. 5

[38] Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, and Peter Rossmanith. On the advice complexity of the knapsack problem. In *Proc. 10th Latin American Theoretical Informatics Symp. (LATIN)*, volume 7256 of *Lecture Notes in Comput. Sci., Springer*, pages 61–72, 2012. 5

[39] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998. 14, 15, 16, 69, 70, 77

[40] Joan Boyar and Lene M. Favrholdt. The relative worst order ratio for online algorithms. *ACM Trans. Algorithms*, 3(2), 2007. 4, 38

[41] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, DEC SRC, 1994. 16

[42] Canterbury Corpus. http://corpus.canterbury.ac.nz/. Accessed: 2014-17-09. 102, 108

[43] Ignacio Castiñeiras, Milan De Cauwer, and Barry O'Sullivan. Weibull-based benchmarks for bin packing. In *Proc. 18th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 207–222. Springer-Verlag, 2012. 35

[44] Brenton Chapin. Switching between two on-line list update algorithms for higher compression of burrows-wheeler transformed data. In *Proc. 10th Data Compression Conf. (DCC)*, pages 183–192, 2000. 102, 103

[45] Miroslav Chlebík and Janka Chlebíková. Inapproximability results for orthogonal rectangle packing problems with rotations. In *Proc.65th International Conf. on Algorithms and Complexity (CIAC)*, pages 199–210, 2006. 12

[46] Fan R. K. Chung, Dan J. Hajela, and Paul D. Seymour. Self-organizing sequential search and Hilbert's inequality. In *Proc. 17th Symp. on Theory of Computing (STOC)*, pages 217–223, 1985. 13

[47] Fan R. K. Chung, Dan J. Hajela, and Paul D. Seymour. Self-organizing sequential search and hilbert's inequalities. *J. Comput. Systems Sci.*, 36(2):148–157, 1988. 4

[48] Edward G. Coffman, Costas Courcoubetis, Michael R. Garey, David S. Johnson, Peter W. Shor, Richard R. Weber, and Mihalis Yannakakis. Bin packing with discrete item sizes, part I: perfect packing theorems and the average case behavior of optimal packings. *SIAM J. Discrete Math.*, 13(3):384–402, 2000. 34

[49] Edward G. Coffman, Michael R. Garey, and David S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation algorithms for NP-hard Problems*. PWS Publishing Co., 1997. 3, 10, 20, 88

[50] Edward G. Coffman, Micha Hofri, Kimming So, and Andrew Chi-Chi Yao. A stochastic model of bin packing. *Inform. and Control*, 44:105–115, 1980. 9, 11, 21

[51] Edward G. Coffman, David S. Johnson, Peter W. Shor, and Richard R. Weber. Bin packing with discrete item sizes, part II: tight bounds on First Fit. *Random Struct. Algorithms*, 10(1-2):69–101, 1997. 11, 21

[52] Edward G. Coffman and George S. Lueker. *Probabilistic analysis of Packing and Partitioning Algorithms*. John Wiley, New York, 1991. 9, 10

[53] Edward G. Coffman and Peter W. Shor. A simple proof of the $O(\sqrt{n \log^{3/4} n})$ up-right matching bound. *SIAM J. Discrete Math.*, 4:48–57, 1991. 24

[54] Edward G. Coffman Jr., János Csirik, Gabor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: survey and classification. In Panos M. Pardalos, Ding-Zhu Du, and Ronald L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 455–531. Springer, 2013. 10

[55] Janos Csirik and Gabor Galambos. An $O(n)$ bin-packing algorithm for uniformly distributed data. *Computing*, 36(4):313–319, 1986. 9, 10

[56] János Csirik, David S. Johnson, and Claire Kenyon. On the worst-case performance of the sum-of-squares algorithm for bin packing. *CoRR*, abs/cs/0509031, 2005. 35, 36

[57] János Csirik, David S. Johnson, Claire Kenyon, James B. Orlin, Peter W. Shor, and Richard R. Weber. On the sum-of-squares algorithm for bin packing. *J. ACM*, 53:1–65, 2006. 35, 36

