ON DISCRETE HYPERBOX PACKING

A Dissertation

by

XIAFENG LI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2008

Major Subject: Computer Science

ON DISCRETE HYPERBOX PACKING

A Dissertation

by

XIAFENG LI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Donald Friesen
Committee Members,	Jianer Chen
	Andreas Klappenecker
	Catherine Yan
Head of Department,	Valerie E. Taylor

December 2008

Major Subject: Computer Science

ABSTRACT

On Discrete Hyperbox Packing. (December 2008)

Xiafeng Li, B.S., Jingdezhen Ceramic Institute;

M.S., Shanghai Jiaotong University

Chair of Advisory Committee: Dr. Donald Friesen

Bin packing is a very important and popular research area in the computer science field. Past work showed many good and real-world packing algorithms. However, due to the complexity of the problem in multiple-dimensional bin packing, also called hyperbox packing, we need more practical packing algorithms for its real-world applications.

In this dissertation, we extend 1D packing algorithms to hyperbox packing problems via a general framework that takes two inputs of a 1D packing algorithm and an instance of hyperbox packing problem and outputs a hyperbox packing algorithm. The extension framework significantly enriches the family of hyperbox-packing algorithms, generates many framework-based algorithms, and simultaneously calls for the analysis for those algorithms.

We also analyze the performance of a couple of framework-based algorithms from two perspectives of worst-case performance and average-case performance. In worst-case analysis, we use the worst-case performance ratio as our metric and analyze the relationship of the ratio of framework-based algorithms and that of the corresponding 1D algorithms. We also compare their worst-case performance against two baselines: strip optimal algorithms and optimal algorithms. In average-case analysis, we use expected waste as a metric, analyze the waste of optimal hyperbox packing algorithms, and estimate the asymptotic forms of the waste for framework-based algorithms.

To my family

ACKNOWLEDGMENTS

In the long journey towards my Ph.D degree, I received a lot of help from a variety of people. I am always grateful for their help and would like to take this opportunity to acknowledge their kindness and express my appreciation.

First of all, I'd like to thank Dr. Donald Friesen, the incomparable research advisor, for his patient guidance and wise advisory on my research. He introduced me to the bin packing problem, referenced me the papers related to the research topic, inspired me to explore complicated problems from different perspectives, even worked with me to derive formulas, and patiently examined the correctness of proofs and the results. My appreciation definitely goes to him for all he did for me. I was incredibly lucky to have him as my advisor.

Second of all, I'd like to thank my other committee members, including Drs. Jianer Chen, Andreas Klappenecker, and Catherine Yan. They took their precious time to review my research manuscripts, to give me preliminary exams, and gave me important comments and suggestions. I very much appreciate their time and effort in helping me improve the quality of my research proposal and dissertation.

Next, I'd like to thank my wife, Dongling Zhan, for giving me the firm support of pursuing the degree. She, along with my little son, Allen Li, gave me a joyful and stress-free life, which gave me great encouragement and relieved the pressure from studying. I also want to thank my parents and my parents-in-law for helping me to take my responsibilities at family. Without their support and encouragement, it would have been impossible for me to get this dissertation done.

Last, but not least, I'd like to thank all my friends that gave me help and care during my long journey to get the degree.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

Bin packing, that is, packing a list of items into minimal number of bins, is one of the most popular theoretical research areas in the computer science field. Many computer scientists and mathematicians devoted more than thirty years to studying this problem and have published thousands of papers. Each year, a variety of packing problem arise and desperately need solutions. Meanwhile, novel techniques are proposed [2], [14], [17], [18], [20], [51], [59],[62], [70],[73], [77] to solve existing open packing problems. Bin packing, despite having been researched for decades, has never lost its attraction for its followers.

Bin packing problems are popular because they are broadly applicable [3], [12], [25], [41], [43], [44], [45], [57], [67], [74], [78], [80], [81]. Even though the problems are theoretical, real-world problems that are abstracted, such as scheduling [9], [10], [11], [16], [40], [50], [68], [65], are prevalent in everyday life. Real-world applications may include shopping, scheduling appointments, making clothes, packing luggage, and manufacturing furniture. Because of the wide applications, new understanding of this problem can extensively change our life style.

Bin packing problems are also popular because they are challenging. Even the simplest bin packing problem is NP-hard, which means no polynomial-time algorithms can solve bin-packing problem unless P = NP. Since optimal solutions cannot practically be obtained, researchers mainly tackle this problem through approximation techniques. One of the approximation techniques is to find asymptotic polynomial time approximation scheme (APTAS), and many APTASs were found in 1D packing.

The journal model is *IEEE/ACM Transactions on Networking*.

In some variants of bin packing problem like multiple-dimensional packing, there even do not exist APTAS [8], [7]. So far, many approximation algorithms have been proposed, and they shed new light on understanding the bin packing problem. However, the lack of standard criteria of evaluating those algorithms makes things even more complicated. For instance, some people prefer using worst-case analysis to evaluate the algorithms, while others like average-case analysis measurement.

A. Bin Packing Problem and Its Variants

The study of the original bin packing problem dates back to the early 1970's. In this problem, bins have identical one dimensional (1D) size B, and items have various 1D size a_i, where $a_i \leq B$. For each occupied bin, items inside can not be overlapped, and the sum of the length of the items should be less than or equal to bin's size B. The goal of the problem is to find a packing configuration so that the number of used bins is minimal. Figure 1 illustrates an example of the original bin packing problem. In the example, big rectangles represent identical bins, small rectangles represent items of various sizes, which are indicated by the integer numbers in the figure. The capacity of bins are set as $B = 5$, 5 items of sizes 1, 2, 2, 3, and 4 are already packed into 3 bins, and 1 item of size 1 is being packed into the third bin.

The original bin packing problem could be classified as either **on-line** or **off-line** depending on the occurrence of to-be-packed items. If each arriving item has to be packed immediately before the next one comes, it is called on-line packing. On the other hand, off-line packing has knowledge of all the to-be-packed items at the beginning.

Nowadays, original packing problem has been widely extended [4], [5], [13], [15], [19], [22], [26], [36], [42], [46], [60], [63], [66], [71], [76], [75]. Changing items' size

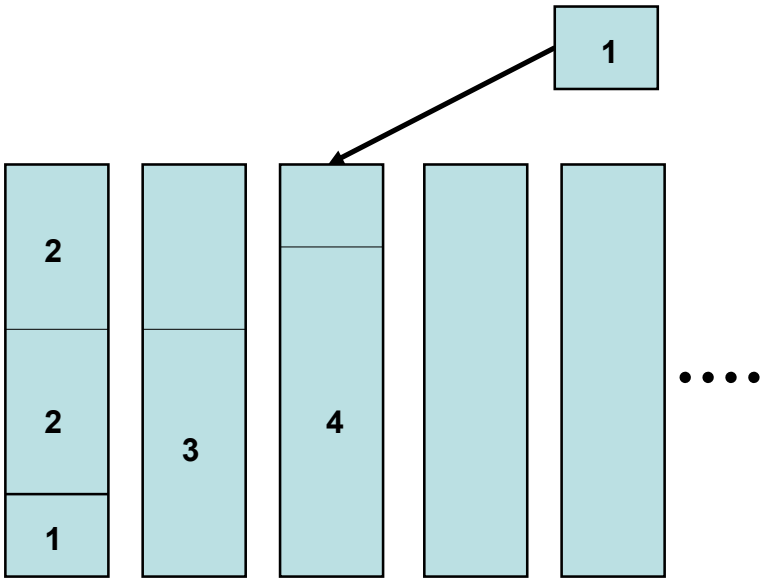

Fig. 1. An example of original bin packing.

from deterministic to random yields stochastic packing. Such packing is usually on-line, and items' sizes are drawn from certain random distributions like uniform distribution. Stochastic packing is often used to evaluate the expected performance of packing algorithms. Restricting items' size to be discrete instead of continuous gives discrete packing, which is more suitable to model certain real life applications. Even though their apparent difference looks small, discrete packing and continuous packing are essentially distinct [27], and likely need quite different approaches for algorithm analyses. Studying discrete packing not only aids to solve the corresponding applications, but also produces more understanding on continuous packing. Changing the sizes from 1 dimension to multi-dimensional hyperboxes, we get hyperbox packing [47], [54], [69], [61]. To model real-world applications like metal cutting and truck loading, hyperbox packing is necessary but creates much more difficulty than one dimensional packing. Besides the above extension, other variant packings include

variable-size packing, which allows bins to have various size; packing with rotation that allows items in multi-dimensional packing to be rotated, and some specified constraint-packings that put more constraints on the packing process. For example, some extensions restrict the maximum number of items in some bins. Others may restrict that particular items can not be packed in the same bin.

The above extended packing problems can be combined properly and form a new packing problem. For example, multi-dimensional packing, discrete packing, and online packing may join together and become an online multi-dimensional discrete packing; in which items are multi-dimensional hyperboxes with discrete edge length.

B. Real-World Applications

As stated before, real-world applications make bin packing problem popular in the computer science area. In fact, existence of those applications is one of the major reasons to study bin packing. To better understand the importance of researching this problem, we next review those applications and explain how they relate to bin packing.

1. 1D Applications

It's often required for us to backup electronic files from computers to compact disks (CD). Usually, the CDs have fixed size (700MB), while the electronic files are of variable sizes. Figure 2 shows an example of copying computer files to CDs. A natural goal in the example is to minimize the number of used CDs. Achieving the goal needs a proper arrangement of recorded files, which is the same as arranging items in bin packing. Comparison of Figure 1 with Figure 2 shows other similarities between 1D bin packing and recording CDs; where electronic files are like items, CDs

are like bins, CDs could not hold files more than their capacity (700MB), and each file should not be split into two CDs.

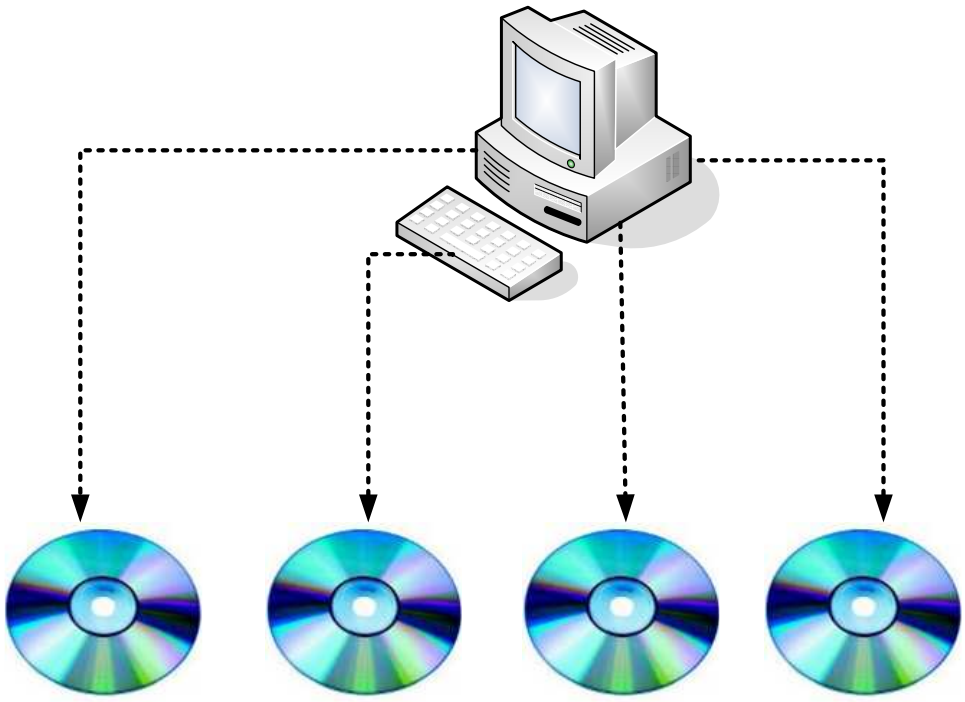

Fig. 2. Copying electronic files from computers to CDs.

The above CD recording example shows us how bin packing models real-life applications, but it is only one instance from thousands of practical applications. Similar examples include copying data to other removable media like floppy disks, and storing items into bags that only have weight limitations. All those examples could be categorized as storing-item field. Real-world applications also happen in scheduling and stock-cutting. For example, scheduling television commercials is a one-dimensional bin packing problem, in which commercial break-time is compared to bins and available advertisements equal to items.

2. 2D Applications

Two dimensional bin packing could also be applied in three fields: item storing, stock-cutting, and scheduling. Next, we illustrate a stock-cutting example: glass cutting.

As we know, windows protect houses from wind, bring light from outside, and make our lives more comfortable. The major element of a window is glass, and you might be surprised that the process of obtaining windows' glass is related to bin packing. Usually, glass factories produce many big sheets of glass called raw glass, and windows' glass is cut from each sheet according to windows' size. Suppose that n sheets of windows' glass are needed to be cut, and that all windows are rectangles but may have different sizes. To minimize the waste of glass, we usually prefer a cutting scheme that uses minimum sheets of raw glass but produces all required windows' glass. The following comparison shows that this problem can be reduced to 2D bin packing, in other words, solving 2D bin packing gives solutions to glass cutting problem. Let's consider sheets of raw glass as 2D bins, and windows' glass as 2D items. Note that the solution of 2D bin packing shows how to pack 2D items into minimum number of 2D bins. Therefore, it indicates how to assemble windows' glass into minimum sheets of glass that are smaller than raw glass. Reversing the assembling process yields the desired cutting process, which minimizes the cutting waste.

Besides glass cutting, other 2D stock cutting problems, like cutting cloth for clothes, cutting metal for widgets, can be generalized by 2D bin packing. Actually, manufacturing factories always need to solve similar problems, such as how to efficiently lay out chips on VLSI board. In the fields of both scheduling and storing items, 2D bin packing is also widely applied.

3. 3D Applications

Nowadays, global trade exchanges products between countries, and lowers our living expense. Container loading plays an important role in global trade, especially in the process of cargo transportation. In general, boxes of products are first packed into containers, then the latter are placed on cargo ships for transportation. Given thousands of product boxes, the packing that minimizes the number of containers could save shipping cost, therefore, is preferred in practice. Container loading could be obviously modeled by 3D bin packing; in which 3D items correspond to product boxes and 3D bins comparable to containers. Also note that container loading and 3D bin packing share similarities in many other aspects, which include: product boxes can not be split into two containers; boxes can not be overlapped once packed; the sum volume of being packed boxes in each container can not exceed the volume of the container; and the number of used containers is minimum.

Other loading problems, like truck loading and train loading, are essentially the same as container loading, and could be modeled by 3D bin packing.

C. Our Work

As shown in previous bin packing examples, many real-world applications like truck-loading need to be modeled as hyperbox packing, in which items and bins are hyper-boxes. Past research has focused more on 1D bin packing than on hyperbox packing, and most results in this area are still limited to 1D packing. The goal of our work is to extend many of those 1D results to hyperbox packing.

Our research work consists of three parts: extension framework and framework-based algorithms, the worst-case analyses of framework-based algorithms, and the average-case analyses of framework-based and optimal algorithms.

To widely extend approximation algorithms from 1D to multiple dimensions, we aim at inventing a general approach to extend all existing 1D algorithms to hyperbox packing. The first part of our work is the extension framework, which serves this exact purpose. For any 1D discrete packing algorithm, our framework provides a simple and general method that transfers it into a hyperbox-packing algorithm.

Applying existing 1D algorithms into the framework, we get many framework based algorithms. It is natural to analyze the performance of those algorithms, and compare their performance with that of their 1D counterparts. The second part of the dissertation is called the worst-case analysis of framework-based algorithms. In this part, we intensively focus on some classical framework-based algorithms, and estimate their asymptotic worst-case ratios. The ratios shed light on the worst-case performance of the framework-based algorithms, and also indicate how good our framework is.

The third part is average-case analyses of packing algorithms through the measure expected waste. We analyze the expected waste on optimal algorithm, and reveal that optimal expected waste, EW_n^{OPT}, of discrete hyperbox packing can only take three asymptotic forms $\Theta(1)$, $\Theta(\sqrt{n})$, and $\Theta(n)$. Which form it takes depends on the items' size distribution and bins' size. In this part, we also do average-case analyses on framework-based algorithms. We apply popular 1D packing algorithms into the framework, estimate their asymptotic expected wastes, and compare them with those of 1D counterparts.

CHAPTER II

BACKGROUND

Recall that bin packing problem is NP hard, and that there is very likely no polynomial time algorithmic solution. Computer scientists observed that the hardness of the problem is mainly due to the strict goal, to use minimal number of bins. The problem will become much easier if the goal is relaxed. For example, in CD recording problem, it is difficult to find a solution that uses *minimal* number of CDs to record given files, but is easier if the requirement for the minimal number is relaxed. In real-world bin packing applications, the goal is to use the number of bins as small as possible, and it is not critical to use the exact minimal number of bins.

Approximation algorithms were invented based on the above observation. They are usually simple, easy-to-understand, of polynomial running-time, and have acceptable performance. Next, we review some classical approximation algorithms for one dimensional bin packing.

A. One-Dimensional Approximation Algorithms

1. Next Fit (NF)

Next Fit [52], [30] algorithm is one of the simplest packing algorithms, and works as follows. In the whole packing process, only one bin is opened at any time. For each to-be-packed item, the algorithm checks if the item could be fit in the opened bin or not. If the opened bin has space for the item, Next Fit will pack the item into the bin. Otherwise, the algorithm closes the bin, opens a new bin, and packs the item into the newly opened bin.

Figure 3 shows an example on how Next Fit works against a bin packing instance,

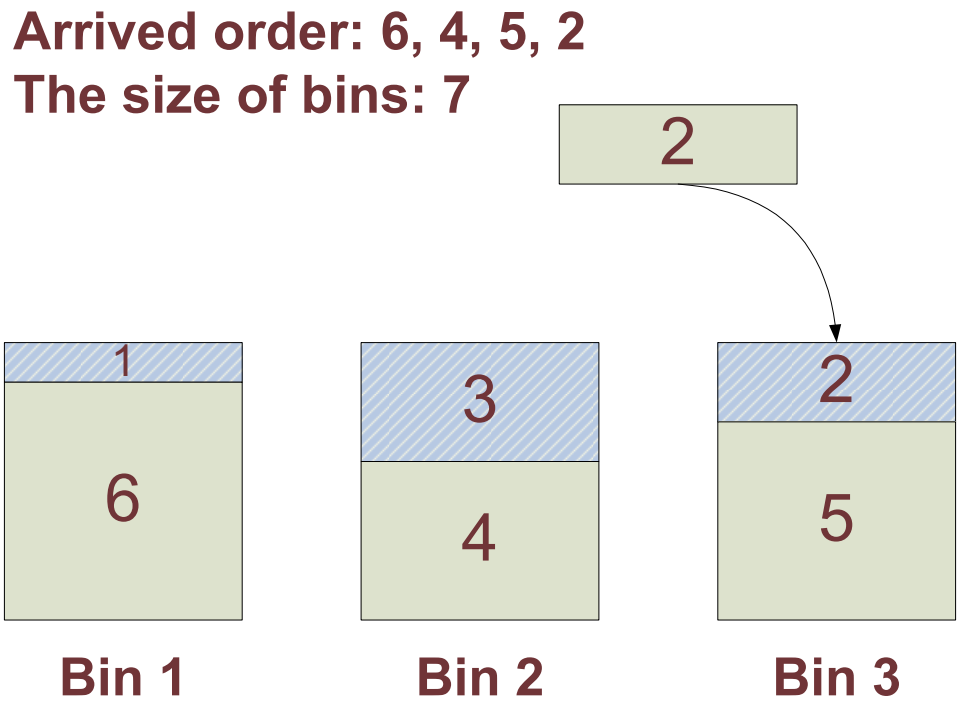

Fig. 3. An example of Next Fit.

where the size of bin is 7, three items of sizes 6, 4, and 5 are already packed into existing bins. When next item of size 2 comes, only bin 3 is opened, while bins 1 and 2 are closed. Therefore, Next Fit will only check bin 3, finds that the bin has enough space for the item, and will pack the item into the bin.

As shown in the above example, Next Fit is a simple and easy-to-be-implemented algorithm. The algorithm has space-complexity $O(1)$ since it only keeps one bin open. For each arriving item, it only compares it with the opened bin, therefore, has running time $O(n)$, where n is the total number of to-be-packed items. However, its performance is not as good as other classical approximation algorithms.

2. First Fit (FF)

Unlike Next Fit, First Fit [52], [30] leaves all partially filled bins rather than the last one open. For each arriving item a_i, it sequentially checks all open bins, and picks the first one (lowest index) that has enough space for the item. If no such open bin exists, which indicates none of the open bins has enough space for the arriving item, First Fit opens a new bin and packs the item into the newly opened bin.

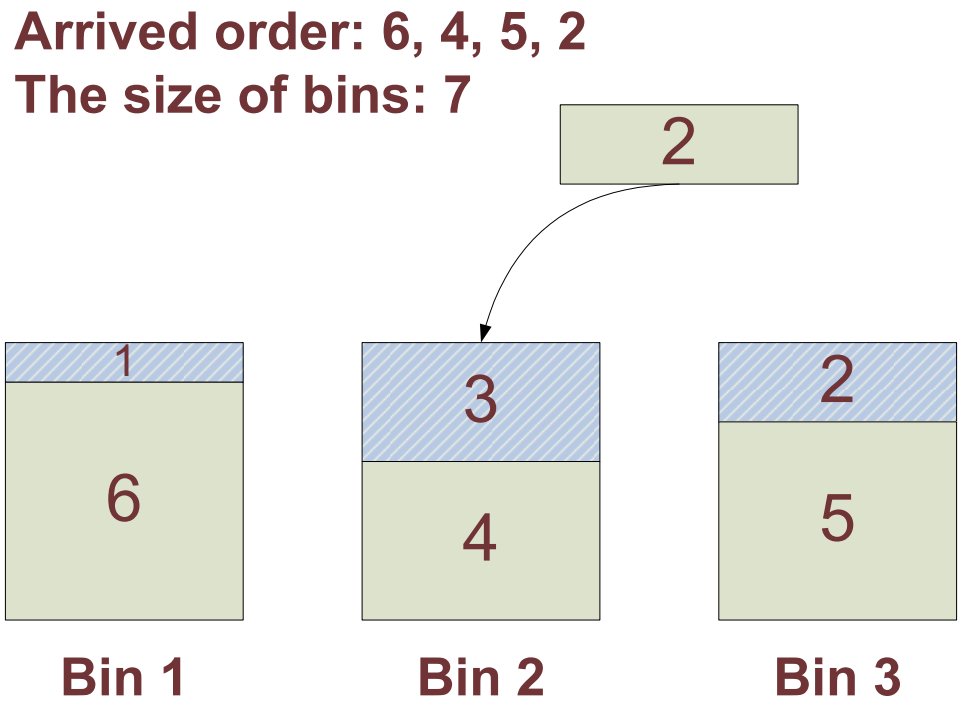

Fig. 4. An example of First Fit.

Figure 4 shows a similar packing instance for First Fit, which could be compared with Figure 3. In the example, items of sizes 6, 4, 5 were sequentially packed in bins 1, 2, and 3. When item of size 2 comes, First Fit will consider bins 1, 2, or 3 in order, and check if the item could fit any of them. Note that bin 1 does not have enough space, but bin 2 does have. Therefore, First Fit will use bin 2, which is the

first eligible bin, to pack the item, and does not consider bin 3.

First Fit is a straight-forward algorithm, and is also easy to-be-implemented. It marks all partially-filled bins open, therefore, has space-complexity $O(n)$, where n is the number of to-be-packed items. In the case where each arriving item does not fit into any existing open bins, First Fit has to compare each item with all open bins, and open a new bin for the item. By using a binary tree data structure, the running time of First Fit will be $O(n \log(n))$.

3. Best Fit (BF)

Note that First Fit tries to pack each item a_i into the first (lowest indexed) bin that the item can fit into. Intuitively, there may exist better packing rules than favoring the lowest index. Best Fit, favoring the partially filled bin that has the least, but enough, space for item a_i (ties broken by index), was proposed under this motivation.

Like First Fit, Best Fit [52], [30] keeps all partially-filled bins open and tries to pack each item into the open bin that has enough, but least, space for the item. If several bins have enough and least space, Best Fit makes ties broken by index. If no opened bin has enough space for the item, the algorithm will open a new bin, and put the item into the newly opened bin.

Figure 5 shows an example of how Best Fit works, where three items of size 6, 4, and 5 were already packed into three bins of size 7. When next item of size 2 comes, Best Fit keeps partially-filled bins 1, 2, and 3 open, and attempts to pack the coming item into one of the three opened bins. Note that bin 3 has least and enough space 2 for the coming item. Therefore, Best Fit will use bin 3 to pack the coming item.

Like First Fit, Best Fit needs to keep all partially filled bins open, and needs $O(n)$ space, where n is the number of to-be-packed items. If all partially-filled bins are maintained in a binary heap data structure, it will take $O(\log n)$ time to maintain

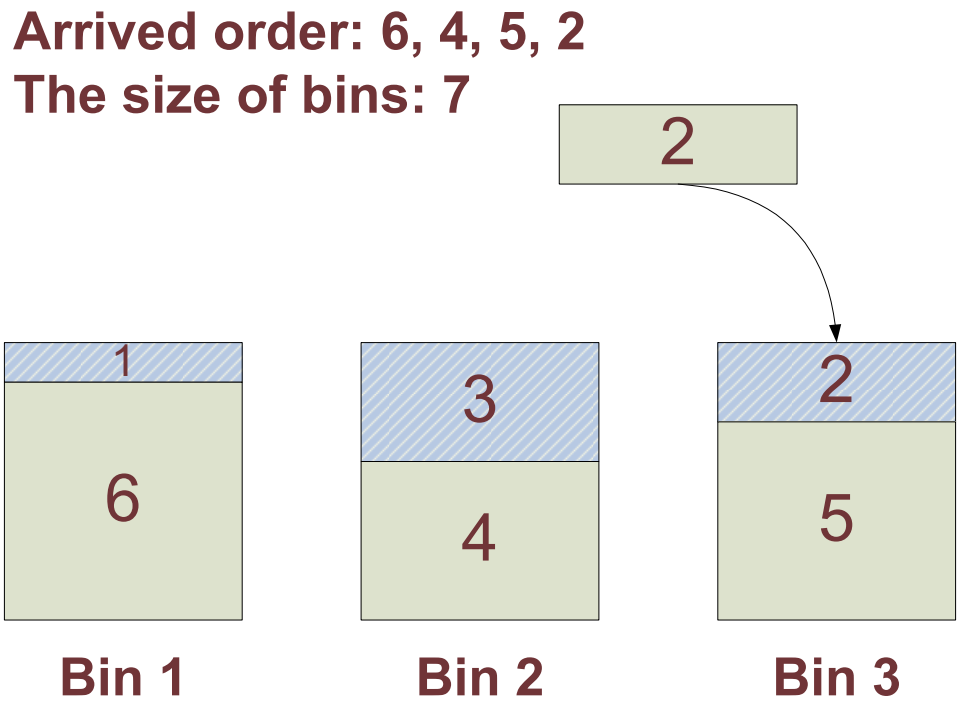

Fig. 5. An example of Best Fit.

the data structure and to find the open bin (or decide to create a new bin) for each arriving item. Therefore, Best Fit at most takes $O(n \log n)$ time to pack n items.

4. Sum of Squares (SS)

Note that First Fit and Best Fit are classical bin packing algorithms, which have been widely-studied since the beginning of bin packing problems. It turns out that the algorithms have acceptable worst-case performance, but their average-case performance is not good enough, and needs to be improved. In the past, researchers tried to find algorithms to beat FF and BF in average-case performance. Sum of Squares (SS) is one such invented algorithm that achieves the objective.

Algorithm Sum of Squares [37], [38] is a little more complicated than First Fit

and Best Fit, and we need the following concepts and notations to better understand the algorithm. It defines the level of a bin as the total size of all filled items in the bin. For example, Figure 6 shows two bins of size 7, where bin 1 has two items of sizes 2 and 4, bin 2 has three items of sizes 1, 2 and 2. In the example, the level of bin 1 is total size of items of sizes 2 and 4, which is $l = 2 + 4 = 6$. Similarly, the level of bin 2 is $l = 1 + 2 + 2 = 5$. The algorithms also uses notation $N(l)$ to denote the number of all partially filled bins with level l. In Figure 6, bin 1 has level 6, and bin 2 has level 5, therefore, $N(1) = N(2) = N(3) = N(4) = 0, N(5) = N(6) = 1$.

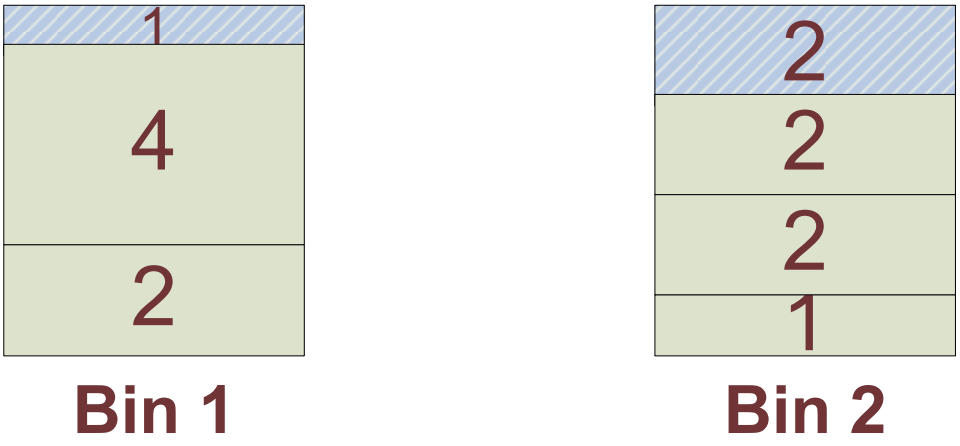

Fig. 6. The level of bins.

The algorithm uses utility function $\sum_{l=1}^{B-1} N^2(l)$, and tries to minimize the value of the function when packing each arriving item. Figure 7 shows an example how the Sum of Squares algorithm packs an item. In the example, items of sizes 6, 4, and 5 are packed into three bins of size 7. When item of size 2 comes, Sum of Squares could possibly pack it into bin 2 or bin 3. If item is packed into bin 2, the value of the utility function will be $\sum l = 1^6 N^2(l) = 2^2 + 1 = 5$, where $N(1) = N(2) = N(3) = N(4) = 0$, $N(6) = 2$, and $N(5) = 1$. If item is packed into bin 3, the value of the utility function

will be $\sum_{l=1}^{6} N^2(l) = 1 + 1 = 2$, where $N(1) = N(2) = N(3) = N(5) = 0$, and $N(6) = N(4) = 1$. Since Sum of squares tries to minimize the value of the utility function, it will use bin 3 to pack to the coming item of size 2.

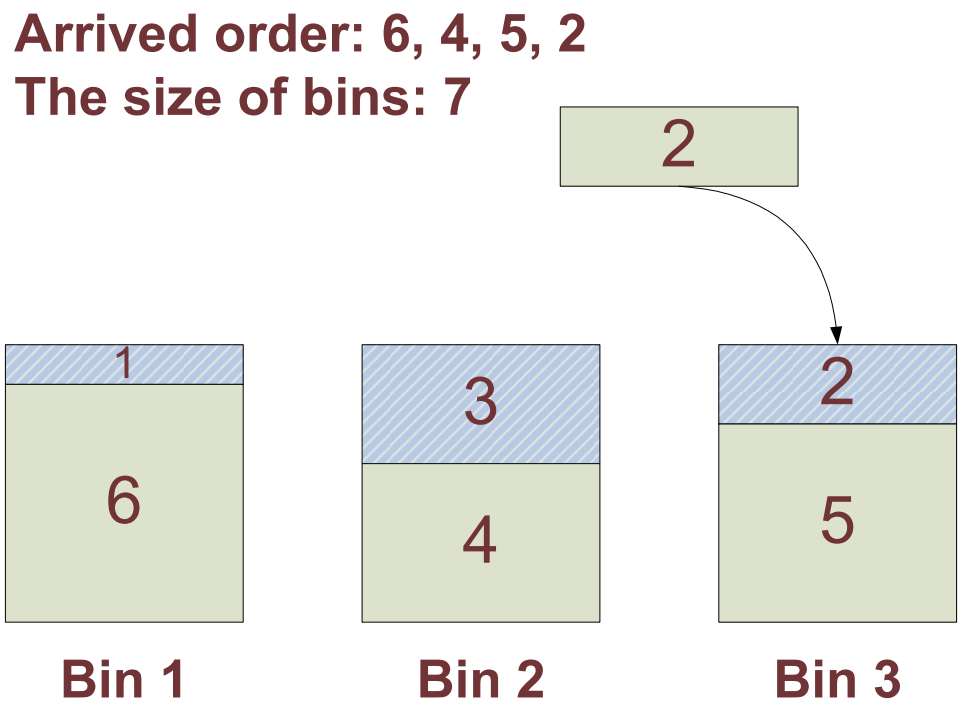

Fig. 7. An example of Sum of Squares.

Since algorithm Sum of Squares also keeps all partially filled bins open, its space complexity is $O(n)$. The analysis of its time complexity is a little tricky. Suppose that the bins' size is B, all available items' sizes are $1, \cdots, B$. Recall that the algorithm's utility function, $\sum_{l=1}^{B-1} N^2(l)$, only depends on bins' level l and their count $N(l)$. For each to-be-packed item of size a, the algorithm essentially only needs to find the best level rather than the best specific bin to pack the item. It may go over all feasible levels from 1 to $B - a$, compute the new values of utility function if the item of size a is packed into the level, and then pick the level that has lowest value. Obviously,

the process will take at most time $O(nB)$.

5. First Fit Decreasing (FFD)

First Fit Decreasing (FFD) algorithm is an offline bin-packing algorithm that has pre-knowledge of all to-be-packed items' sizes and order even before the packing process. FFD takes advantage of the pre-knowledge, and sorts all to-be-packed items in decreasing order based on their sizes. After all items are sorted, the algorithm uses First Fit to packed the ordered items.

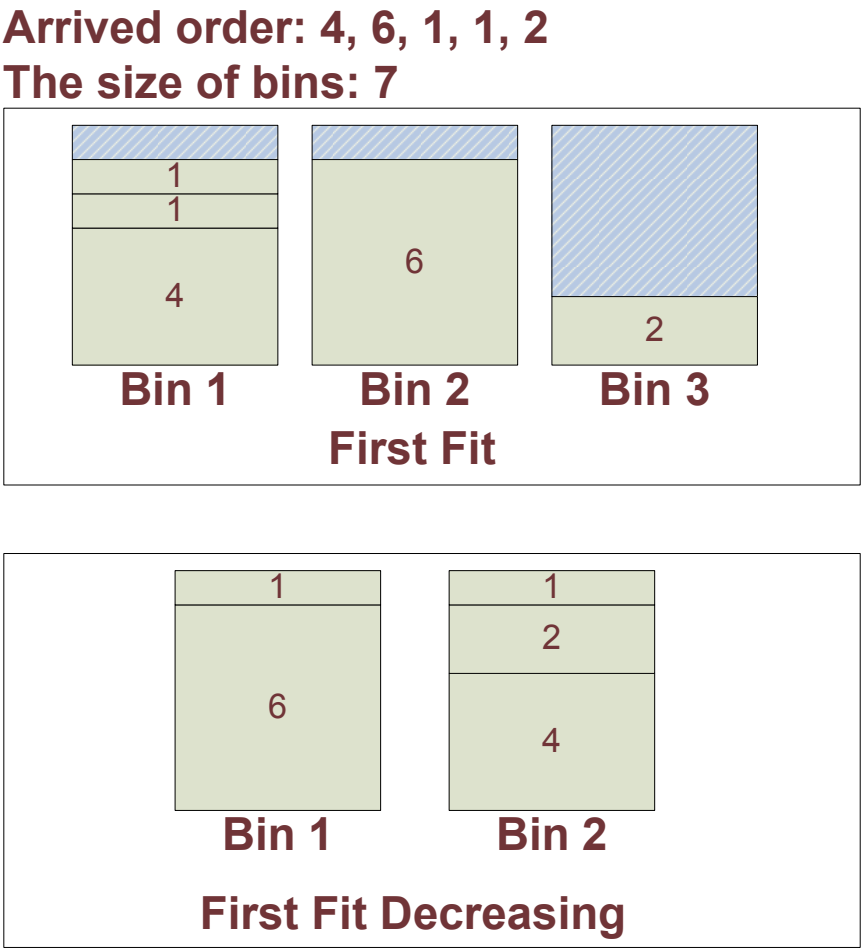

Fig. 8. Comparison of First Fit (Top) with First Fit Decreasing (Bottom).

Figure 8 shows a packing example for FFD and FF. In the example, bins are of size 7, and items of size 4, 6, 1, 1, 2 arrive in order. First Fit packs the items into three bins as shown on the top of Figure 8. Regarding the FFD, all items are first sorted by their sizes in decreasing order, which will make items arriving order as 6, 4, 2, 1, 1. After the sorting process, FFD uses First Fit to pack the sorted items, which yields the packing configuration on the bottom of Figure 8. As we see, the sorting process makes FFD use one less bin than FF.

To pack n items, the space complexity of FF and sorting process are both $O(n)$, which gives the space complexity of FFD as $O(n)$. Note that the sorting process in FFD can take $O(n \log n)$ time, and FF algorithm can take $O(n \log n)$ time. Therefore, the time complexity of FFD is $O(n \log n)$.

6. Best Fit Decreasing (BFD)

Just like FFD, Best Fit Decreasing(BFD) also has pre-knowledge of to-be-packed items, first sorts items by their sizes in decreasing order, and then uses BF to pack the sorted items. Figure 9 compares how BFD and BF pack an item sequence, where bins' size is 7, and items' sequence is: 4, 6, 1, 1, 2. In the example, both BF and BFD use two bins to pack the 5 items. Also note that BF uses $O(n)$ space and $O(n \log n)$ time, and sorting process does not increase either space complexity or time complexity. Therefore, the algorithm takes $O(n \log n)$ time and $O(n)$ space as well.

B. Worst-Case Analysis

Performance ratio is one of the most popular measures to evaluate approximation algorithms' performance. Denote by L the input list of items, by B the bin's capacity,

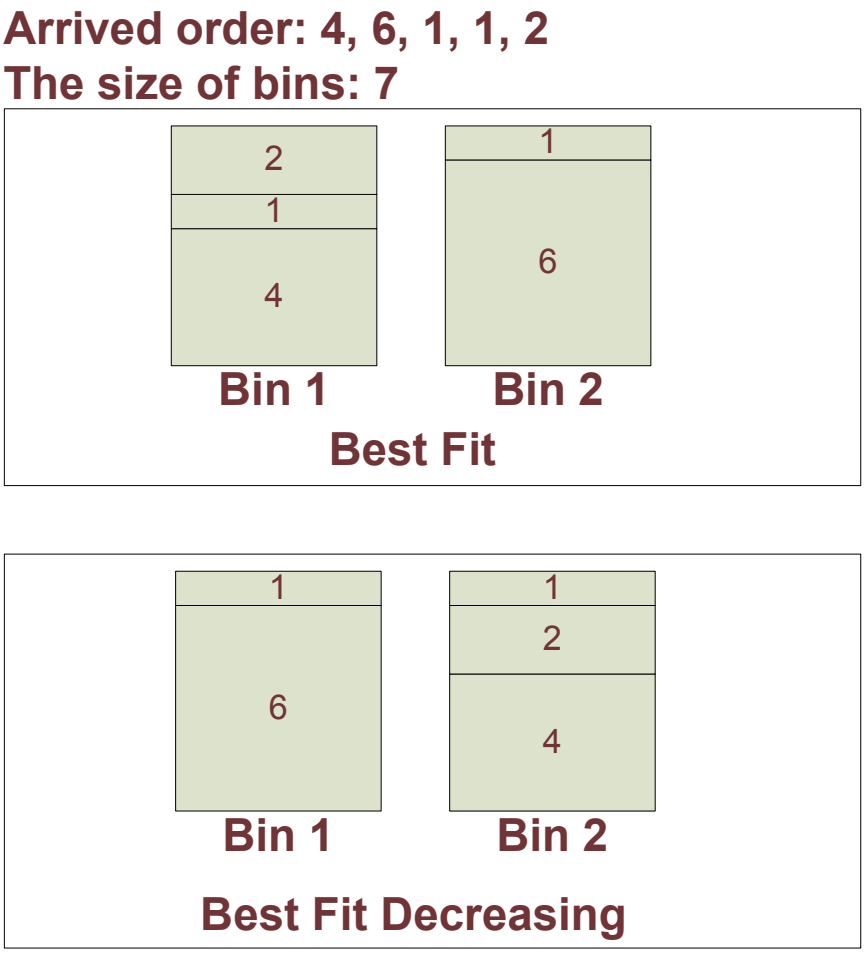

Fig. 9. Comparison of Best Fit with Best Fit Decreasing.

by $A(L)$ the number of bins used by an approximation algorithm A, and by $OPT(L)$ the number of bins used by optimal algorithms. The **performance ratio** of A w.r.t. L and B is defined as

$$R_A(L,B) = \frac{A(L,B)}{OPT(L,B)}. \tag{2.1}$$

Measure **worst-case performance ratio** $R_A(B)$ is the supremum of $R_A(L, B)$ for all L. Precisely,

$$R_A(B) = \sup_{L \in \mathcal{L}} R_A(L, B), \tag{2.2}$$

where $\mathcal{L}$ is the set of all packing instance. The **asymptotic worst-case performance ratio** $R_A^\infty(B)$ is defined as

$$R_A^\infty(B) = \limsup_{OPT(L) \to \infty} R_A(L, B). \tag{2.3}$$

Note that the above asymptotic worst-case ratio depends on bins' size B. In some cases, we use $R_A^\infty = \sup_{B \in \mathcal{R}} R_A^\infty(B)$ to denote another form of asymptotic worst-case ratio, which is independent of B. In this dissertation, we refer to both forms as **asymptotic worst-case ratio**.

Note that it is very hard to get the exact value of worst-case performance ratio, and past work usually tackled the problem from two perspectives: lower bound and upper bound. Next, we review the two bounds of ratios for algorithms NF, BF, FF, SS, FFD, and BFD.