[58] János Csirik, David S. Johnson, Claire Kenyon, Peter W. Shor, and Richard R. Weber. A self organizing bin packing heuristic. In *Proc. 1st Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 246–265, London, UK, UK, 1999. Springer-Verlag. 34, 35

[59] CyrusOne. Build vs. buy: Addressing capital constraints in the data center. *Executive Report*, 2013. 83

[60] 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. 43

[61] Zeger Degraeve and Marc Peeters. Optimal integer solutions to industrial cutting-stock problems: Part 2, benchmark results. *INFORMS J. on Computing*, 15(1):58–81, January 2003. 35

[62] Stefan Dobrev, Rastislav Královic, and Euripides Markou. Online graph exploration with advice. In *Proc. 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 7355 of *Lecture Notes in Comput. Sci., Springer*, pages 267–278, 2012. 5

[63] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO Inform. Theor. Appl.*, 43(3):585–613, 2009. 5

[64] Reza Dorrigiv, Martin R. Ehmsen, and Alejandro López-Ortiz. Parameterized analysis of paging and list update algorithms. In *Proc. 7th International Workshop in Approximation and Online Algorithms (WAOA)*, pages 104–115, 2009. 17

[65] Reza Dorrigiv, Alejandro López-Ortiz, and J. Ian Munro. An application of self-organizing data structures to compression. In *Proc. 8th Symp. on Experimental Algorithms (SEA)*, volume 5526 of *Lecture Notes in Comput. Sci., Springer*, pages 137–148, 2009. 17, 102, 103, 107

[66] Martin R. Ehmsen, Jens S. Kohrt, and Kim S. Larsen. List factoring and relative worst order analysis. *Algorithmica*, 66(2):287–309, 2013. 4, 13

[67] Ran El-Yaniv. There are infinitely many competitive-optimal online list accessing algorithms. Manuscript, 1996. 13, 14

[68] Aaron J. Elmore, Sudipto Das, Alexander Pucher, Divyakant Agrawal, Amr El Abbadi, and Xifeng Yan. Characterizing tenant behavior for placement and crisis mitigation in multitenant dbmss. In *Proc. International Conference on Management of Data (SIGMOD)*, pages 517–528, 2013. 83

[69] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoret. Comput. Sci.*, 412(24):2642 – 2656, 2011. 5

[70] Leah Epstein and Asaf Levin. Robust approximation schemes for cube packing. *SIAM J. Optim.*, 23(2):1310–1343, 2013. 12

[71] Leah Epstein and Rob van Stee. Optimal online bounded space multidimensional packing. In *Proc. 15th Symp. on Discrete Algorithms (SODA)*, pages 214–223, 2004. 12, 56

[72] Leah Epstein and Rob van Stee. Online square and cube packing. *Acta Inform.*, 41(9):595–606, 2005. 12

[73] Sally Floyd and Richard M. Karp. FFD bin packing for item sizes with uniform distributions on [0, 1/2]. *Algorithmica*, 6(1-6):222–240, 1991. 34

[74] Michal Forišek, Lucia Keller, and Monika Steinová. Advice complexity of online coloring for paths. In *Proc. 6th International Conf. on Language and Automata Theory and Applications (LATA)*, volume 7183 of *Lecture Notes in Comput. Sci., Springer*, pages 228–239, 2012. 5

[75] Gabor Galambos. A 1.6 lower bound for the two-dimensional online rectangle bin packing. *Acta Cybernet.*, 10:21–24, 1991. 12

[76] Gabor Galambos and Andre van Vliet. Lower bounds for 1-, 2-, and 3-dimensional online bin packing algorithms. *Computing*, 52:281–297, 1994. 12

[77] Gabor Galambos and Gerhard J. Woeginger. Repacking helps in bounded space online bin packing. *Computing*, 49:329–338, 1993. 3

[78] Michael R. Garey, Ronald L. Graham, and Jeffrey D. Ullman. Worst-case analysis of memory allocation algorithms. In *stoc72*, pages 143–150, 1972. 8

[79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the theory of of NP-Completeness*. Freeman and Company, San Francisco, 1979. 11

[80] Michel X. Goemans and Thomas Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proc. 25th Symp. on Discrete Algorithms (SODA)*, pages 830–839, 2014. 43

[81] Alexander Golynski and Alejandro López-Ortiz. Optimal strategies for the list update problem under the mrm alternative cost model. *Inform. Process. Lett.*, 112(6):218–222, 2012. 14

[82] Ansgar Grüne. MTF2 is not 2-competitive. Unpublished Manuscript, 2003. 77, 78

[83] Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: maintaining a constant-competitive steiner tree online. In *Proc. 45th Symp. on Theory of Computing (STOC)*, pages 525–534, 2013. 3

[84] Xiaodong Gu, Guoliang Chen, and Yinlong Xu. Deep performance analysis of refined harmonic bin packing algorithm. *J. Comput. Sci. Tech.*, 17:213–218, 2002. 10, 11, 20, 21

[85] Rohit Gupta, Sumit Kumar Bose, Srikanth Sundarrajan, Manogna Chebiyam, and Anirban Chakrabarti. A two stage heuristic algorithm for solving the server consolidation problem with item-item and bin-item incompatibility constraints. In *IEEE SCC'98*, pages 39–46, 2008. 84

[86] Torben Hagerup. Online and offline access to short lists. In *Proc. 32nd Symp. on Mathematical Foundations of Computer Science (MFCS)*, volume 4708 of *Lecture Notes in Comput. Sci., Springer*, pages 691–702, 2007. 14

[87] Xin Han, Francis Y. L. Chin, Hing-Fung Ting, Guochuan Zhang, and Yong Zhang. A new upper bound 2.5545 on 2d online bin packing. *ACM Trans. Algorithms*, 7(4):1–18, September 2011. 12

[88] Xin Han, Deshi Ye, and Yong Zhou. A note on online hypercube packing. *Cent. Eur. J. Oper. Res.*, 18(2):221–239, 2010. 12, 56

[89] James H. Hester and Daniel S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17:295–312, 1985. 17

[90] Juraj Hromkovič, Rastislav Královič, and Richard Královič. Information complexity of online problems. In *Proc. 35th Symp. on Mathematical Foundations of Computer Science (MFCS)*, volume 6281 of *Lecture Notes in Comput. Sci., Springer*, pages 24–36, 2010. 5

[91] Sandy Irani. Two results on the list update problem. *Inform. Process. Lett.*, 38:301–306, 1991. 13, 14, 15, 16

[92] Sandy Irani, Nick Reingold, Daniel Sleator, and Jeffery Westbrook. Randomized competitive algorithms for the list update problem. In *Proc. 2nd Symp. on Discrete Algorithms (SODA)*, pages 251–260, 1991. 14, 15

[93] David S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis, MIT, Cambridge, MA, 1973. 8, 10, 11

[94] David S. Johnson. Fast algorithms for bin packing. *J. Comput. Systems Sci.*, 8:272–314, 1974. 10

[95] David S. Johnson, Alan J. Demers, Jeffrey D. Ullman, Michael R. Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3:256–278, 1974. 4, 8, 21

[96] David S. Johnson and Michael R. Garey. A 71/60 theorem for bin packing. *J. Complexity*, 1(1):65–106, 1985. 11

[97] Shahin Kamali and Alejandro López-Ortiz. A survey of algorithms and models for list update. In *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of *Lecture Notes in Comput. Sci., Springer*, pages 251–266, 2013. 77

[98] Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc. 23rd Symp. on Foundations of Computer Science (FOCS)*, pages 312–320, 1982. 11

[99] Richard M Karp, Michael Luby, and A. Marchetti-Spaccamela. Probabilistic analysis of multi-dimensional binpacking problems. In *Proc. 16th Symp. on Theory of Computing (STOC)*, pages 289–298, 1984. 24

[100] Claire Kenyon. Best-fit bin-packing with random order. In *Proc. 7th Symp. on Discrete Algorithms (SODA)*, pages 359–364, 1996. 4