1. Next Fit

Regarding the lower bound of R_{NF}^∞, we may use the packing example as shown in Figure 10, where bins' size B is an even number, NB size-$B/2$ items and NB size-1 items come alternatively. In other words, items sequence is like: $B/2, 1, B/2, 1, \cdots$. In this case, NF packs the items into NB bins as shown on the top of Figure 10, while optimal algorithm packs the items into $NB/2 + N$ bins as on the bottom of the Figure 10. Letting N go to infinity, we get the lower bound of R_{NF}^∞ as:

$$R_{NF(B)}^\infty \geq \lim_{N \to \infty} \frac{NB}{NB/2 + N} = \frac{2}{1 + 1/B}, \tag{2.4}$$

which gives

$$R_{NF}^{\infty} \geq \sup_{B \in \mathcal{N}} R_{NF}^{\infty}(B) = 2. \tag{2.5}$$

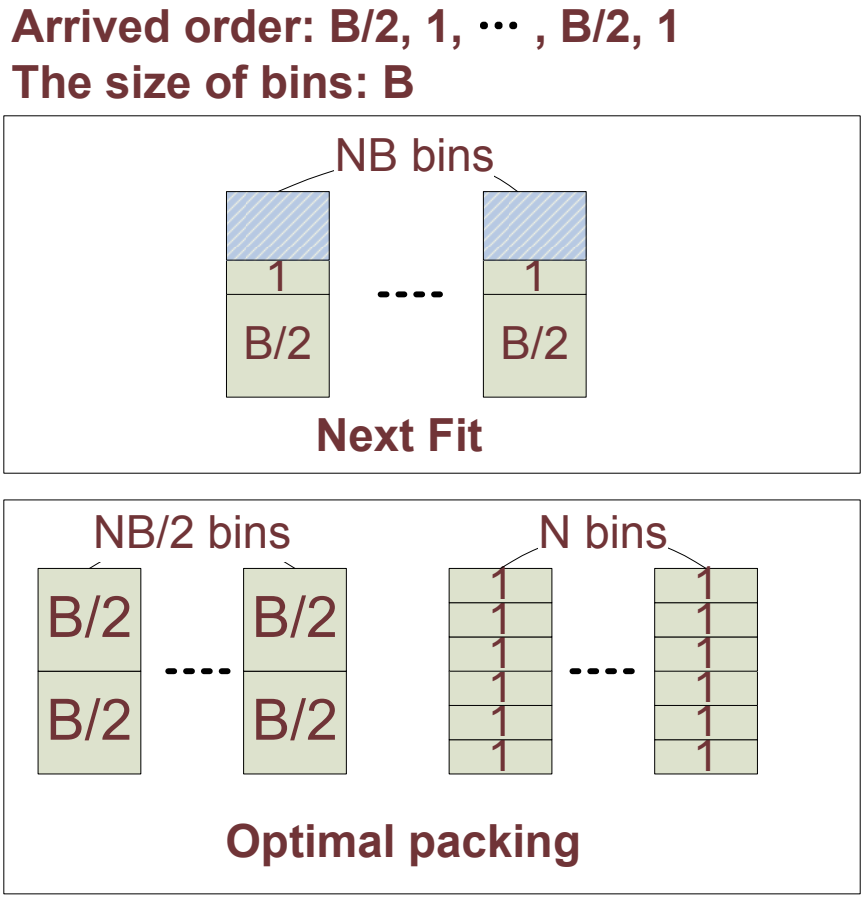

Fig. 10. An example of lower bound of performance ratio for Next Fit.

To evaluate the upper bound of its worst-case performance ratio, we next revisit the packing rule in NF. Recall that NF only opens one bin, and tries to use the opened bin to pack each arrived item. If the arrived item does not fit into the bin, it will be closed, and a new bin will be opened for the item. Based on NF's packing rule, it is obvious that the total size of packed items in bins $2i-1$ and $2i$, where $i = 1, 2, \cdots$

should be strictly greater than bins' size B. Therefore, in a packing configuration of Next Fit that uses N bins, the total sizes of packed items should be greater than $(N-1)B/2$. Note that optimal algorithms have to use at least $(N-1)/2$ bins to pack those items, which gives the upper bound of the asymptotic worst-case performance ratio of NF as:

$$R_{NF}^{\infty} \leq \lim_{N\to\infty} \frac{N}{(N-1)/2} = 2. \tag{2.6}$$

Based on (2.6) and (2.5), we get the tight bound for the asymptotic worst-case performance ratio of NF as 2.

2. FF and BF

FF and BF are the most popular online-packing algorithms. However, it is much harder to compute their worst-case performance ratios than NF. Next, we review several packing examples to analyze their lower bounds, and go over how to use weighting functions to evaluate the upper bounds.

a. Lower Bounds

Johnson *et al.* [53] showed some packing examples for BF and FF in continuous packing. In this section, we will convert those examples into discrete integral packing, and analyze their lower bounds. We may first consider the following discrete packing example that has bins of size $6B$, where $B \geq 15$, and $6N$ items of size $B-2$, $6N$ items of size $2B+1$, and $6N$ items of size $3B+1$ arrive in order. Notice that sizes $B-2$, $2B+1$, and $3B+1$ could be packed into one bin of size $6B$. Therefore, the above $18N$ items could be optimally packed into $6N$ bins, which is shown in the bottom of Figure 11. On the other hand, the top part of Figure 11 demonstrates that FF and BF will pack the first $6N$ items of size $B-2$ into N bins, pack the the second $6N$

items of size $2B+1$ into $3N$ bins, and pack the last $6N$ items of size $3B+1$ into $6N$ bins. Therefore, FF and BF will use $N+3N+6N=10N$ bins to pack the items, which makes the performance ratio of the packing example as:

$$R_{FF}^{\infty} \geq R_{FF}^{\infty}(6B) \geq \frac{10N}{6N} = 5/3. \tag{2.7}$$

$$R_{BF}^{\infty} \geq R_{BF}^{\infty}(6B) \geq \frac{10N}{6N} = 5/3. \tag{2.8}$$

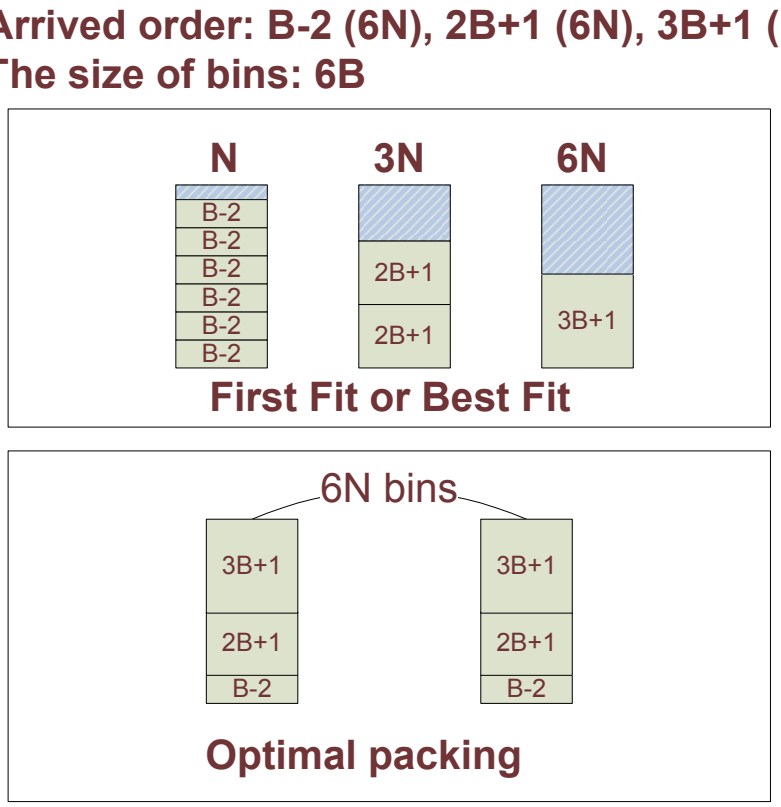

Fig. 11. Packing example for $R_{FF}^{\infty} \geq 5/3$ and $R_{BF}^{\infty} \geq 5/3$.

The above packing example could be extended to get a better lower bound as follows. Define

$$t_1 = 1 \tag{2.9}$$

$$t_{i+1} = t_i(t_i + 1), \text{ for } i = 1, 2, 3, \cdots \tag{2.10}$$

and let bins' size be $t_k B$ with condition $B \geq t_k * (k-1) + k$, items are of k different sizes, $B - (k-1), (t_k B)/(t_{k-1}+1)+1, \cdots, (t_k B)/(t_i+1)+1, \cdots, (t_k B)/(t_1+1)+1$, and each size has $t_k n$ items. The total $k t_k n$ items arrive in a non-decreasing order for packing. Note that the sum of all the k different sizes are:

$$\begin{aligned}
& B - (k-1) + \sum_{i=1}^{k-1} \left(\frac{t_k B}{t_i + 1} + 1 \right) && (2.11)\\
= \; & t_k B \left(\frac{1}{t_k} + \sum_{i=1}^{k-1} \frac{1}{t_i + 1} \right) && (2.12)\\
= \; & t_k B \left(\frac{1}{(t_{k-1}+1)t_{k-1}} + \frac{1}{t_{k-1}+1} + \sum_{i=1}^{k-2} \frac{1}{t_i + 1} \right) && (2.13)\\
= \; & t_k B \left(\frac{1}{t_{k-1}} + \sum_{i=1}^{k-2} \frac{1}{t_i + 1} \right) && (2.14)\\
& \cdots && (2.15)\\
= \; & t_k B \left(\frac{1}{t_2} + \frac{1}{t_1 + 1} \right) && (2.16)\\
= \; & t_k B. && (2.17)
\end{aligned}$$

Therefore, optimal packing algorithms for the above example should use $t_k n$ bins, and form a perfect packing. However, FF and BF will use n bins for size $B-(k-1)$, $(t_k n)/(t_{k-1}+1)$ bins for size $(t_k B)/(t_i+1)+1$, $(t_k n)/t_i$ bins for size $(t_k B)/(t_i+1)+1$. Therefore, the total number of bins used by FF and BF is: $\sum_{i=1}^{k}(t_k n)/t_i$, which give us the performance ratio for the example as:

$$R_{FF}^{\infty} \geq R_{FF}^{\infty}(t_k B) \geq \sum_{i=1}^{k} \frac{1}{t_i}. \tag{2.18}$$

$$R_{BF}^{\infty} \geq R_{BF}^{\infty}(t_k B) \geq \sum_{i=1}^{k} \frac{1}{t_i}. \tag{2.19}$$

Notice that

$$T_\infty = \lim_{k\to\infty} \sum_{i=1}^{k} \frac{1}{t_i} = 1.69103\cdots, \tag{2.20}$$

which gives another lower bound for FF and BF.

The above lower bound could be further improved through the following example, where bins' size is $B = 18^{2N}$ and N is an integer divisible by 17. Let $\delta = 18^{-2N}, 1 \leq i \leq N/17$, and $\delta_i = \delta \cdot 18^{(N/17)-i}$. For each i, we define the following coefficients:

$$\begin{aligned} a_{1,i} &= 1/6 + 33\delta_i & a_{5,i} &= 1/6 - 13\delta_i \\ a_{2,i} &= 1/6 - 3\delta_i & a_{6,i} &= 1/6 + 9\delta_i \\ a_{3,i} = a_{4,i} &= 1/6 - 7\delta_i & a_{7,i} = a_{8,i} = a_{9,i} = a_{10,i} &= 1/6 - 2\delta_i, \end{aligned}$$

and

$$\begin{aligned} b_{1,i} &= 1/3 + 46\delta_i & b_{5,i} &= 1/3 + 12\delta_i, \\ b_{2,i} &= 1/3 - 34\delta_i & b_{6,i} &= 1/3 - 10\delta_i, \\ b_{3,i} = b_{4,i} &= 1/3 + 6\delta_i & b_{7,i} = b_{8,i} = b_{9,i} = b_{10,i} &= 1/3 + \delta_i, \end{aligned}$$

and

$$c_{1,i} = c_{2,i} = c_{3,i} = c_{4,i} = c_{5,i} = c_{6,i} = c_{7,i} = c_{8,i} = c_{9,i} = c_{10,i} = 1/2 + \delta.$$

The arriving items will be sequentially in three groups, and each group has $10N/17$ items. For $i = 1, \cdots, N/17$, the first group has items of sizes:

$$a_{1,1}B, a_{2,1}B, \cdots, a_{10,1}B, a_{1,2}B, a_{2,2}B, \cdots, a_{10,2}B, \cdots, a_{1,i}B, a_{2,i}B, \cdots, a_{10,i}B, \cdots$$

Notice that the summation of the first 5 items in block i of group 1 is:

$$\sum_{j=1}^{5} a_{j,i}B = \left(\frac{5}{6} + 3\delta_i\right) B, \tag{2.21}$$

and the summation of the second 5 items in block i of group 1 is:

$$\sum_{j=6}^{10} a_{j,i}B = \left(\frac{5}{6} + \delta_i\right) B. \tag{2.22}$$

Therefore, FF and BF will pack the items in group 1 into $2N/17$ bins.

The second group also has $N/17$ block, and each block has 10 items as follows:

$$b_{1,1}B, b_{2,1}B, \cdots, b_{10,1}B, b_{1,2}B, b_{2,2}B, \cdots, b_{10,2}B, \cdots, b_{1,i}B, b_{2,i}B, \cdots, b_{10,i}B, \cdots$$

Notice that

$$\begin{aligned}
b_{1,i} + b_{2,i} &= \frac{2}{3} + 12\delta_i \\
b_{3,i} + b_{4,i} &= \frac{2}{3} + 12\delta_i \\
b_{5,i} + b_{6,i} &= \frac{2}{3} + 2\delta_i \\
b_{7,i} + b_{8,i} &= \frac{2}{3} + 2\delta_i \\
b_{9,i} + b_{10,i} &= \frac{2}{3} + 2\delta_i
\end{aligned}$$

Obviously, FF and BF will pack the 10 items of block i in group 2 into 5 bins, and the total number of bins used for group 2 is $5N/17$.

The third group has $N/17$ blocks, the 10 items in each block is:

$$c_{1,1}B, c_{2,1}B, \cdots, c_{10,1}B, c_{1,2}B, c_{2,2}B, \cdots, c_{10,2}B, \cdots, c_{1,i}B, c_{2,i}B, \cdots, c_{10,i}B, \cdots$$

FF and BF will have to pack each item in group 3 into a single bin, and use totally $10N/17$ bins. Therefore, the total number of bins used by FF and BF for the above

items in the three groups is:

$$\frac{2N}{17} + \frac{5N}{17} + \frac{10N}{17} = N. \tag{2.23}$$

Next, we consider the following combination, and show how optimal algorithms pack the above $30N/17$ items. Notice that

$$\begin{aligned}
&a_{j,i} + b_{j,i} + c_{j,i} = 1 - \delta_i + \delta \leq 1, \text{ where } 3 \leq j \leq 10, 1 \leq i \leq \frac{N}{17}, \\
&a_{1,i} + b_{2,i} + c_{1,i} = 1 - \delta_i + \delta \leq 1 \text{ where } 1 \leq i \leq \frac{N}{17}, \\
&a_{2,i} + b_{1,(i+1)} + c_{2,i} = 1 - \frac{4\delta_i}{9} + \delta \leq 1 - 8\delta, \text{ where } 1 \leq i \leq \frac{N}{17} - 1, \\
&a_{2,N/17} + b_{1,1} = \frac{1}{2} - 3\delta + 46\delta_1 < 1 \\
&c_{2,N/17} = \frac{1}{2} + \delta < 1.
\end{aligned}$$

Using the above combination, we can pack the $30N/7$ items into $8N/17 + N/17 + N/17 - 1 + 1 + 1 = 10N/17 + 1$ bins. Therefore, the performance ratio of FF and BF for the above packing example is:

$$R_{FF}(B) = R_{FF}(18^{2N}) \geq \frac{N}{10N/17 + 1} = \frac{17}{10 + 17/N}, \tag{2.24}$$

$$R_{BF}(B) = R_{BF}(18^{2N}) \geq \frac{N}{10N/17 + 1} = \frac{17}{10 + 17/N}. \tag{2.25}$$

As $N \to \infty$, the lower bound of the ratio is:

$$\begin{aligned}
R_{FF}^{\infty} &= \limsup_{B \to \infty} R_{FF}(B) \geq 1.7, \\
R_{BF}^{\infty} &= \limsup_{B \to \infty} R_{BF}(B) \geq 1.7.
\end{aligned}$$

b. Upper Bounds

Johnson *et al.* [53] showed that the worst-case performance ratios for FF and BF are upper bounded by 1.7. Next, we briefly review how the upper bound is achieved, and

get familiar with the weighting function used in the proof, which will be extended in this dissertation.

To get the upper bound, we may use the following weighting function $f(x)$ as illustrated in Figure 12.

$$f(x) = \begin{cases} \frac{6}{5B}x & x \in [0, \frac{B}{6}] \\ \frac{9}{5B}x - \frac{1}{10} & x \in [\frac{B}{6}, \frac{B}{3}] \\ \frac{6}{5B}x + \frac{1}{10} & x \in [\frac{B}{3}, \frac{B}{2}] \\ 1 & x \in (\frac{B}{2}, B] \end{cases}, \tag{2.26}$$

where B is the bins' size.

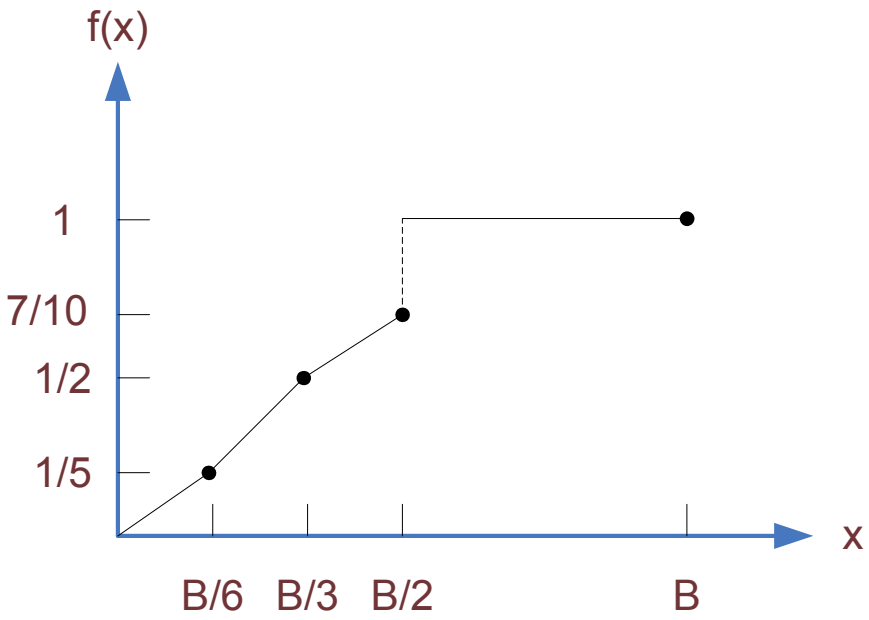

Fig. 12. Weighting function for the upper bound of R_{FF}^{∞} and R_{BF}^{∞}.

The main purpose of the weighting function is to connect the number of bins used by FF(BF) and optimal algorithms. The connection is shown in the following two lemmas.

Lemma 1 *For any set of items a_i that are packed in a single bin of size B, their total weight $\sum f(a_i)$ is less than or equal to* 1.7.

Lemma 2 *For a list of to-be-packed items L, the number of bins used by FF and BF are:*

$$FF(L,B) \leq \sum_{i \in L} f(a_i) + 3. \tag{2.27}$$

$$BF(L,B) \leq \sum_{i \in L} f(a_i) + 3. \tag{2.28}$$

For a packing list L of items a_i, lemma 1 indicates that the number of bins used by optimal algorithm is:

$$\sum_{i \in L} f(a_i) \leq 1.7 OPT(L,B). \tag{2.29}$$

Combining the above inequality with lemma 2 gives us the upper bounds of R_{FF}^{∞} and R_{BF}^{∞} as:

$$R_{FF}^{\infty} = \sup_{B \in \mathcal{R}} \limsup_{OPT(L,B) \to \infty} \frac{FF(L,B)}{OPT(L,B)} \leq 1.7. \tag{2.30}$$

$$R_{BF}^{\infty} = \sup_{B \in \mathcal{R}} \limsup_{OPT(L,B) \to \infty} \frac{BF(L,B)}{OPT(L,B)} \leq 1.7. \tag{2.31}$$

For the detailed proof of the above two lemmas, please refer to [53].