[101] Walter Knödel. A bin packing algorithm with complexity $o(n \log n)$ and performance 1 in the stochastic limit. In *Proc. 10th Symp. on Mathematical Foundations of Computer Science (MFCS)*, volume 118 of *Lecture Notes in Comput. Sci., Springer*, pages 369–378, 1981. 11

[102] Yoshiharu Kohayakawa, Flávio Keidi Miyazawa, Prabhakar Raghavan, and Yoshiko Wakabayashi. Multidimensional cube packing. *Elect. Notes on Discrete Math.*, 7:110–113, 2001. 12

[103] Dennis Komm, Richard Královic, and Tobias Mömke. On the advice complexity of the set cover problem. In *Proc. 7th International Computer Science Symp. in Russia (CSR)*, volume 7353 of *Lecture Notes in Comput. Sci., Springer*, pages 241–252, 2012. 5

[104] Dennis Komm and Richard Královič. Advice complexity and barely random algorithms. *RAIRO Inform. Theor. Appl.*, 45(2):249–267, 2011. 5

[105] Ralf Lämmel. Google's mapreduce programming model - revisited. *Sci. Comput. Program.*, 70(1):1–30, 2008. 55

[106] Chung-Chieh Lee and Der-Tsai Lee. A simple online bin packing algorithm. *J. ACM*, 32:562–572, 1985. 9, 10, 11, 20, 21, 31, 85

[107] Frank T. Leighton and Peter Shor. Tight bounds for minimax grid matching with applications to the average case analysis of algorithms. *Combinatorica*, 9:161–187, 1989. 9, 11, 21, 24

[108] Joseph Y. T. Leung, Tommy W. Tam, Chin S. Wong, Gilbert H. Young, and Francis Y. L. Chin. Packing squares into a square. *J. Parallel Distrib. Comput.*, 10(3):271–275, 1990. 12

[109] Yusen Li, Xueyan Tang, and Wentong Cai. On dynamic bin packing for resource allocation in the cloud. In *Proc. 26th Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 2–11, 2014. 83

[110] George S. Lueker. An average-case analysis of bin packing with uniformly distributed item sizes. Technical Report 181, University of California at Irvine Technical Report, 1982. 11

[111] Mark Manasse, Lyle A. McGeoch, and Daniel Sleator. Competitive algorithms for online problems. In *Proc. 20th Symp. on Theory of Computing (STOC)*, pages 322–333, 1988. 13

[112] Conrado Martínez and Salvador Roura. On the competitiveness of the move-to-front rule. *Theoret. Comput. Sci.*, 242(1-2):313–325, 2000. 103

[113] John McCabe. On serial files with relocatable records. *Oper. Res.*, 12:609–618, 1965. 12

[114] Rakesh Mohanty and N. S. Narayanaswamy. Online algorithms for self-organizing sequential search - a survey. *Elect. Coll. on Comput. Complexity.*, 16:97, 2009. 14

[115] J. Ian Munro. On the competitiveness of linear search. In *Proc. 8th European Symp. on Algorithms (ESA)*, volume 1879 of *Lecture Notes in Comput. Sci., Springer*, pages 338–345, 2000. 103

[116] Frank D. Murgolo. Anomalous behavior in bin packing algorithms. *Discrete Appl. Math.*, 21(3):229–243, 1988. 23, 38

[117] Prakash Ramanan and Kazuhiro Tsuga. Average-case analysis of the modified harmonic algorithm. *Algorithmica*, 4:519–533, 1989. 10, 11, 21

[118] Prakash V. Ramanan., Donna J Brown, Chung-Chieh Lee, and Der-Tsai Lee. On-line bin packing in linear time. *J. Algorithms*, 10:305–326, 1989. 10, 11, 21

[119] Nick Reingold and Jeffery Westbrook. Randomized algorithms for the list update problem. Technical Report YALEU/DCS/TR-804, Yale University, 1990. 15