3. FFD and BFD

Recall that FFD and BFD are offline algorithms, and that they first sort items in non-decreasing order, then pack the sorted items using FF and BF algorithms respectively. Next, we briefly review the lower bound and the upper bound of their asymptotic worst-case performance ratios.

a. Lower Bounds

The lower bounds of the ratios could be achieved via the following packing example, where bins' size is B such that $B \bmod 4 = 0$ and $B \geq 44$. The arriving items are of four different sizes as follows.

$$\left(\frac{B}{2}+1\right), \quad \left(\frac{B}{4}+2\right), \quad \left(\frac{B}{4}+1\right), \quad \left(\frac{B}{4}-2\right). \tag{2.32}$$

The first three sizes have $6N$ copies, and the last one has $12N$ copies. Note that FFD and BFD first sort the items in non-decreasing order, and will pack items into $11N$ bins as shown in top of Figure 13. On the other hand, the summation of the first, third, and last sizes is:

$$\left(\frac{B}{2}+1\right)+\left(\frac{B}{4}+1\right)+\left(\frac{B}{4}-2\right)=B, \tag{2.33}$$

and that the summation of two copies of the second and the last sizes is:

$$2\left(\frac{B}{4}+2\right)+2\left(\frac{B}{4}-2\right)=B. \tag{2.34}$$

Therefore, optimal algorithms will pack the items into 9N bins as shown in the bottom of Figure 13. This gives us the lower bounds of FFD's and BFD's worst-case performance ratio as:

$$R_{FFD}^{\infty} \geq \frac{11}{9}, \tag{2.35}$$

$$R_{BFD}^{\infty} \geq \frac{11}{9}. \tag{2.36}$$

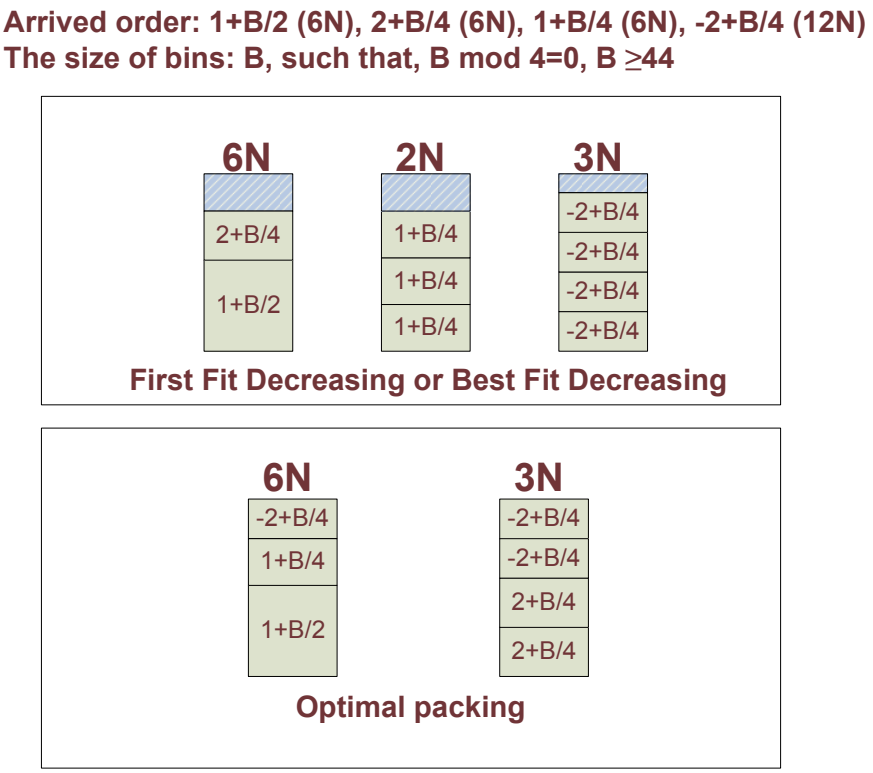

Fig. 13. Packing example for $R^{\infty}_{FFD} \geq 11/9$ and $R^{\infty}_{BFD} \geq 11/9$.

b. Upper Bounds

Using more than 70 pages in his dissertation, Johnson [52] showed the relationship between FFD and optimal packing as:

$$FFD(L,B) \leq \frac{11}{9}OPT(L,B) + 4, \quad \forall L, B \tag{2.37}$$

Later Baker [6] simplified the proof and reduced the constant from 4 to 3, and got result

$$FFD(L,B) \leq \frac{11}{9}OPT(L,B) + 3, \quad \forall L, B \tag{2.38}$$

and Yue [79] further significantly improved the proof and reduced the constant to 1, which is:

$$FFD(L,B) \leq \frac{11}{9}OPT(L,B) + 1. \quad \forall L, B \tag{2.39}$$

Regardless of the constants in the inequalities, all the results indicate

$$R_{FFD}^{\infty} = \sup_{B \in \mathcal{R}} \lim_{OPT(L,B) \to \infty} \frac{FFD(L,B)}{OPT(L,B)} \leq \frac{11}{9}. \tag{2.40}$$

Regarding BFD, Johnson [52], [53] showed that $BFD(L) \leq FFD(L)$ if no to-be-packed item is smaller than $B/6$. Therefore, if each item a_i in packing list L meets condition $a_i \geq B/6$, we have:

$$BFD(L) \leq FFD(L) \leq \frac{11}{9}OPT(L) + 1, \ \forall a_i \in L \text{ s.t. } a_i \geq \frac{B}{6}. \tag{2.41}$$

In the case where to-be-packed items include size less than $B/6$, we need the following two case-analyses. In case 1, all those items of size less than $B/6$ do not start a new bin in BFD, and we still have:

$$BFD(L) \leq FFD(L) \leq \frac{11}{9}OPT(L) + 1. \tag{2.42}$$

On the other hand, if any item of size less than $B/6$ starts a new bin, then all bins except the last one must be $5/6$ full, therefore,

$$BFD(L) \leq \frac{6}{5}OPT(L) + 1. \tag{2.43}$$

Combining inequalities (2.41), (2.42), and (2.43), we have:

$$R_{BFD}^{\infty} \leq \frac{11}{9}. \tag{2.44}$$

4. Sum of Squares

a. Upper Bounds

Csirik *et. al.* [38] showed that the performance ratio of sum of squares (SS) is upper bounded by 3.0, which is further improved to 25/9 in [39].

b. Lower Bounds

To get a lower bound of worst-case performance ratio of SS, we may use the following packing example, where bins' size is $2B + 1$, and there are $B(B + 1)N$ items. In optimal packing, B items will be packed into a single bin, and it uses $(B + 1)N$ bins as shown on the top of Figure 14.

Recall that SS tries to minimize the sum of the squares of the number of packing levels $\sum_{l=1}^{2B} N^2(l)$. In the packing example, SS will produce levels $2i$, where $i = 1, \cdots, B$. To minimize the sum of the square, the number of each level should be the same, which is $2N$. SS uses $2BN$ bins as shown in the bottom of Figure 14. Therefore, the performance ratio of SS is:

$$R_{SS}^{\infty}(B) \geq \lim_{N \to \infty} \frac{2BN}{(B+1)N} = \frac{2B}{B+1}, \tag{2.45}$$

which further indicates

$$R_{SS}^{\infty} = \sup_{B \in \mathcal{R}} R_{SS}^{\infty}(B) \geq 2. \tag{2.46}$$

C. Average-Case Analysis

In the average-case analysis of bin packing algorithms, **expected waste** is a primary measure used to evaluate the algorithms' typical behavior. Next, we first review the notation of expected waste, and then present the expected waste for optimal

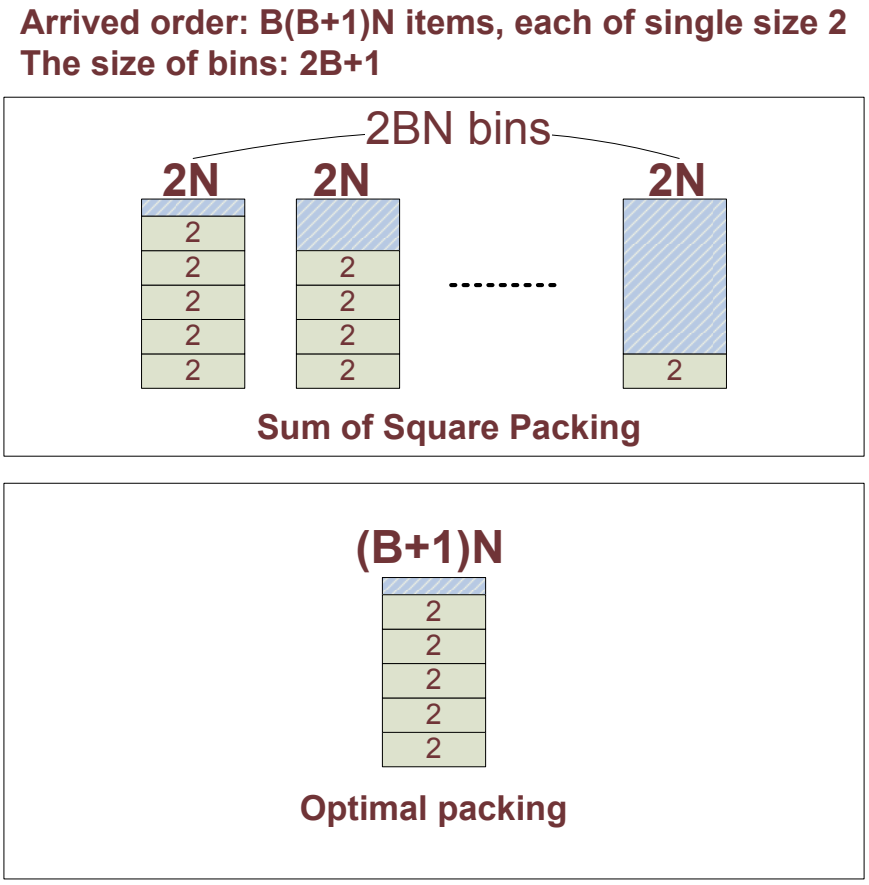

Fig. 14. An example of $R_{SS}^{\infty} \geq 2$.

algorithm, and First Fit and Best Fit.

1. Notations

Recall that the *waste* of a packing is referred to the waste area in the packing. Figure 15 shows an example of waste, which is marked as the shaded areas in the figure. In the example, the waste is 1 for bin 1, 3 for bin 2, and 2 for bin 3. The total waste for all three bins is: $1 + 3 + 2 = 6$.

To evaluate the typical behavior of packing algorithm A, researchers normally

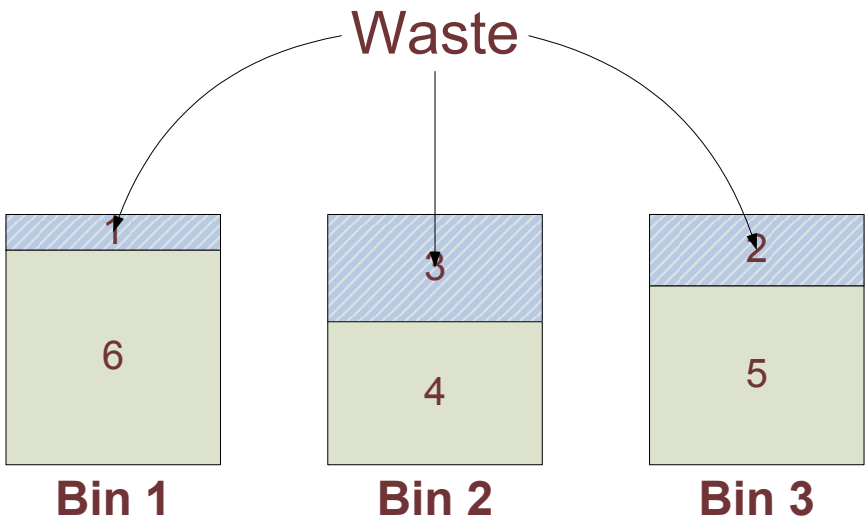

Fig. 15. An example of waste.

first select some distribution, then independently generate n random sizes (items) under the distribution, and finally estimate the expectation of the total waste of the packing after algorithm A packs the n items. The expectation of the waste is referred as *expected waste*, and denoted by EW_n^A.

For discrete packing, the most popular distribution for expected waste is called *discrete uniform distribution* $U\{J, B\}$, where each size i, $i = 1, \cdots, J$, arrives with equal probability $\lambda_i = 1/J$.

For example, if items are randomly chosen from set $\{4, 5\}$ with equal probability $p = 0.5$, and bin's size is 7, algorithm FF or BF has to pack each item into one bin. The waste for a single bin that packs size 4 is $7 - 4 = 3$, and for each bin that packs size 5 is $7 - 5 = 2$. Since items' sizes could either be 4 or 5 with equal probability $p = 0.5$, we have the expected waste for algorithm FF and BF as:

$$EW_n^{FF} = EW_n^{BF} = (3 * 0.5 + 2 * 0.5)n = 2.5n. \tag{2.47}$$

2. Expected Waste of OPT

Recall that optimal algorithms use the minimum number of bins, and it may also have waste for some packing list. We denote its expected waste by EW_n^{OPT}.

a. Asymptotic Forms of EW_n^{OPT}

Courcoubetis *et al.* [33], [34], [35] analyzed EW_n^{OPT} in the following problem: bins' size is B, items' sizes are independently and randomly chosen from integral set $\{1, \cdots, J\}$, $J \leq B$, with PMF:

$$P[L = i] = \lambda_i, \tag{2.48}$$

where $i = 1, \cdots, J$, $\lambda_i \geq 0$, and $\sum_{i=1}^{J} \lambda_i = 1$.

In the problem, there are total J possible items, and some combinations of these items could be perfectly packed into a single bin without any waste. Noticing that the perfect combinations are closely related to the EW_n^{OPT}, Courcoubetis *et al.* defined a perfect configuration as a tuple of size J. Precisely, it is:

$$c_k = (c_k[1], \cdots, c_k[J]), \tag{2.49}$$

where $\sum_{i=1}^{J} i c_k[i] = B$, and $c_k[i] \in Z^+ \bigcup \{0\}$. Suppose that there are total K such perfect configurations, they further defined a *perfect packing matrix* as:

$$Q = (c_1, c_2, \cdots, c_K), \tag{2.50}$$

and denoted by $\Lambda = \{Qf : f \geq 0\}$ the cone spanned by Q.

Using the above notations, they analyzed optimal expected waste EW_n^{OPT}, and showed that EW_n^{OPT} can only take three asymptotic forms. Precisely,

- $EW_n^{OPT} = \Theta(1)$ if the probability tuple $\lambda = (\lambda_1, \cdots, \lambda_J)$ is in the interior of

cone Λ

- $EW_n^{OPT} = \Theta(\sqrt{n})$ if λ lies on the boundary of Λ
- $EW_n^{OPT} = \Theta(n)$ if λ is outside of Λ.

b. EW_n^{OPT} Under $U\{B-2, B\}$

As shown in the previous section, optimal expected waste is closely related to packing configurations, which generate the cone Λ. To estimate the optimal expected waste under discrete distribution, Coffman *et al.* studied the perfect packing configurations, and developed a perfect packing theorem as follows.

Theorem 3 *In a bin packing problem, bins' size is B and maximum item's size is J, $J \leq B$. One can perfectly pack a list L consisting of RJ items, R each of size* 1 *through J into bins* if and only if *the sum of the RJ item sizes is a multiple of bins' size B.*

The theorem gives a condition under which items could be perfectly packed into bins. Coffman further used the perfect packing theorem and showed that the probability tuple under distribution $U\{B-2, B\}$ is in the cone of Λ, spanned by the perfect packing matrix. Therefore, we have:

$$EW_n^{OPT} = \Theta(1), \tag{2.51}$$

under distribution $U\{B-2, B\}$.

3. Theoretical Packing Algorithm

Recall that $\Lambda = \{Qf : f \geq 0\}$ is a cone spanned by a perfect-packing matrix. If items' size distribution $\lambda \in \Lambda$ and items' arriving process is Poisson, Courcoubetis *et*

al. [33], [34] presented an approximation algorithm A with

$$EW_n^A = \Theta(EW_n^{OPT}) \tag{2.52}$$

Even though the algorithm does not need to know λ, and has polynomial running-time with the number of items n; it is too complicated and is only of theoretical interest.

4. Expected Waste of FF and BF

Past works also analyzed the expected waste of FF and BF under discrete uniform distribution.

Coffman **et al.** [29] proposed the approach of multi-dimensional Markov Chains. Kenyon **et al.** [56] then applied the method and found $EW_n^{BF} = \Theta(1)$ under the distribution $U\{B-2, B\}$. Albers **et al.** [1] further applied the Markov Chain technique to First Fit and showed that $EW_n^{FF} = \Theta(1)$ under distribution $U\{B-2, B\}$.

5. Expected Waste of SS

Regarding algorithm SS, Csirik *et al.* [38] showed that

- $EW_n^{SS} = O(\sqrt{n})$ if $EW_n^{OPT} = O(\sqrt{n})$,
- $EW_n^{SS} = O(\log n)$ or $O(1)$ if $EW_n^{OPT} = O(1)$.

Csirik *et al.* [38] also presented a variant online algorithm $SS^\star$ based on SS, and showed that $EW_n^{SS^\star} = \Theta(EW_n^{OPT})$. Algorithms SS and $SS^\star$ are more practical than the algorithms in [33] and [34], because they are simpler and easier to implement.

CHAPTER III

EXTENSION FRAMEWORK

As shown in the previous chapter, there exist many 1D packing algorithms with well analyzed performance. For example, the asymptotic worst-case performance ratio for FF and BF is 1.7, for FFD and BFD is 11/9. The problem of whether or not we can apply these algorithms to higher-dimensional packing is of great interest. Our *extension framework* that will be presented in this section is to serve this purpose.

Our extension framework is a hyperbox packing algorithm that takes two inputs: 1) 1D packing algorithms like FF, BF etc. 2) hyperbox packing instance, and outputs a packing solution for the packing instance. An overview of the framework is shown in Figure 16.

The extension framework offers at least the following three benefits:

- It shows the possibility of extending 1D algorithms and their analyses to higher dimensional packing
- It provides a general platform for the extension
- It produces many effective multi-dimensional packing algorithms

Next, we will introduce some framework-related concepts, then present how the framework works, and finally go over some framework-based algorithms.

A. Framework-Related Concepts and Notations

To better understand our framework algorithm, we will need the following concepts and definitions.

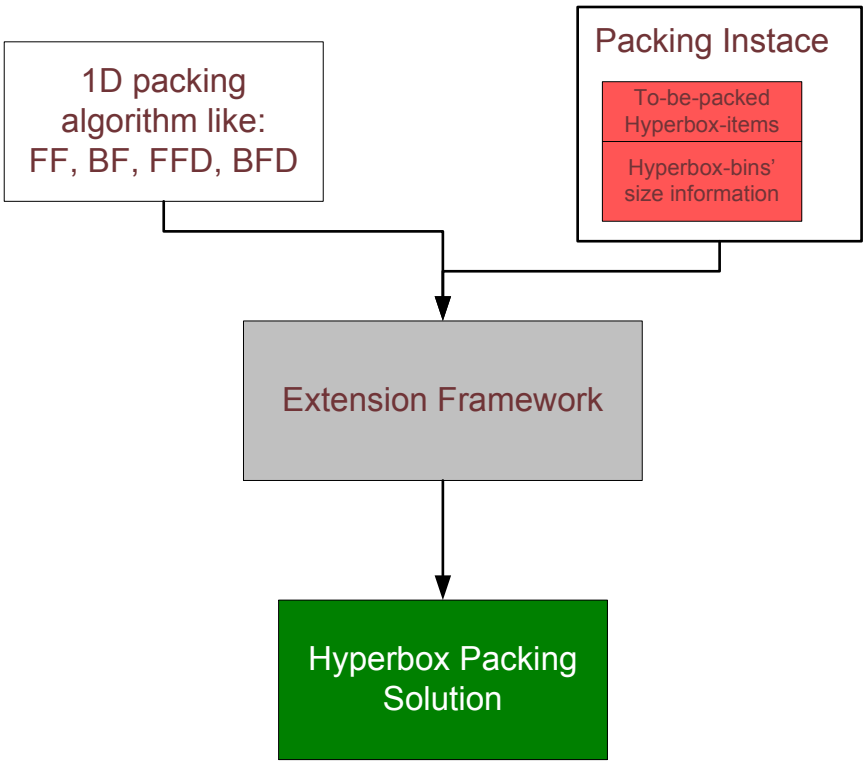

Fig. 16. Overview of the extension framework.

1. dth Surface of Hyperbox

We refer to the surface of a D-dimensional hyperbox as a hyperplane of $D-1$ dimensions. The dth surface, where $d = 1, \cdots, D$, of a hyperbox is the surface normal to dimension d. Note that for each D-dimensional hyperbox, there exist two dth surfaces as follows:

$$x_d = c, \tag{3.1}$$

$$x_d = c + l_d, \tag{3.2}$$

where l_d is the edge length of the hyperbox in the dth dimension. In the dissertation, the dth surface refers to the one in equation (3.1), which is the left or bottom surface

in 2D case. Figure 17 shows an example of the dth surface in 2D packing.

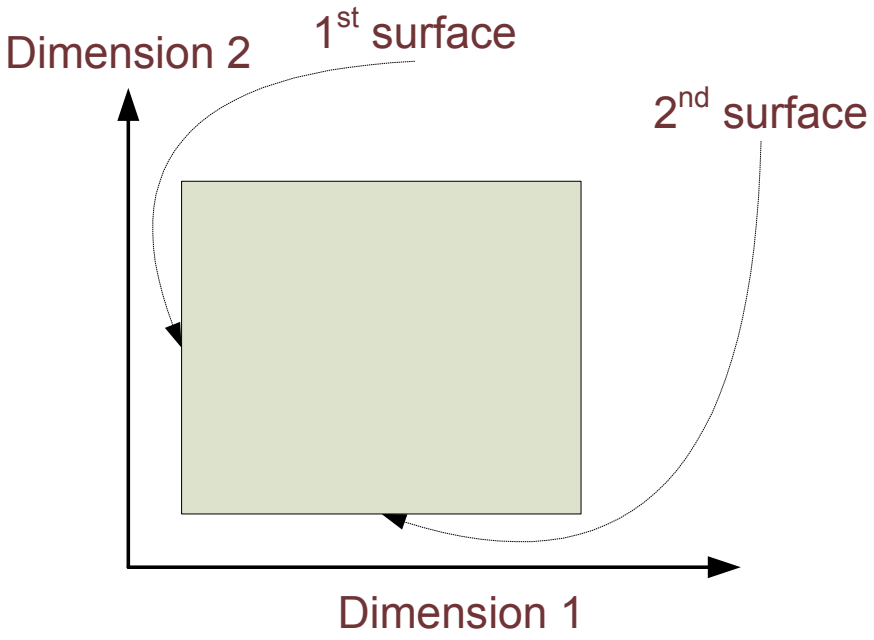

Fig. 17. Example of dth surface in 2D packing.

2. d-Adjacent

The first concept in our framework is *d-adjacent*, which is closely related to the order of the items' dimensions. The formal definition of d-adjacent is:

Definition A.1 *In a D-dimensional hyperbox packing, we mark the order of the dimensions arbitrarily. Two items are said to be d-**adjacent**, $d = 1, \cdots, D$, if they meet the following three conditions: 1) they are adjacent to a surface S that has dimension $D-1$, 2) the dth dimension is perpendicular to surface S, and 3) the two items have the same edge-length in the ith dimension where $i = 1, \cdots, d-1$.*

Figure 18 gives an example of d-adjacent in 2 dimensional packing, where bins and items are rectangles. In the example, items have 2 dimensions, the first dimension

(dimension X) is horizontal, and the second one (dimension Y) is vertical.

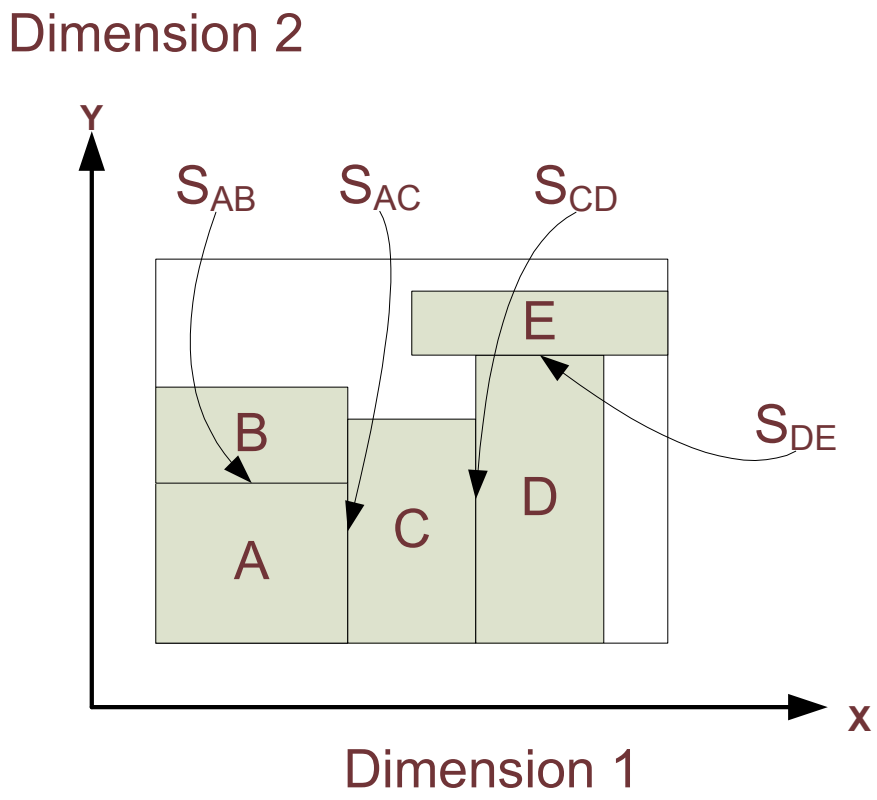

Fig. 18. Example of d-adjacent in 2D packing.

Note that items A and B touch together, and 1) have a common surface S_{AB}, 2) the second dimension (dimension Y) is perpendicular to surface S_{AB}, and 3) items A and B have the same width (same edge length in the first dimension). Therefore, items A and B are 2-adjacent in the packing example. Items D and E have different edge length in the first dimension, and so do not meet the condition 3 in definition A.1. Therefore, they are not 2-adjacent. Items A and C touch together and share a common surface S_{AC}, and the surface is normal to the first dimension (dimension X). Therefore, they are 1-adjacent. Similarly, items C and D are also 1-adjacent.

We next check the adjacent items in a 3D bin packing as shown in Figure 19. In the example, 3D-items A and B 1) share a common surface, 2) the surface is perpendicular to dimension 3, and 3) they have the same edge length in dimensions 1 and 2. Therefore, items A and B are 3-adjacent. Similarly, items A and D are 2-adjacent, and items A and E are 1-adjacent. However, items B and C are not 3-adjacent since their edge lengths in dimension 2 are not the same. Similarly, items D and F are not 2-adjacent because their edge lengths in dimension 1 are not the same.

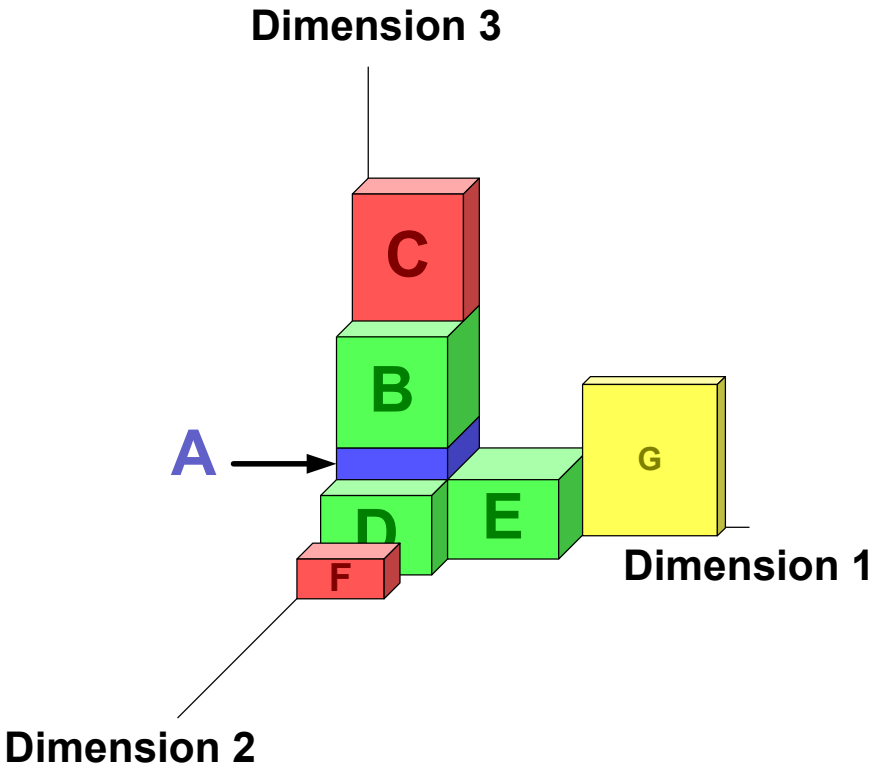

Fig. 19. Example of d-adjacent in 3D packing.

3. d-Pile and d-Length

The next concept that will be frequently used in our framework is called a d-pile, and its edge-length in the dth dimension is called d-length. d-pile is a sequence of d-adjacent items. Precisely,

Definition A.2 *In a hyperbox packing with D dimensions, d-*pile *is a sequence of either hyperbox-items or $(d+1)$-piles, in which each item or $(d+1)$-pile except the first one is successively d-adjacent to previous one.*

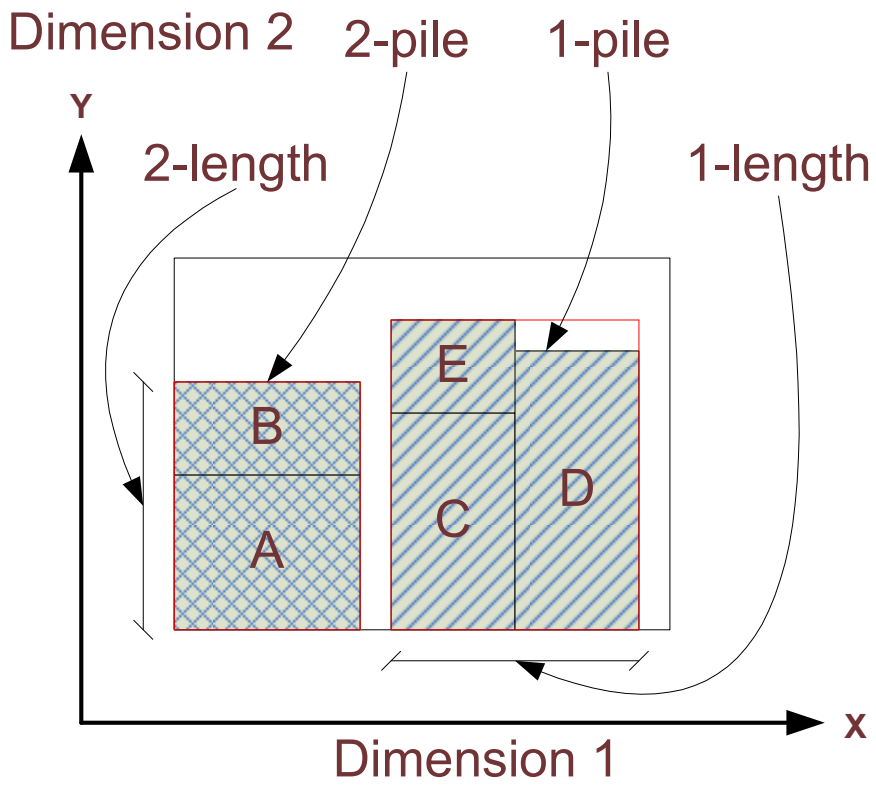

Fig. 20. Example of d-pile and d-length in 2D packing.

The edge length of d-pile in dth dimension is referred as d-length. Its formal definition is:

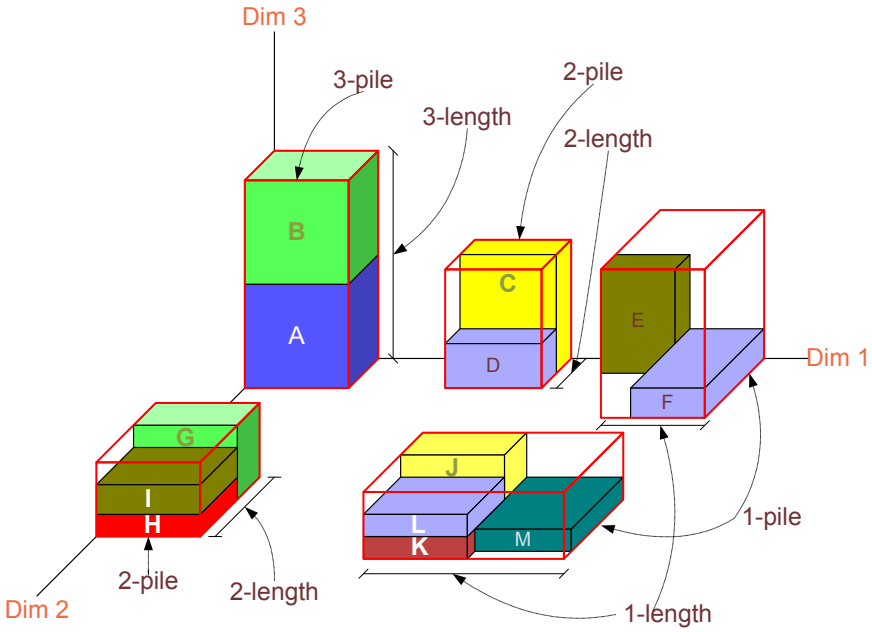

Fig. 21. Example of d-pile and d-length in 3D packing.

Definition A.3 *In a d-pile P, its* d-length *refers to the sum of the edge-lengths of all pile-items in dimension d.*

In Figure 20, items A and B form a 2-pile since A and B are 2-adjacent. Items C and E form a 2-pile since they are 2-adjacent. Also notice that items C, D, and E form a 1-pile, because the 2-pile consisting of items C and E, item C, and item D are 1-adjacent.

Figure 21 gives a few examples of d-pile and d-length in 3D packing, where all d-piles and d-lengths are marked and noted correspondingly. In the example, items A and B form a 3-pile since they are 3-adjacent, items C and D form a 2-pile since they are 2-adjacent, items E and F are 1-adjacent since they share a common surface. Note that items I and H form a 3-pile, which again with item G forms a 2-pile, items J, K, L form a 2-pile, which along with item M forms a 1-pile.

4. Virtual Bin

The last concept is virtual bin, which is associated with some d-pile.

Definition A.4 *In a hyperbox-bin of size $B = (B_1, B_2, \cdots, B_D)$, for any d-pile P of d-length l, its associated virtual bin is a 1D bin of size B_d that is partially filled 1D item of size l.*

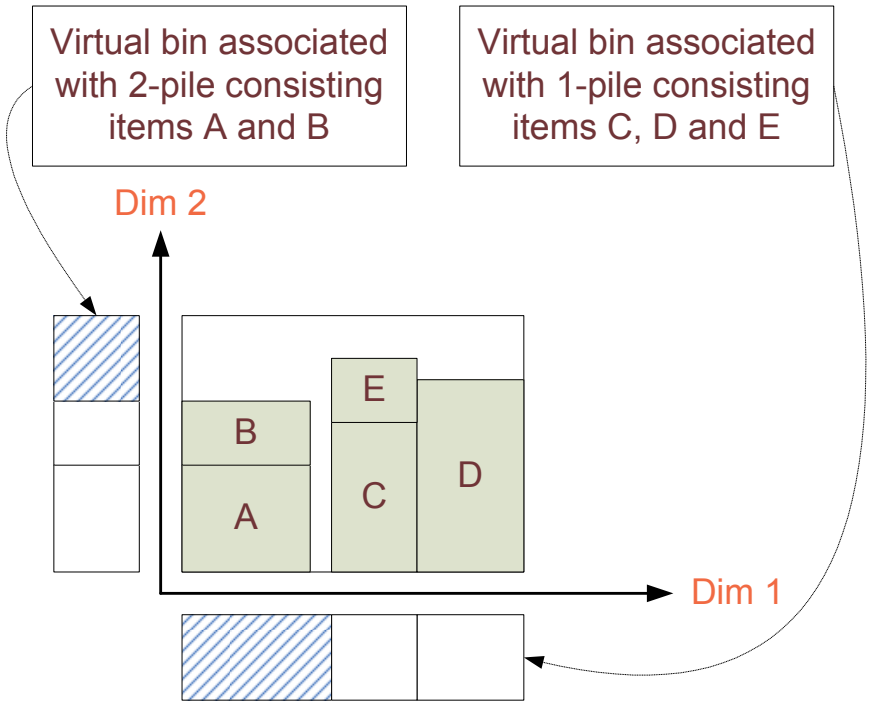

Fig. 22. Example of virtual bins in 2D packing.

Figure 22 gives examples of virtual bins on 2D packing. Items A and B form a 2-pile, and its virtual bin is shown on the left of the figure. Items C, D, and E form a 1-pile, and its virtual bin is shown at the bottom of the figure.

B. How Does the Extension Framework Work?

1. Key Points

The basic idea of the framework is to use *strip packing rule*, convert hyperbox packing problem into 1-dimensional packing problem, and apply 1D packing algorithm on the divided 1D packing problem. Following lists some key points on our framework.

a. Strip Packing Constraint

In a hyperbox packing, we refer the *Strip Packing rule* to the requirements that any items that share a common surface S of dimension $D-1$ should be d-adjacent, where surface S is normal to dimension d, and $d = 1, \cdots, D$.

Our framework should strictly follow the strip packing rule. Precisely, any two items that share a surface of dimension $D-1$ that is normal to dimension d are d-adjacent, where $d = 1, \cdots, D$.

b. Pile Length Constraint

In each occupied hyperbox-bin, items packed by framework will form many d-piles, where $d = 1, \cdots, D$. The d-length of the piles should be less than or equal to the edge length B_d of hyperbox-bin in dth dimension.

c. Offline Sorting

If input algorithm in Figure 16 is an offline algorithm like FFD or BFD, framework will sort the input hyperbox items on the D edge lengths via lexicographical order. Precisely, for two hyperbox item $a = (l_1, l_2, \cdots, l_D)$, and $a' = (l'_1, l'_2, \cdots, l'_D)$, we define

1) $a = a'$ if and only if $l_d = l'_d, \forall d = 1, \cdots, D$.

2) $a < a'$ if and only if $\exists k, 1 \leq k \leq D$ such that $l_k < l'_k$, and $\forall d = 1, \cdots, k-1, l_d = l'_d$.

3) $a > a'$ if and only if $\exists k, 1 \leq k \leq D$ such that $l_k > l'_k$, and $\forall d = 1, \cdots, k-1, l_d = l'_d$.

For example, for 3D items of sizes

$$a_1 = (3, 2, 1),$$

$$a_2 = (1, 3, 4),$$

$$a_3 = (2, 1, 3),$$

$$a_4 = (2, 2, 1),$$

the decreasing order of the items will be: a_1, a_4, a_3, a_2.

d. Dimension Ordering

The order of dimensions is critical to the framework packing process. Once a to-be-packed item arrives, framework will test dimensions $d = D, D-1, \cdots, 1$ in order, and check if the item could be d-adjacent to any existing d-pile.

At the beginning, if there exists any such D-pile, the to-be-packed item will be packed into one such pile, and packing process stops. Otherwise, framework will check if there are any $(D-1)$-piles that the item could be $(D-1)$-adjacent. If such $(D-1)$-piles do exist, framework will pick one of them, and pack the item in the following way: a) the item will be $(D-1)$-adjacent to the pile, b) the D-surface that is normal to dimension D will be adjacent to the D-surface of the hyperbox-bin. If no such $(D-1)$-piles exist, framework will use the similar pattern to check dimension $D-2$, $D-3, \cdots, 1$.

Generally, for $d = D-1, \cdots, 1$, if no $D, D-1, \cdots, (d+1)$-pile exists for the

to-be-packed item, then the d-piles will be checked. If there exist any such d-pile, framework will choose one of the d-piles, and pack the item d-adjacent to the pile, and the d'-surface, where $d' = D, D-1, \cdots, d+1$, of the item will be adjacent to the d'-surface of the bin.

e. Use 1D Algorithm and Virtual Bin to Choose d-Pile

Suppose that framework is trying to pack an item of size $(l_1, l_2, \cdots, l_D)$ into a bin, and the item could not be $d' = D, D-1, \cdots, (d+1)$-adjacent to existing piles. When framework checks dimension d, it finds that there exist more than one d-pile that the item could be d-adjacent. In this case, which d-pile the framework should choose for the item?

In the above case, let's assume there are N eligible d-piles, $P_1, P_2, \cdots P_N$ for the item, and framework will choose pile via the following approach. It will first identify virtual bins for piles $P_1, \cdots, P_N$, then apply 1D algorithm in Figure 16 to pack 1D item of size l_d to the available virtual bins, which should find a virtual bin. The d-pile associated with the virtual bin will be the one that framework will use.

During d-pile choosing process, if 1D algorithm (like Next Fit) closes any virtual bin, the associated d-pile will never be considered as eligible candidate for any future items.

2. Framework Algorithm

The framework algorithm is listed as Figure 23.

Input: 1) a hyperbox packing instance $\mathcal{I}$ with D dimensions,
2) 1D packing algorithm A.

Output: a hyperbox packing solution to the instance $\mathcal{I}$

1. If algorithm A is offline (like FFD, and BFD), and needs to sort input items before packing, framework will sort the input hyperbox-items based on their edge lengths in lexicographical order.
2. For each arrived item a_t of size $(l_1, l_2, \cdots, l_D)$
3. For $d = D, D-1, \ldots, 1$
4. If there exist N eligible d-piles $P_1, P_2, \cdots, P_N$ that a_t could be d-adjacent to then
5. Identify the virtual bins associated with piles $P_1, P_2, \cdots, P_N$
6. Apply 1D algorithm to 1D packing problem where bins size is B_d, and item's size is l_d, and choose a virtual bin. If the 1D algorithm closes any virtual bin, the associated d-pile will never be considered as a candidate in step 4.
7. Pack a_t d-adjacent to the chosen d-pile. The surface of a_t in dimension $d' = d+1, \ldots, D$ should be adjacent to the d'th surface of hyperbox bin.
8. Go to step 2.
9. End If
10. End For
11. Open a new hyperbox-bin, and pack a_t in the new bin such that the ith surface of a_t is adjacent to the ith surface of the bin, where $i = 1, \cdots, D$.
12. End For

Fig. 23. Framework algorithm.

The following scenario illustrates how our framework packs D-dimensional items. Suppose a hyperbox-item a_t of size $(l_1, \cdots, l_{D-1}, l_D)$ arrives. Our framework will first search for all D-piles of D-length $\leq (B_D - l_D)$, and of size $(l_1, \cdots, l_{D-1})$ in the first $(D-1)$ dimensions. If the D-piles do exist, it then puts a_t D-adjacently to one of the hyperbox-piles, which is chosen by applying some 1D packing algorithm like FF, BF, FFD, or BFD on their D-lengths. Otherwise, the framework will search all $(D-1)$-piles of $(D-1)$-length $\leq (B_{D-1} - l_{D-1})$, and of size $(l_1, \cdots, l_{D-2})$ in the first $(D-2)$ dimensions. If the $(D-1)$-piles do exist, one pile will be picked up using 1D packing algorithm, and a_t will be $(D-1)$-adjacent to the pile and D-adjacent to the surface of the bin that contains the pile. If no $(D-1)$-pile is found, a similar rule will be recursively applied until reaching the first dimension. If no 1-pile of 1-length $\leq (B_1 - l_1)$ exists, then a new hyperbox-bin will be opened, and a_t will be d-adjacent with $d = 1, \cdots, D$, to the surface of the newly opened bin.

C. Framework-Based Algorithms

In this section, we will first list frequently discussed framework-based algorithms, and then give some 2D and 3D packing examples of these algorithms.

1. Notations of Framework Algorithms

As shown before, framework provides a general approach of using 1D packing algorithms to hyperbox packing problem. To better differentiate the 1D algorithm and framework algorithms, we put a prefix E to the front of 1D algorithms, and mark them as framework-based algorithms. For example, EBF, and EFF are the corresponding framework algorithms for 1D algorithms Best Fit and First Fit. Table I lists the framework based algorithm that we will frequently reference in this dissertation.

Table I. Notation of framework algorithms used in this dissertation.

Framework Algorithm	**1D Algorithm**	**Description**
ENF	NF	Next Fit Packing
EFF	FF	First Fit Packing
EBF	BF	Best Fit Packing
ESS	SS	Sum of Squares Packing
EFFD	FFD	First Fit Decreasing
EBFD	BFD	Best Fit Decreasing

2. 2D Packing Examples

To better understand how framework works, we next give a 2D packing instance, and check how framework-based algorithms ENF, EFF, EBF, ESS, EFFD, and EBFD pack the instance. In the example, bins' size is 7×5, which has width 5 and height 7. The item list is: $3 \times 2, 3 \times 6, 3 \times 4, 2 \times 4, 2 \times 5, 2 \times 1$, and 2×3.