[120] Nick Reingold and Jeffery Westbrook. Off-line algorithms for the list update problem. *Inform. Process. Lett.*, 60(2):75–80, 1996. 14, 16, 66, 69, 77, 104

[121] Nick Reingold, Jeffery Westbrook, and Daniel D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11:15–32, 1994. 14, 15, 16, 80

[122] Marc P. Renault and Adi Rosén. On online algorithms with advice for the k-server problem. In *Proc. 9th International Workshop in Approximation and Online Algorithms (WAOA)*, volume 7164 of *Lecture Notes in Comput. Sci., Springer*, pages 198–210, 2011. 5

[123] Marc P. Renault, Adi Rosén, and Rob van Stee. Online algorithms with advice for bin packing and scheduling problems. *CoRR*, abs/1311.7589, 2013. 54

[124] Wansoo T. Rhee and Michel Talagrand. Exact bounds for the stochastic upward matching problem. *Trans. AMS*, 307(1):109–125, 1988. 24

[125] Ronald Rivest. On self-organizing sequential search heuristics. *Commun. ACM*, 19:63–67, 1976. 4, 13

[126] Thomas Rothvoß. Approximating bin packing within $O(\log OPT. \log \log OPT)$ bins. In *Proc. 54th Symp. on Foundations of Computer Science (FOCS)*, pages 20–29, 2013. 11

[127] Jan Schaffner, Tim Januschowski, Megan Kercher, Tim Kraska, Hasso Plattner, Michael J. Franklin, and Dean Jacobs. RTP: robust tenant placement for elastic in-memory database clusters. In *Proc. International Conference on Management of Data (SIGMOD)*, pages 773–784, 2013. 83, 85, 90, 112

[128] Guntram Scheithauer and Johannes Terno. Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *Oper. Res. Lett.*, 20(2):93–100, 1997. 11

[129] Frank Schulz. Two new families of list update algorithms. In *Proc. 9th International Symp. on Algorithms and Computation (ISAAC)*, volume 1533 of *Lecture Notes in Comput. Sci., Springer*, pages 99–108, 1998. 13, 14

[130] Sebastian Seibert, Andreas Sprock, and Walter Unger. Advice complexity of the online coloring problem. In *Proc. 8th International Conf. on Algorithms and Complexity (CIAC)*, volume 7878 of *Lecture Notes in Comput. Sci., Springer*, pages 345–357, 2013. 5

[131] Steven S. Seiden. On the online bin packing problem. *J. ACM*, 49:640–671, 2002. 10, 11, 21

[132] Peter W. Shor. The average-case analysis of some online algorithms for bin packing. *Combinatorica*, 6:179–200, 1986. 9, 10, 11, 21, 24

[133] Peter W. Shor. How to pack better than Best-Fit: Tight bounds for average-case on-line bin packing. In *Proc. 32nd Symp. on Foundations of Computer Science (FOCS)*, pages 752–759, 1991. 10

[134] Daniel Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28:202–208, 1985. 4, 12, 13, 14, 17, 103

[135] Benjamin Speitkamp and Martin Bichler. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE T. Services Comput.*, 3(4):266–278, 2010. 84, 101

[136] Wolfgang M. Stille. *Solution Techniques for specific Bin Packing Problems with Applications to Assembly Line Optimization*. PhD thesis, TU Darmstadt, 2008. 34

[137] Chandrasekar Subramanian, Arunchandar Vasan, and Anand Sivasubramaniam. Reducing data center power with server consolidation: Approximation and evaluation. In *HiPC '10*, pages 1–10, 2010. 84

[138] Boris Teia. A lower bound for randomized list update algorithms. *Inform. Process. Lett.*, 47:5–9, 1993. 16

[139] Jeffrey D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University Technical Report, 1971. 8

[140] Andre van Vliet. *Lower and upper bounds for online bin packing and scheduling heuristics*. PhD thesis, Erasmus University, Rotterdam, The Netherlands, 1995. 12

[141] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2004. 43

[142] Andrew C. C. Yao. New algorithms for bin packing. *J. ACM*, 27:207–227, 1980. 8, 10, 11, 21