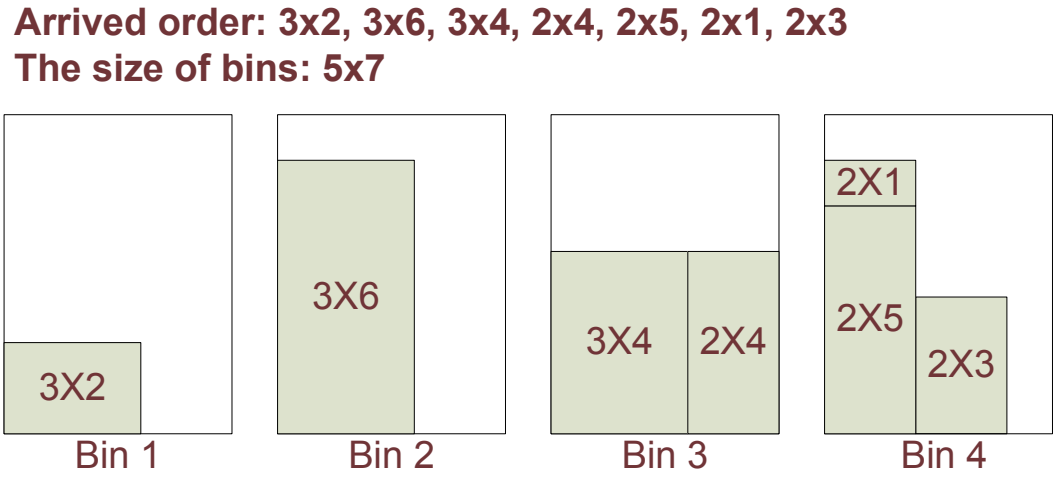

Fig. 24. Example of algorithm ENF in 2D packing.

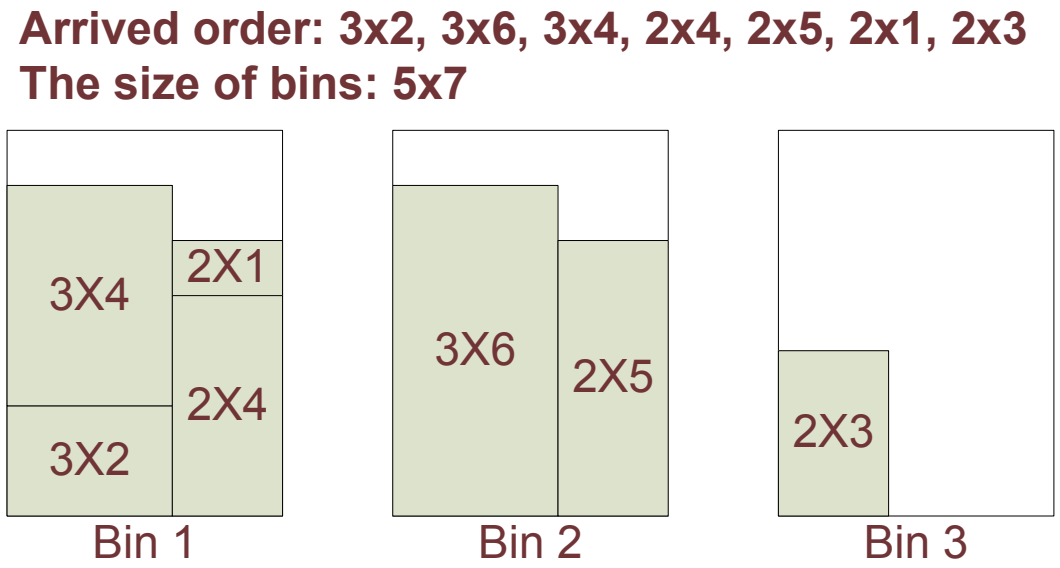

Fig. 25. Example of algorithm EFF in 2D packing.

Figure 24 shows the packing solution for ENF, where it uses 4 bins. Obviously, the first item of size 3×2 will be packed into the first bin. When the second item of size 3×6 comes, ENF checks the 2-pile and 1-pile in bin 1, and realizes that neither of them could be d-adjacent to the arriving item. Therefore, ENF closes the virtual bins associated with 2-pile and 1-pile in bin 1, and opens bin 2 for the item. Similarly, when the item of size 3×4 comes, virtual bins associated with 1-pile and 2-pile in bin 2 are also closed, and bin 3 is opened for the item of size 3×4. Next, when item of size 2×4 comes, ENF could not pack it into either bin 1 or bin 2, because the associated virtual bins are all closed. In this case, ENF packs it into bin 3. When next item of size 2×5 comes, EFF closes the virtual bin associated with item 2×4 in bin 3, and opens a bin 4 for the arriving time. Using the similar pattern, ENF packs the rest items of sizes 2×1, and 2×3 into bin 4.

Figure 25 shows how EFF packs the 2D instance. Like in ENF, the first two

items of sizes 3×2 and 3×6 are packed into bins 1 and 2. However, since FF does not close partially-filled bins, virtual bins associated with the 2-pile and 1-pile in bin 1 are not closed. When item of size 3×4 comes, EFF will pack it 2-adjacent to the 2-pile of size 3×2 in bin 1. When next item of size 2×4 comes, EFF could not find any eligible 2-pile, but has two candidate 1-piles in bins 1 and 2 that the arriving item could be 1-adjacent. EFF chooses the first 1-pile in bin 1, and makes it 1-adjacent to item of size 2×4. Items of size 2×1 is also packed into bin 1, and 2-adjacent to the 2-pile of size 2×4 because the 2-pile is the first candidate. Similarly, the rest two items of sizes 2×5 and 2×3 are packed into bins 2 and 3 respectively.

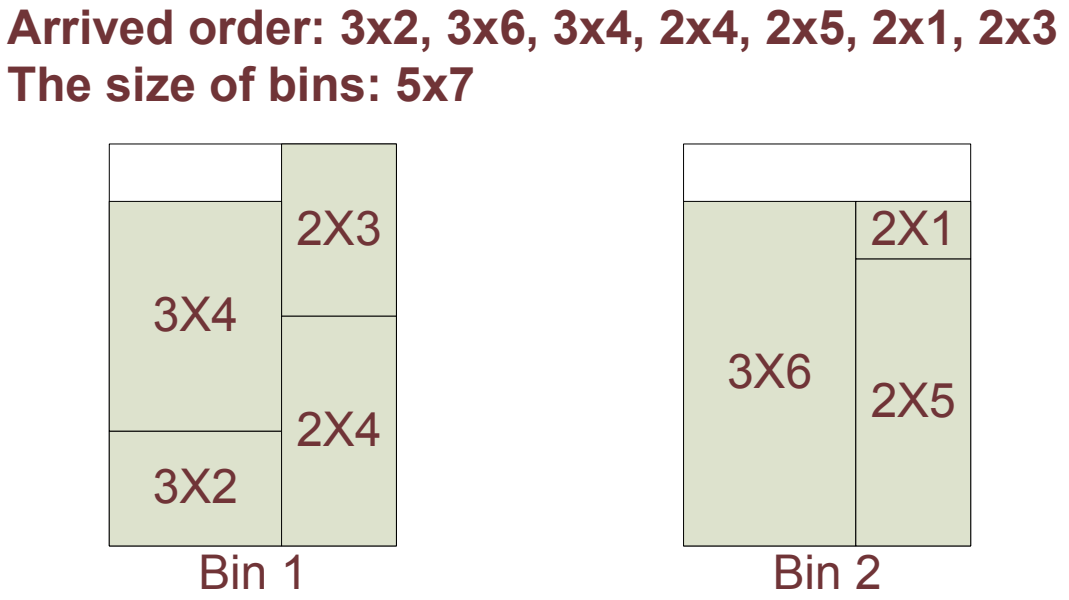

Fig. 26. Example of algorithm EBF in 2D packing.

Figure 26 shows the packing solution of framework-based algorithm EBF. Note that EBF uses the same way to packing the first 5 items of sizes $3\times2, 3\times6, 3\times4, 2\times4$, and 2×5. However, when the next item of size 2×1 comes, it makes the item 2-adjacent to the 2-pile of size 2×5 in bin 2 because the associate virtual bin fits the

arriving item better. Regarding the last item of size 2×3, EBF uses the virtual bin associated with 2-pile of size 2×4 in bin 1.

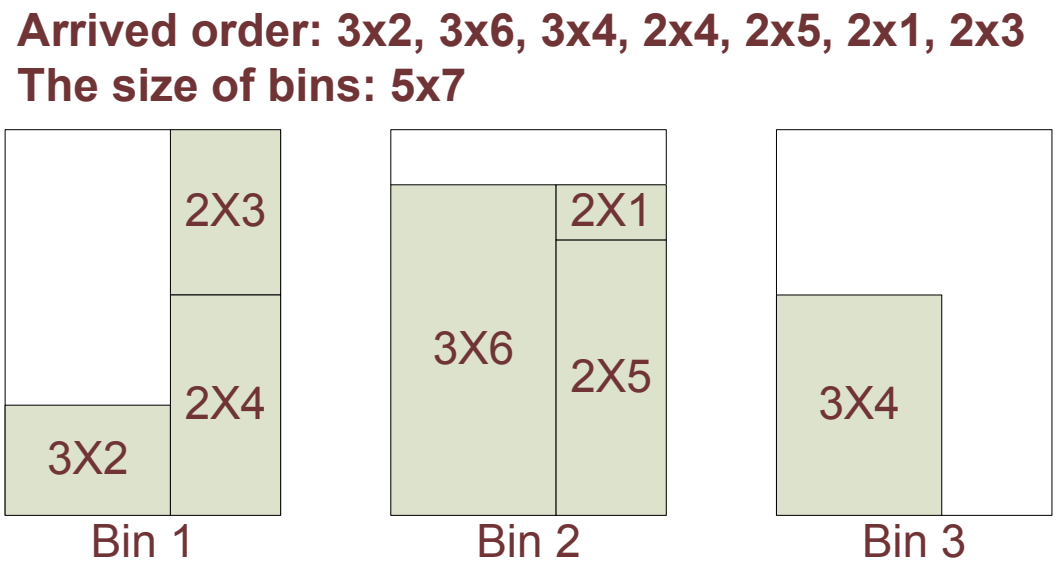

Fig. 27. Example of algorithm ESS in 2D packing.

Figure 27 shows how ESS packs the 2D instance, where ESS uses bins 1 and 2 to pack the first two items of sizes 3×2 and 3×6 respectively. When the third item of size 3×4 comes, ESS has two options. It either packs the item 2-adjacent to the 2-pile in bin 1, or creates a new 1-pile in bin 3. Note that ESS's utility function $\sum_{l=1}^{6} N^2(l)$ for the first option is: $2^2 = 4$, and for the second option is $1 + 1 + 1 = 3$. To minimize the utility function, ESS opens bin 3 for the item of size 3×4. After that, ESS packs the next two items 2×4, and 2×5 into bins 1 and 2. When item of size 2×1 comes, ESS again has three options to pack the item

1) 2-adjacent to the item 2×4 in bin 1

2) 2-adjacent to the item 2×5 in bin 2

3) opens a new 1-pile in bin 3 and makes it 1-adjacent to the item of size 3×4 in bin 3

Note that the ESS's utility function for the first option is $2^2 = 4$, for the second option is $1 + 1 = 2$, for the third option is $1 + 1 + 1 = 3$. ESS prefers to the minimal values, and uses bin 2 to pack the item of size 2×1. Regarding the last item of size 2×3, ESS makes it 2-adjacent to 2-pile of size 2×4 in bin 1.

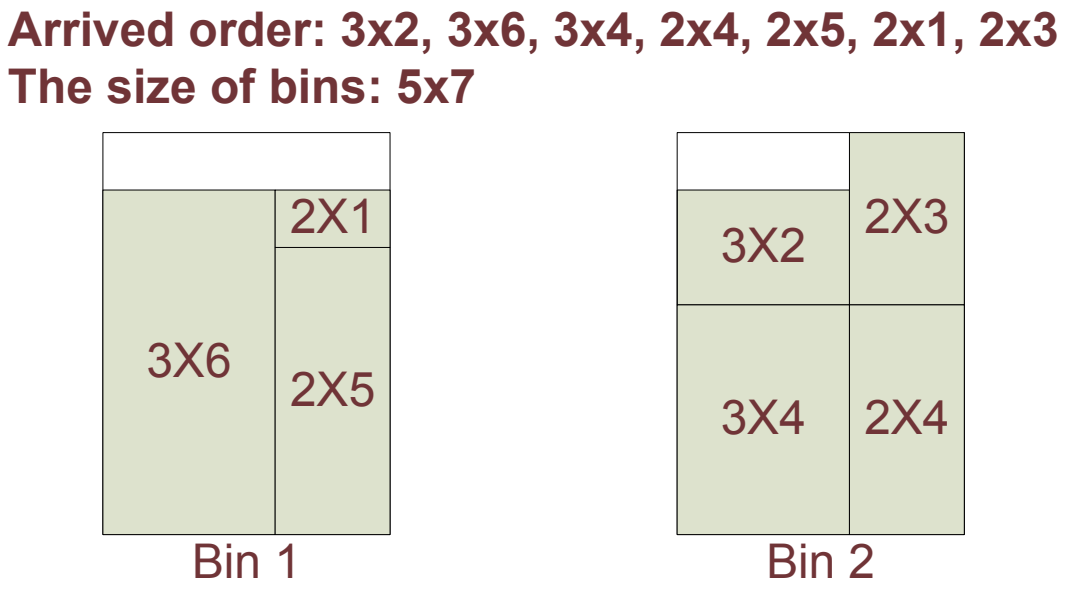

Fig. 28. Example of algorithm EFFD and EBFD in 2D packing.

Figure 28 shows how EBFD and EFFD pack the packing instance. Recall that EBFD and EFFD first packs the input items lexicographically in non-increasing order. After sorting, the arriving order of the items will be changed to:

$$3 \times 6, 3 \times 4, 3 \times 2, 2 \times 5, 2 \times 4, 2 \times 3, \text{ and } 2 \times 1. \tag{3.3}$$

After sorting the input items, EFFD and EBFD use algorithm EFF and EBF to pack the sorted item respectively. In this example, EFFD and EBFD have the same

packing solution as shown in Figure 28.

CHAPTER IV

WORST-CASE ANALYSES

A. R_{EA} vs. R_A

In this section, we will investigate the relationship of the performance ratios between framework-based algorithms EA and their 1D counterparts A. For some special packing lists, we find that the ratios R_{EA} and R_A are closely related for both offline algorithms and online algorithms. Before giving our results on the relationships, we next present some definitions and notations.

In 1D bin packing, let $L = \{L[1], L[2], \cdots, L[n]\}$ be an ordered packing list, where $L[i]$ is the size of the ith arriving item, and $n = |L|$ is the number of items in list L. We define three operations on the list: addition, repeater, and scalar multiplier. The addition operator "+" appends the second list to the end of the first list. Repeater operation $Rep(N, L)$, $N \in \mathcal{N}$, produces a new list $\sum_{i=1}^{N} L$, which essentially adds N copies of list L together. Denote by NL a scalar multiplier on list L, and it refers to repeating each item in L by N times. The definition of the above three operators are represented by the following equations, where L_1 and L_2 are two ordered lists.

$$L_1 + L_2 = \{L_1[1], L_1[2], \cdots, L_1[|L_1|], L_2[1], L_2[2], \cdots, L_2[|L_2|]\}. \tag{4.1}$$

$$Rep(N, L_1) = \sum_{i=1}^{N} L_1. \tag{4.2}$$

$$NL_1 = \sum_{i=1}^{|L_1|} Rep(N, \{L_1[i]\}). \tag{4.3}$$

For example, if $L_1 = \{1, 2, 3\}$ and $L_2 = \{4, 5, 6\}$, then,

$$L_1 + L_2 = \{1, 2, 3, 4, 5, 6\}, \tag{4.4}$$

$$Rep(2, L_1) = \{1, 2, 3, 1, 2, 3\}, \tag{4.5}$$

$$2L_1 = \{1, 1, 2, 2, 3, 3\}. \tag{4.6}$$

1. Offline Algorithms

Denote by A either *EFFD* or *EBFD*. A list L of 1D items is called (A, B)-admissible if 1) L is perfect-packable on bins of size B, and 2) A's performance ratio on packing L and NL, $\forall N \in \mathcal{N}$, are the same

$$R_A(L, B) = R_A(NL, B). \tag{4.7}$$

Since admissible list L is perfect-packable, (4.7) indicates that algorithm A will use $N \cdot A(L, B)$ bins to pack list NL.

For example, packing list L

$$Rep(6, 3), Rep(12, 2), \tag{4.8}$$

which includes 6 items of size 3, and 12 items of size 2, is a $(FFD, 7)$-admissible because 1) L could be perfectly packed into 6 bins, and 2) $R_{FFD}(NL, 7) = R_{FFD}(L, 7) = 7/6$ for any natural number N.

For the admissible packing lists, our next theorem shows that the performance ratio of EFFD and EBFD could be represented in terms of R_{FFD} and R_{BFD} in 2-dimensional packing.

Theorem 4 *Denote by A either FFD or BFD, and by EA the extension of A. Let L_1 be (A, B_1)-admissible, L_2 be (A, B_2)-admissible, and $L = |L_1| \times |L_2|$ be a list of rectangle-items*

$$L = \sum_{i=1}^{|L_1|} \sum_{j=1}^{|L_2|} \{\vec{s}_{i,j}\}, \tag{4.9}$$

where $s_{i,j} = (L_1[i], L_2[j])$ is the size of width $L_1[i]$ and of height $L_2[j]$. Then, the

performance ratio of EA on packing the list L into bins of size $\vec{B} = (B_1, B_2)$ *is:*

$$R_{EA}(L, \vec{B}) = R_A(L_1, B_1)R_A(L_2, B_2). \tag{4.10}$$

Note that Theorem 4 relates the performance ratio of 1D packing with that of 2D packing, and provides us an easy way to compute the performance ratio of EFFD and EBFD for admissible packing lists. For example, we know that list L

$$Rep(6, 3), Rep(12, 2) \tag{4.11}$$

is $(FFD, 7)$-admissible, and has performance ratio $R_{FFD}(L, 7) = 7/6$. The theorem says that for 2D packing list L'

$$Rep\left(36, (3, 3)\right), Rep\left(72, (3, 2)\right), Rep\left(72, (2, 3)\right), Rep\left(144, (2, 2)\right), \tag{4.12}$$

the performance ratio of $EFFD$ is

$$R_{EFFD}(L', (7, 7)) = R_{FFD}(L, 7)^2 = \frac{49}{36}. \tag{4.13}$$

Next, we will prove theorem 4.

PROOF. On packing L with algorithm EA, we have two kinds of waste, type-X and type-Y, as shown in Figure 29. Type-X waste is the unused rectangle-area of height B_2 and of width from the last strip in each bin to the right-most side of the bin. Type-Y waste happens in each strip of height less than B_2. In such a strip of width i, type-Y waste is the unused rectangle-area of width i and of height from the last item in the strip to the top of the bin. Denote by w_x the summation of all type-X waste, by w_{y_i} the summation of all type-Y waste in strips of width i, by $w_y = \sum_{i=1}^{B_1} w_{y_i}$ the summation of all type-Y waste.

Let $S_1 = \sum_{i=1}^{|L_1|} L_1[i]$, and $S_2 = \sum_{i=1}^{|L_2|} L_2[i]$ be the total length of items in lists L_1 and L_2 respectively. Suppose that the number of size-i, $i = 1, 2, \cdots, B_1$, items in

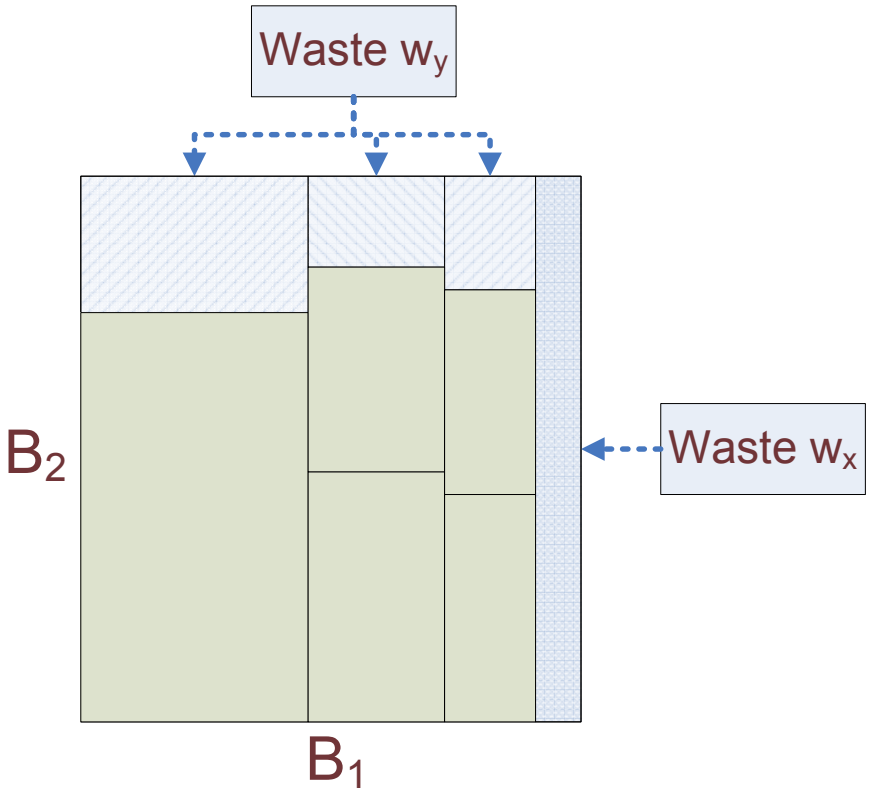

Fig. 29. The waste type w_x and w_y in EFFD(EBFD).

L_1 is n_i. Then, $\sum_{i=1}^{B_1} i \cdot n_i = S_1$. Note that while EA packs L, rectangle-items are ordered first on widths, then on heights; and that EA works like A in packing items into strips. For each width i, EA would create $A(n_i L_2) = n_i \cdot A(L_2)$ strips of width i.

Let $r_1 = R_A(L_1, B_1)$, $r_2 = R_A(L_2, B_2)$, Denote by $w_1(w_2)$ the waste of packing $L_1(L_2)$ into size-B_1(-B_2) bins with algorithm A. Since L_1 is admissible, therefore, perfect-packable, we have $OPT(L_1) = S_1/B_1$. Note that

$$r_1 = R_A(L_1, B_1) = \frac{A(L_1)}{OPT(L_1)} = \frac{(w_1 + S_1)/B_1}{S_1/B_1} = \frac{w_1 + S_1}{S_1}. \tag{4.14}$$

Therefore, $w_1 = (r_1 - 1)S_1$, and similarly, we have $w_2 = (r_2 - 1)S_2$. We can compute

the total of type-Y waste as:

$$w_y = \sum_{i=1}^{B_1} w_{y_i} = \sum_{i=1}^{B_1} i n_i w_2. \tag{4.15}$$

Now, let's compute w_x. Note that we have $n_i A(L_2)$ strips for each width i, where $i = 1, 2, \cdots, B_1$. Packing those strips into bins of size $\vec{B}$ will use the same number of bins as packing list $A(L_2)L_1$ into bins of size B_1. Then, we get type-X waste as:

$$w_x = B_2 A(L_2) w_1 = (w_2 + S_2) w_1 = w_1 w_2 + w_1 S_2. \tag{4.16}$$

Using the above equation, $w_1 = (r_1 - 1)S_1$ and $w_2 = (r_2 - 1)S_2$, we get the total waste of EA on packing L w.r.t $\vec{B}$ as:

$$\begin{aligned} w_y + w_x &= S_1(r_2 - 1)S_2 + (r_1 - 1)S_1 S_2 + (r_1 - 1)S_1(r_2 - 1)S_2 & (4.17) \\ &= (r_1 r_2 - 1)S_1 S_2, & (4.18) \end{aligned}$$

which indicates that the performance ratio of EA is $r_1 r_2$, and completes the proof. □

Note that theorem 4 targets 2D packing. Actually, we can generalize the result for multiple dimensions as the following theorem.

Theorem 5 *Let A be either FFD or BFD, and EA be the extension of A. If L_d is an (A, B_d)-admissible list, $d = 1, \cdots, D$, and*

$$L = \sum_{i_1=1}^{|L_1|} \sum_{i_2=1}^{|L_2|} \cdots \sum_{i_D}^{|L_D|} \{(L_1[i_1], L_2[i_2], \cdots, L_D[i_D])\} \tag{4.19}$$

then the performance ratio of EA on packing the list L into size-$\vec{B}$ bins is:

$$R_{EA}(L, \vec{B}) = \textstyle\prod_{d=1}^{D} R_A(L_d, B_d), \tag{4.20}$$

where $\vec{B} = (B_1, B_2, \cdots, B_D)$.

PROOF. Prove by induction.

1) It obviously holds for $D = 1$

2) We next shows that if the theorem holds for natural number D, it is also true for number $D + 1$. Precisely, we need to show that for packing list L'

$$L' = \sum_{i_1=1}^{|L_1|} \sum_{i_2=1}^{|L_2|} \cdots \sum_{i_D}^{|L_D|} \sum_{i_{D+1}}^{|L_{D+1}|} \{(L_1[i_1], L_2[i_2], \cdots, L_D[i_D], L_{D+1}[i_{D+1}])\}, \quad (4.21)$$

the performance ratio of $EFFD$ or $EBFD$ is $R_{EA}(L', \vec{B}') = \prod_{d=1}^{D+1} R_A(L_d, B_d)$, where $\vec{B}' = (B_1, \cdots, B_{D+1})$.

Denote by

$$L'' = \sum_{i_2=1}^{|L_2|} \cdots \sum_{i_{D+1}}^{|L_{D+1}|} \{(L_2[i_2], \cdots, L_{D+1}[i_{D+1}])\}, \quad (4.22)$$

a packing list of D dimensions, and by $\vec{B}'' = (B_2, \cdots, B_{D+1})$ the corresponding bins' size. With the notation, We can represent L' as $L' = \sum_{i_1=1}^{|L_1|} L_1[i_1] \times L''$. Regarding L', $SOPT$ will produce $SOPT(L'', \vec{B}'')$ 1-pile of each 1-length $L_1[i]$, where $i = 1, \cdots, |L_1|$, and will use $SOPT(L'', \vec{B}'')SOPT(L_1, B_1)$ bins. In the meantime, EA produces $EA(L'', \vec{B}'')$ 1-pile of each 1-length $L_1[i]$, and uses $EA(L'\prime, \vec{B}'')A(L_1, B_1)$ bins. Therefore,

$$\frac{EA(L', \vec{B}')}{SOPT(L', \vec{B}')} = \frac{A(L_1, B)EA(L'', \vec{B}'')}{OPT(L_1, B)SOPT(L'', \vec{B}'')}. \quad (4.23)$$

Note that $SOPT$ on packing list L'' and bins size $\vec{B}''$ is a perfect packing. Therefore, $SOPT(L'', \vec{B}'') = OPT(L'', \vec{B}'')$. Also note that

$$\frac{EA(L'', \vec{B}'')}{OPT(L'', \vec{B}'')} = \prod_{d=2}^{D+1} R_A(L_d, B_d). \quad (4.24)$$

Using (4.23) gives

$$R_{EA}(L', \vec{B}') = \frac{EA(L', \vec{B}')}{OPT(L', \vec{B}')} = \prod_{d=1}^{D+1} R_A(L_d, B_d). \tag{4.25}$$

□

2. Online Algorithms

Similarly, we can get the relations of performance ratios for EFF, EBF, FF and BF. Since these algorithms are online, we use a different definition. Let A be an online algorithm of packing a list L into identical bins of size B. In the packing problem, the items' order in L plays an important role because each item in L has to be packed immediately once arrived. L is called an A**-normalized** order if A packs each item in L into either the last open-bin or newly-opened bin. List L of 1D items is called (A, B)-admissible if 1) L is perfect-packable on bins of size B, 2) after A packs L, the waste gap of each bin is less than the minimum size in L, and 3) L is in A-**normalized** order. For example, packing list $L = \{3, 3, 5, 5\}$ is $(FF, 8)$-admissible because it meets the above three conditions.

It is easy to check that list $Rep(N, L), \forall N \in \mathcal{N}$, is also (A, B)-admissible, and $R_A(L, B) = R_A\left(Rep(N, L), B\right)$. Note that the minimum size in L is larger than any existing waste gap, and that applying algorithm A on $Rep(N, L)$ will repeat the same packing configuration for N times.

In an (A, B)-admissible list L, denote by $G_i(A, L)$, $i = 1, \cdots, A(L)$, the ordered list of items that are packed by A in the ith bin. If $L = \{3, 3, 5, 5\}$ and bins' size is 8, FF will pack 2 items of size 3 into the first bin, and the rest two items of size 5 into the second and the third bins respectively. Therefore,

$$G_1(FF, L) = \{3, 3\}, G_2(FF, L) = \{5\}, G_3(FF, L) = \{5\}. \tag{4.26}$$

Having the above notation, we will show the performance ratio relationship between the EFF(EBF) and FF(BF) as the following theorem.

Theorem 6 *Let A be either Best Fit (BF) or First Fit (FF), and EA be the extension of A under our framework. Denote by L_d, an (A, B_d)-admissible list of 1D items, where $d = 1, \cdots, D$. We construct a list L of D-dimensional items as*

$$L = \sum_{i_1=1}^{A(L_1)} \cdots \sum_{i_D=1}^{A(L_D)} \sum_{j_1=1}^{|G_{i_1}|} \cdots \sum_{j_D=1}^{|G_{i_D}|} \{ \ (G_{i_1}[j_1], G_{i_2}[j_2], \cdots, G_{i_D}[j_D]) \ \}, \tag{4.27}$$

where G_{i_d}, short for $G_{i_d}(A, L_d)$, is the list of items in the i_dth bin while A packs list L_d into size-B_d bins, and $|G_{i_d}|$ is the number of items in G_{i_d}, and $G_{i_d}[j_d]$ is the j_dth element in G_{i_d}. The performance ratio of EA on list L and size $\vec{B} = (B_1, B_2, \cdots, B_D)$ is:

$$R_{EA}(L, \vec{B}) = \prod_{d=1}^{D} R_A(L_d, B_d). \tag{4.28}$$

Proof. Prove by induction.

1. When $D = 1$, (4.28) obviously holds.

2. Suppose that (4.28) holds for D dimensions, we next show that it also holds for $D+1$ dimensions. Let L_{D+1} be (A, B_{D+1})-admissible, $B' = (B_1, B_2, \cdots, B_{D+1})$, and let

$$L' = \sum_{i_1=1}^{A(L_1)} \cdots \sum_{i_{D+1}=1}^{A(L_{D+1})} \sum_{j_1=1}^{|G_{i_1}|} \cdots \sum_{j_{D+1}=1}^{|G_{i_{D+1}}|} \left\{ \ \left(G_{i_1}[j_1], G_{i_2}[j_2], \cdots, G_{i_{D+1}}[j_{D+1}]\right) \ \right\}.$$

We need to show that

$$R_{EA}(L', \vec{B'}) = \prod_{d=1}^{D+1} R_A(L_d, B_d). \tag{4.29}$$

Note that L_d, $d = 1, \cdots, D+1$, is (A, B_d)-admissible, therefore, perfect-packable.

In optimal packing, we may reorder L' to list

$$L'' = \sum_{i_1=1}^{|L_1|} \cdots \sum_{i_{D+1}=1}^{|L_{D+1}|} \{(L_1[i_1], \cdots, L_{D+1}[i_{D+1}])\}, \tag{4.30}$$

where $(L[i_1], \cdots, L_{D+1}[i_{D+1}])$ is $(D+1)$-dimensional items. Obviously, L'' is perfect-packable, then, L' and L are also perfect-packable. Denote by $s_d = \sum_{i=1}^{|L_d|} L_d[i]$ the sum length in list L_d. We have

$$OPT(L, \vec{B}) = \prod_{d=1}^{D} \frac{s_d}{B_d}, \tag{4.31}$$

$$OPT(L', \vec{B}') = \prod_{d=1}^{D+1} \frac{s_d}{B_d} = OPT(L, \vec{B}) \frac{s_{D+1}}{B_{D+1}}. \tag{4.32}$$

Based on our assumption, we get the number of bins used by EA of packing L as:

$$EA(L, \vec{B}) = OPT(L, \vec{B}) \prod_{d=1}^{D} R_A(L_d, B_d). \tag{4.33}$$

We next evaluate $EA(L', \vec{B}')$. Expand L' into a sequence of groups:

$$\begin{array}{llll}
(G_{i_1}[j_1], G_{i_2}[j_2], & \cdots & , G_{i_D}[j_D], G_{i_{D+1}}[1]) \\
(G_{i_1}[j_1], G_{i_2}[j_2], & \cdots & , G_{i_D}[j_D], G_{i_{D+1}}[2]) \\
& \vdots & \\
(G_{i_1}[j_1], G_{i_2}[j_2], & \cdots & , G_{i_D}[j_D], G_{i_{D+1}}[|G_{i_{D+1}}|]).
\end{array}$$

Note that each of the above groups will be packed by EA into a $(D+1)$-pile of size $(G_{i_1}[j_1], G_{i_2}[j_2], \cdots, G_{i_D}[j_D])$ in the the first D dimensions. The length of the pile in dimension $D+1$ is less than or equal to B_{D+1}. and any following items after the group can NOT be put $(D+1)$-adjacently to the strip because L_{D+1} is (A, B_{D+1})-admissible, and the waste length on the strip is less than minimum size in L_{D+1}. Based on the observation, packing L' by EA is equivalent to packing $A(L_{D+1})$ lists of L by EA.

Therefore,

$$EA(L', \vec{B}') = A(L_{D+1})EA(L, \vec{B}) = \frac{s_{D+1}}{B_{D+1}} R_A(L_{D+1}, B_{D+1})EA(L, \vec{B}). \quad (4.34)$$

Using (4.32) and (4.34), we get the performance ratio of EA on packing L' in $\vec{B}'$ as:

$$R_{EA}(L', \vec{B}') = \frac{EA(L', \vec{B}')}{OPT(L', \vec{B}')} = \prod_{i=1}^{D+1} R_A(L_d, B_d), \quad (4.35)$$

which meets the requirement of (4.29) and completes the proof. □

Theorem 6 gives the performance-ratio relationship between EFF, EBF and their 1D counterparts, and offers a easy way to compute the performance ratio of EFF or EBF for some special packing lists. For example, we know that FF uses 3 bins to pack list $L_1 = \{3, 3, 5, 5\}$ into bins of size 8, and uses 3 bins to pack list $L_2 = \{4, 4, 5, 5\}$ into bins of size 9 as well. Note that optimal algorithms use 2 bins for both lists L_1 and L_2, and $R_{FF}(L_1, 8) = R_{FF}(L_2, 9) = 1.5$. Using theorem 6, we can quickly compute the performance ratio of EFF for packing list L':

$$Rep\,(4, (3, 4))\,, Rep\,(2, (3, 5))\,, Rep\,(2, (5, 4))\,, Rep\,(4, (5, 5)) \quad (4.36)$$

and bins of size $\vec{B} = (8, 9)$ as:

$$R_{EFF}(L', \vec{B}) = 1.5 \times 1.5 = 2.25. \quad (4.37)$$

B. EA vs. SOPT

Recall that SOPT are algorithms that follow strip packing rule and use minimal number of bins. Since framework-based algorithms EA also follow strip packing rule, it is natural to compare algorithms EA and SOPT. The comparison offers the following benefits:

- it helps us to better understand the framework-based algorithms and strip packing rule
- it allows us to use SOPT as a baseline, and differentiate the performance of framework-based algorithms EA
- by using the relationship between SOPT and OPT studied in the next sections, it bridges the gap between algorithms EA and OPT

Denote by $EA(L, \vec{B})$ and $SOPT(L, \vec{B})$ the number of bins used by algorithms EA and SOPT respectively with respect to packing list L and bins of size $\vec{B}$. In this section, we will use metric

$$R^{\infty}(EA, SOPT) = \sup_{\vec{B} \in \mathcal{R}^{\mathcal{D}}} \limsup_{SOPT(L,\vec{B}) \to \infty} \frac{EA(L, \vec{B})}{SOPT(L, \vec{B})} \tag{4.38}$$

to estimate the performance of framework algorithms ENF, EFF, EBF, EFFD, EBFD, and ESS. As in 1D packing, we will estimate the lower bounds and upper bounds of $R^{\infty}(EA, SOPT)$.

1. Lower Bounds

To get the lower bounds, we essentially need to find some packing instance that has high value for metric $R^{\infty}(EA, SOPT)$. We next try to find the packing examples for algorithms ENF, EFF, EBF, EFFD, EBFD, and ESS.

a. ENF

Recall that in 1D packing, packing instance $(B/2, 1, B/2, 1, \cdots,)$ gives the worst packing instance for Next Fit, where B is the bins' size. To work on the ENF for

ratio $R^{\infty}(EA, SOPT)$, we consider the following packing example for 2D packing

$$L = \left\{ \left(\frac{B_1}{2}, \frac{B_2}{2} \right), \left(\frac{B_1}{2}, 1 \right), \left(1, \frac{B_2}{2} \right), (1,1) \right\}, \tag{4.39}$$

where B_1 and B_2 are the bins' width and height. Note that ENF packs the first two items into a 2-pile. The last two items will close the first 2-pile, and pack into a new 2-pile in the same bin. If we repeat the above packing set for $4NB_1B_2$ times, ENF will pack them into $4NB_1B_2$ bins.

SOPT should pack the $4NB_1B_2$ sets of items in the way, where uses $NB_1B_2 + 2NB_1 + 2NB_2 + 4N$ bins. Therefore,

$$R^{\infty}(ENF, SOPT) \geq \sup_{B_1 \in \mathcal{R}, B_2 \in \mathcal{R}} \lim_{N \to \infty} \frac{4NB_1B_2}{NB_1B_2 + 2NB_1 + 2NB_2 + 4N} = 4. \tag{4.40}$$

Using the similar packing pattern, we may extend the packing example to D-dimensional hyperbox packing as follows. Denote by

$$L = S_1 \times S_2, \cdots, \times S_D \tag{4.41}$$

the cross product of set S_i, where

$$S_i = \left\{ \frac{B_i}{2}, 1 \right\}, \tag{4.42}$$

is a set of two elements, and B_i is bins' edge length in the ith dimension. In three dimensions, L is in the form of:

$$\left\{ \left(\frac{B_1}{2}, \frac{B_2}{2}, \frac{B_3}{2} \right), \left(\frac{B_1}{2}, \frac{B_2}{2}, 1 \right), \cdots, (1,1,1) \right\}. \tag{4.43}$$

Let the packing items be $N2^D \prod_{i=1}^{D} B_i$ copies of set L. ENF needs a new bin to pack each copy of the set, and close all piles in the bin when the next copy comes.

Therefore, ENF uses

$$ENF(L) = N2^D \prod_{i=1}^{D} B_i, \tag{4.44}$$

bins for the items. Also note that input items could be perfectly packed by SOPT, and that the summation of items area is:

$$N2^D \prod_{i=1}^{D} B_i \prod_{i=1}^{D} \left(\frac{B_i}{2} + 1 \right). \tag{4.45}$$

Therefore, the number of bins used by SOPT is:

$$SOPT(L) = N \prod_{i=1}^{D} (B_i + 2). \tag{4.46}$$

Combining (4.44) and (4.46), we get:

$$R^{\infty}(ENF, SOPT) = \sup_{B_i \in \mathcal{R}} \limsup_{N \to \infty} \frac{ENF(L)}{SOPT(L)} \geq 2^D. \tag{4.47}$$

b. EFF and EBF

Let $t_1 = 1, t_{i+1} = t_i(t_i + 1)$ for $i \in \mathcal{N}$, and $B \geq t_k * (k-1) + k$ where $k \geq 1$. Recall that for packing list L

$$Rep\left(t_k n, B - (k-1)\right), \sum_{i=k-1}^{1} Rep\left(t_k n, \frac{t_k B}{t_i + 1} + 1\right) \tag{4.48}$$

and bins of size $t_k B$, FF and BF have performance ratios

$$R_{FF}(L, t_k B) = R_{BF}(L, t_k B) = \sum_{i=1}^{k} \frac{1}{t_i}. \tag{4.49}$$

Notice that L is both $(FF, t_k B)$-admissible and $(BF, t_k B)$-admissible. We can construct a packing list L' as:

$$\sum_{i_1=1}^{A(L)} \cdots \sum_{i_D=1}^{A(L)} \sum_{j_1=1}^{|G_{i_1}|} \cdots \sum_{j_D=1}^{|G_{i_D}|} \{ \ (G_{i_1}[j_1], G_{i_2}[j_2], \cdots, G_{i_D}[j_D]) \ \}, \tag{4.50}$$

where G_i is the list of items packed by FF by BF into ith bin of size t_kB. Using theorem 6, we conclude that the performance ratio of EFF and EBF on packing list L' on bins of size $\vec{B} = (t_kB, \cdots, t_kB)$ is:

$$R_{EFF}(L', \vec{B}) = R_{EBF}(L', \vec{B}) = \left(\sum_{i=1}^{k} \frac{1}{t_i}\right)^D. \tag{4.51}$$

Since L' is perfect-packable by SOPT on bins of size $\vec{B}$, SOPT and OPT should use the same number of bins regarding L'. Therefore,

$$R^{\infty}(EFF, SOPT) \geq \sup_{k \in \mathcal{N}} \lim_{n \to \infty} \frac{EFF(L', \vec{B})}{SOPT(L', \vec{B})} = \left(\sum_{i=1}^{\infty} \frac{1}{t_i}\right)^D. \tag{4.52}$$

Using the similar technique, we may show that the lower bound for EBF as:

$$R^{\infty}(EBF, SOPT) \geq \left(\sum_{i=1}^{\infty} \frac{1}{t_i}\right)^D. \tag{4.53}$$

c. ESS

As shown in Figure 14, SS has performance ratio $R^{\infty}_{SS} = 2B/(B+1)$ if bins' size is $2B+1$, and all to-be-packed items are of size 2. Next, we will show a lower bound of ESS on 2D packing.

In the example, bins are squares and of size $(2B+1, 2B+1)$ and B is an odd number. Items are squares as well, and of size $(2, 2)$. Note that ESS will use NB bins to pack $NB(B+1)^2/4$ such items, and there are $N(B+1)/2$ 2-piles with 2-length $l = 2, 4, \cdots, 2B$. Figure 30 shows such an example for $B = 3$ and $N = 1$. Note that each bin could at most hold B^2 items. Therefore, for list L consisting of $NB(B+1)^2/4$ items of size $(2, 2)$, optimal algorithms use

$$SOPT(L) = OPT(L) = \left\lceil \frac{NB(B+1)^2/4}{B^2} \right\rceil \tag{4.54}$$

bins, and ESS uses NB bins. Therefore,

$$R^{\infty}(ESS, SOPT)(B) \geq \lim_{N\to\infty} NB/\left\lceil \frac{NB(B+1)^2/4}{B^2} \right\rceil \tag{4.55}$$

$$\geq \lim_{N\to\infty} NB/\left(\frac{N(B+1)^2/4}{B} + 1 \right) = \frac{4B^2}{(B+1)^2}. \tag{4.56}$$

and

$$R^{\infty}(ESS, SOPT) = \sup_{B\in\mathcal{R}} R^{\infty}_{ESS}(B) \geq 4. \tag{4.57}$$

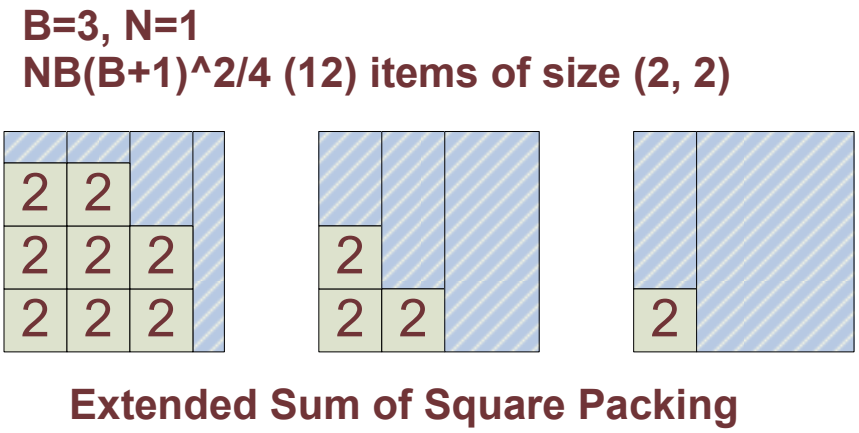

Fig. 30. 12 items of size $(2, 2)$ packed into 3 bins of size $(7, 7)$.

The 2D packing example could be extended to D-dimensional packing, where bins are hypercube with edge-length $2B + 1$, and to-be-packed items are hypercube as well, and their edge-lengths are 2. Using similar analyses, we could obtain the lower bound of performance ratio for ESS as:

$$R^{\infty}(ESS, SOPT) = \sup_{B\in\mathcal{R}} R^{\infty}_{ESS}(B) \geq 2^D. \tag{4.58}$$

d. EFFD and EBFD

As shown in Figure 13 in chapter II, a list L of $6N$ items of sizes $B/2+1, B/4+2, B/4+1$, and $12N$ items of size $B/4-2$ could be perfectly packed into $9N$ bins of size B, while EFF and EBF need $11N$ bins to pack the list. Also note that list L is (FFD, B)-admissible and (BFD, B)-admissible. We can construct a new D-dimensional packing list L' as:

$$L' = \sum_{i_1=1}^{|L|} \sum_{i_2=1}^{|L|} \cdots \sum_{i_D}^{|L|} \{(L[i_1], L[i_2], \cdots, L[i_D])\}. \tag{4.59}$$

And theorem 5 indicates that the performance ratios of EFFD and EBFD against the list are:

$$R_{EFFD}(L', \vec{B}) = R_{EBFD}(L', \vec{B}) = \left(\frac{11}{9}\right)^D, \tag{4.60}$$

where $\vec{B} = (B, \cdots, B)$. Also note that for packing list L' and bins of size $\vec{B}$, SOPT has perfect packing, and uses the same number of bins as OPT. Therefore,

$$R^{\infty}(EFFD, SOPT) \geq \lim_{N\to\infty} \frac{EFFD(L', \vec{B})}{SOPT(L', \vec{B})} = \left(\frac{11}{9}\right)^D. \tag{4.61}$$

2. Upper Bounds

Next, we investigate the upper bounds of performance ratio of extended framework algorithms comparing to strip optimal packing algorithms. The summary results of the upper bounds are shown in the following theorem.

Theorem 7 *On the problem of packing a list of D-hyperbox-items into D-hyperbox-bins that have edge length B_d in dimension d, we have:*

$$EFFD(L) \leq \left(\frac{11}{9}\right)^D SOPT(L) + c, \tag{4.62}$$

$$EBFD(L) \leq \left(\frac{11}{9}\right)^D SOPT(L) + c, \tag{4.63}$$

$$EFF(L) \leq 1.7^D SOPT(L) + c, \tag{4.64}$$

$$EBF(L) \leq 1.7^D SOPT(L) + c, \tag{4.65}$$

$$ESS(L) \leq (25/9)^D SOPT(L) + c, \tag{4.66}$$

$$ENF(L) \leq 2^D SOPT(L) + c, \tag{4.67}$$

where $EA(L)$ and $SOPT(L)$ are the number of bins used by algorithms EA and $SOPT$ respectively on list L, and c is a constant independent of L.

Recall that the upper bound of performance ratio is 11/9 for FFD and BFD, 1.7 for FF and BF, and 25/9 for SS. The above theorem shows that the upper bounds of framework algorithms are closely related to their 1D counterparts. Before proving the above theorem, we will need the following lemma.

On the problem of packing items into bins of size B, denote by $L[i]$ the number of items of size i, by $L = \{L[i]|i = 1, \cdots, B\}$ the ordered list of all to-be-packed items, where the ith element represents the number of items of size i. Recall that a list of items is perfect-packable if they can be packed into bins without any waste space. For 1D perfect-packable list, we have the following lemma.

Lemma 8 *If L is a list of 1D items, and is perfectly-packable by optimal algorithm OPT, then*

$$OPT(\lceil \alpha L \rceil) \leq \alpha OPT(L) + K_B, \tag{4.68}$$

where $OPT(L)$ the number of bins used by optimal algorithms to pack the list L, $\lceil \alpha L \rceil = \{\lceil \alpha L[i] \rceil | i = 1, \cdots, B\}$, $\alpha \in \mathcal{R}$, is another list, and K_B is the number of perfect-packing configurations w.r.t the bins of size B.

As shown in [64], K_B could be estimated via the following formula:

$$K_B \sim \frac{1}{4B\sqrt{3}} e^{\pi\sqrt{2B/3}}. \tag{4.69}$$

Next, we will prove lemma 8 as follows.

PROOF. Let $C = \{C_{ij}\}, i = 1, \cdots, B, j = 1, \cdots, K_B$ be the perfect packing matrix, where each element C_{ij} represents the number of size-i items in configuration C_j. Denote by $N_j, j = 1, \cdots, K_B$, the number of configurations C_j to perfectly pack list L. Then we have:

$$\sum_{j=1}^{K_B} C_{ij} N_j = L[i]. \tag{4.70}$$

Consider a perfect packing with $\lceil \alpha N_j \rceil$ configurations of C_j, the number of size-i items is:

$$\sum_{j=1}^{K_B} C_{ij} \lceil \alpha N_j \rceil \geq \lceil \alpha L[i] \rceil. \tag{4.71}$$

Also note that the number of bins used in the packing is:

$$\sum_{j=1}^{K_B} \lceil \alpha N_j \rceil \leq \sum_{j=1}^{K_B} \alpha N_j + K_B = \alpha OPT(L) + K_B. \tag{4.72}$$

Therefore, the list of items $\lceil \alpha L \rceil$ could be packed using at most $\alpha OPT(L) + K_B$ bins. □

Next, we will prove the first inequality in theorem 7, which is $EFFD(L) \leq (11/9)^D SOPT(L) + c$.

PROOF. Note that in strip-based packing a couple of dth-adjacent, $d = 1, \cdots, D$, items could be successively touched together and form a dth-pile, and several dth-piles could be further form a $(d-1)$th-pile, and so on. For all dth-piles in this proof, we will only focus on their sizes in the first $d-1$ dimensions.

Let L_d be an ordered list of $(d+1)$th piles in which only the edge lengths in the first d dimensions are considered. Precisely,

$$L_d = \{L[l_1, \cdots, l_d] \mid i \in [1, d], l_i \in [1, B_i]\}, \tag{4.73}$$

where the $(l_1, \cdots, l_d)$th elements $L[l_1, \cdots, l_d]$ represents the number of $(d+1)$th-piles that have edge length l_i, for $i = 1, \cdots, d$. Let I_d be a unit list of $(d+1)$th-piles, where there exists exactly one item for each size $(l_1, \cdots, l_d)$. For any positive constant α, we define

$$\lceil \alpha L_d \rceil = \{\lceil \alpha L[l_1, \cdots, l_d] \rceil \mid i \in [1, d], l_i \in [1, B_i]\}, \tag{4.74}$$

Denote by $L1_d + L2_d$ the addition of $L1_d$ and $L2_d$. Particularly,

$$L1_d + L2_d = \{L1[s_1, \cdots, s_d] + L2[s_1, \cdots, s_d] \mid i \in [1, d], s_i \in [1, B_i]\}. \tag{4.75}$$

Let A_d be a strip-based packing algorithm that packs items in dimension d, and denote by $A_d < L_d >$ the list of dth-piles generated by A_d. Let $\mathcal{L}_D = \mathcal{L}''_D = L$, where L is the input list of D-hyperbox-items as we defined previously. Also define:

$$\mathcal{L}_{d-1} = EFFD_d < \mathcal{L}_d >, \tag{4.76}$$

$$\mathcal{L}'_{d-1} = SOPT_d < \mathcal{L}_d >, \tag{4.77}$$

$$\mathcal{L}''_{d-1} = SOPT_d < \mathcal{L}''_d >, \tag{4.78}$$

We next show that:

$$\mathcal{L}_d \leq \left\lceil \left(\frac{11}{9}\right)^{D-d} \mathcal{L}''_d + F(d) I_d \right\rceil, \tag{4.79}$$

where $d = 0, \cdots, D$, and

$$F(d-1) = \frac{11}{9}\left(K_{B_d} + \left\lceil \frac{B_d - 1}{2} \right\rceil + 1\right)(F(d)+1) + 1, \tag{4.80}$$

with initial value $F(D) = 0$, where K_{B_d} is the number of perfect-packing configurations with bin-size B_d. Note that (4.79) will give the result of the theorem when $d = 0$. Prove (4.79) by induction.

1. When $d = D$, we have $\mathcal{L}_D = \mathcal{L}''_D = L$ and (4.79) obviously hold with $F(D) = 0$.

2. Suppose (4.79) hold for d, and we need to show that it also hold for $d-1$.

Note that

$$\begin{aligned}
\mathcal{L}'_{d-1} &= SOPT_d < \mathcal{L}_d > & (4.81)\\
&\leq SOPT_d \left\langle \left\lceil \left(\frac{11}{9}\right)^{D-d} \mathcal{L}''_d + F(d) I_d \right\rceil \right\rangle & (4.82)\\
&\leq SOPT_d \left\langle \left\lceil \left(\frac{11}{9}\right)^{D-d} \mathcal{L}''_d \right\rceil \right\rangle + SOPT_d \left\langle \lceil F(d) I_d \rceil \right\rangle . & (4.83)
\end{aligned}$$

If B_d is odd, items in I_d could be packed in the dth dimensions with sizes $(1, B_d - 1), (2, B_d - 2), \cdots ((B_d - 1)/2, (B_d - 1)/2), B_d$, and $(B_d - 1)/2 + 1$ dth-piles are generated. Similarly, we will get $\lceil (B_d - 1)/2 \rceil + 1$ dth-piles when B_d is even. Hence, $SOPT_d$ at most creates $\lceil (B_d - 1)/2 \rceil + 1$ dth-pile. Using lemma 8 on (4.83), we get:

$$\mathcal{L}'_{d-1} \leq \left(\frac{11}{9}\right)^{D-d} \mathcal{L}''_{d-1} + K_{B_d} I_{d-1} + (F(d)+1)\left(\left\lceil \frac{B_d - 1}{2} \right\rceil + 1\right) I_{d-1}. \tag{4.84}$$

Using (4.84) and the result on [79], we have:

$$\begin{aligned}
\mathcal{L}_{d-1} &= EFFD_d < \mathcal{L}_d > & (4.85)\\
&\leq \left\lceil \frac{11}{9} \mathcal{L}'_{d-1} + I_{d-1} \right\rceil & (4.86)\\
&\leq \left\lceil \left(\frac{11}{9}\right)^{D-d+1} \mathcal{L}''_{d-1} + F(d-1) I_{d-1} \right\rceil . & (4.87)
\end{aligned}$$

which yields

$$EFFD(L) \leq \left(\frac{11}{9}\right)^D SOPT(L) + c, \tag{4.88}$$

with $c = F(0) + 1$. $\square$

Using the similar proving technique, and that the upper bound for BFD is 11/9, for FF and BF is 1.7, and for SS is 25/9, we can easily prove the remaining inequalities in theorem 7.

C. EA vs. OPT

In this section, we will estimate the performance of framework-based algorithms by using optimal algorithm OPT as a base line. Precisely, we will estimate the following performance ratio,

$$R_{EA}^{\infty} = \sup_{B \in \mathcal{R}} \limsup_{OPT(L) \to \infty} \frac{EA(L, \vec{B})}{OPT(L, \vec{B})}, \tag{4.89}$$

where EA is framework-based algorithms.

Note that in discrete packing, the edge length of hyperbox bins could be either bounded or unbounded. We next study how these two cases affect the performance ratio R_{EA}^{∞} of framework based algorithms.

1. Unbounded Edge Length

We find that when the edge lengths of hyperbox bins are unbounded, the performance ratios of framework based algorithms may go to infinity as well. Next theorem shows why it happens.

Theorem 9 *In a D-hyperbox packing with edge length B_d, $d = 1, \cdots, D$, in dimension d, algorithms EFF, EBF, EFFD, and EBFD have unbounded asymptotic worst-*

case ratio if edge lengths B_d are unbounded.

PROOF. Let's first consider a 2D bin-packing with edge length B_1 and B_2 in dimensions 1 and 2. Assume that $B_1 \geq B_2$, let L be a list of items of height 1 and of widths $1, B_1 - 1, 2, B_1 - 2, \cdots, B_2, B_1 - B_2$.

Under the above packing instance, we have $OPT(L) = 1$ and $EFF(L) = B_2$. Therefore,

$$R_{EFF}(L) = \frac{EFF(L)}{OPT(L)} = B_2. \tag{4.90}$$

If B_2 is unbounded, then $R^{\infty}_{EFF} = B_2 \to \infty$.

Note that 2D bin packing is a special case of D-hyperbox packing. Therefore, the asymptotic worst-case ratio of D-hyperbox packing is larger that of 2D bin packing, and is also infinite when bin's edge-lengths are unbounded.

Usinsg similar proving technique, we may show that $EBF, EBFD$ and $EFFD$ have unbounded asymptotic worst-case ratios as well. □

The above theorem shows that the worst-case ratios of framework based algorithms are related to hyperbox bins edge lengths, and that when the edge lengths go to infinity, the ratios will be unbounded as well. To better understand how the bins sizes affect the ratios, we study the performance ratio relationships for different bins sizes, and find the following result.

Theorem 10 *In a discrete D-hyperbox packing with bins of size $\vec{B} = (B_1, \cdots, B_D)$, the worst-case ratio of framework-based algorithms EA on the size-$(2\vec{B} + \vec{1})$ bins, which has edge-length $2B_d + 1$ in each dimension d, is:*

$$R_{EA}(2\vec{B} + \vec{1}) \geq R_{EA}(\vec{B}) + \frac{1}{2B_D + 1}. \tag{4.91}$$

PROOF. Denote by L a list that makes algorithm EA to get worst-case ratio while packing items into bins of size $\vec{B}$. Let $L' = \sum_{i=1}^{|L|}\{2L[i]\}$ be a copied list of L that doubles the size of each item in L. Obviously, packing L into $\vec{B}$ and packing L' into $(2\vec{B} + \vec{1})$ will use EA the same number of bins.

Without loss of generality, we assume that the optimal packing OPT on L and $\vec{B}$ has no waste. Applying the OPT on items instance L' and bins of size $(2\vec{B} + 1)$ will leave each bin identical waste that is of length 1 in each dimension. Note that the waste in each bin can hold one more item of size $\hat{L}' = (2B_1 + 1, 2B_2 + 1, \cdots, 2B_{D-1} + 1, 1)$. In the problem of packing $L' + \hat{L}'$, optimal algorithm will still use $OPT(L)$ bins. However, EA will need to open $OPT(L)/(2B_D + 1)$ more bins for item list $\hat{L}'$ because EA is a strip-based algorithm, and does NOT put items in L' and any one in $\hat{L}'$ into the same bin. Therefore, the ratio of EA on packing list $L' + \hat{L}'$ is:

$$R_{EA}(L' + \hat{L}') \geq \frac{EA(L) + \frac{OPT(L)}{2B_D+1}}{OPT(L)} = R_{EA}(L, \vec{B}) + \frac{1}{2B_D + 1}. \tag{4.92}$$

Therefore, the worst-case ratio of EA w.r.t size-$(2\vec{B}+\vec{1})$ bins is at least $R_{EA}(\vec{B}) + \frac{1}{2B_D+1}$. □

The above theorem shows that with the bins' size increase, the worst-case ratios of algorithms EA will monotonically increase. To avoid unbounded asymptotic worst-case ratio, we will limit the edge length to be bounded to better estimate R^{∞}_{EA} for different framework-based algorithms EA.

For most framework-based algorithms, it is very hard to compute the exact values of R^{∞}_{EA}. Instead, we estimate the range of the metric from two directions: lower bounds and upper bounds, while keeping the edge lengths bounded.

2. Lower Bounds

For certain packing instance L and bins of size $\vec{B}$, optimal algorithms OPT should not use more bins than strip optimal algorithms SOPT, therefore, $OPT(L, \vec{B}) \leq SOPT(L, \vec{B})$. Also recall that

$$R^{\infty}_{EA} = \sup_{\vec{B} \in \mathcal{R}^{\mathcal{D}}} \limsup_{OPT(L) \to \infty} \frac{EA(L, \vec{B})}{OPT(L, \vec{B})}, \tag{4.93}$$

and

$$R^{\infty}_{EA}(EA, SOPT) = \sup_{B \in \mathcal{R}^{\mathcal{D}}} \limsup_{SOPT(L) \to \infty} \frac{EA(L, \vec{B})}{SOPT(L, \vec{B})}. \tag{4.94}$$

Therefore, the lower bounds of $R^{\infty}(EA, SOPT)$ are also the lower bounds of R^{∞}_{EA} for each framework-based algorithm EA. Using the results from the previous section which states the lower bounds of SOPT vs. EA, we have

$$R^{\infty}_{ENF} \geq 2^D, \tag{4.95}$$

$$R^{\infty}_{EFF} \geq \left(\sum_{i=1}^{\infty} \frac{1}{t_i} \right)^D, \tag{4.96}$$

$$R^{\infty}_{EBF} \geq \left(\sum_{i=1}^{\infty} \frac{1}{t_i} \right)^D, \tag{4.97}$$

$$R^{\infty}_{ESS} \geq 2^D, \tag{4.98}$$

$$R^{\infty}_{EFFD} \geq \left(\frac{11}{9} \right)^D, \tag{4.99}$$

$$R^{\infty}_{EBFD} \geq \left(\frac{11}{9} \right)^D. \tag{4.100}$$

3. Upper Bounds

For framework-based algorithms EFFD, EBFD, EFF, EBF, and ENF, all non-last d-piles should be at least half full. If the edge lengths of the bins' are bounded, then it is easy to conclude that 2^D is one upper bound for R^{∞}_{EA}, where D is the number

of bins' dimensions. Note that the upper bound is loose, and we next study tighter upper bounds for framework-based algorithms.

a. Upper Bound for R^{∞}_{EFFD} and R^{∞}_{EBFD}

Recall that [21] showed the relationship between OPT and SOPT in 2D packing, and [23] showed the extended results in general D-dimensional hyperbox packing as follows:

$$R^{\infty}(SOPT, OPT) = \sup_{\vec{B}\in\mathcal{R}^{\mathcal{D}}} \lim_{OPT(L,\vec{B})\to\infty} \frac{SOPT(L,\vec{B})}{OPT(L,\vec{B})} \tag{4.101}$$

$$= \left(\sum_{i=1}^{\infty} \frac{1}{t_i}\right)^{D-1} < 1.7^{D-1}, \tag{4.102}$$

where $t_1 = 1, t_{i+1} = t_i(t_i + 1)$. Also note that,

$$R^{\infty}_{EA} = \sup_{\vec{B}\in\mathcal{R}^{\mathcal{D}}} \limsup_{OPT(L)\to\infty} \frac{EA(L,\vec{B})}{OPT(L,\vec{B})} \tag{4.103}$$

$$= \sup_{\vec{B}\in\mathcal{R}^{\mathcal{D}}} \limsup_{OPT(L)\to\infty} \frac{EA(L,\vec{B})}{SOPT(L,\vec{B})} \frac{SOPT(L,\vec{B})}{OPT(L,\vec{B})} \tag{4.104}$$

$$\leq R^{\infty}(EA, SOPT) R^{\infty}(SOPT, OPT). \tag{4.105}$$

Using (4.102), and that $R^{\infty}(EFFD, SOPT) \leq (11/9)^D$ in D-dimensional hyperbox packing, we get the upper bound of EFFD as:

$$R^{\infty}_{EFFD} \leq \left(\sum_{i=1}^{\infty} \frac{1}{t_i}\right)^{D-1} * \left(\frac{11}{9}\right)^{D} < \frac{2.078^D}{1.7}. \tag{4.106}$$

Similarly, the corresponding upper bound for EBFD in D-dimensional hyperbox packing is:

$$R^{\infty}_{EBFD} \leq \left(\sum_{i=1}^{\infty} \frac{1}{t_i}\right)^{D-1} * \left(\frac{11}{9}\right)^{D} < \frac{2.078^D}{1.7}. \tag{4.107}$$

Note that the upper bound of $R^{\infty}(EFF, SOPT)$ and $R^{\infty}(EFF, SOPT)$ is 1.7^D,

and if we use the similar reasoning for EFF, and EBF, the upper bound we can get for D-dimensional packing is:

$$R_{EFF}^{\infty} \leq \left(\sum_{i=1}^{\infty} \frac{1}{t_i}\right)^{D-1} * 1.7^D, \tag{4.108}$$

$$R_{EBF}^{\infty} \leq \left(\sum_{i=1}^{\infty} \frac{1}{t_i}\right)^{D-1} * 1.7^D, \tag{4.109}$$

which is even looser then $2^D = 4$. Therefore, we need to use some alternative approach to investigate the upper bound of R_{EFF}^{∞} and R_{EBF}^{∞}.

b. Upper Bound for R_{EFF}^{∞} and R_{EBF}^{∞}

In this sub-section, we are going to study the upper bound of R_{EFF}^{∞} and R_{EBF}^{∞} in 2D packing. Before walking through the proof, we first denote the weighting function $f(x)$ by:

$$f(x) = \begin{cases} \frac{6}{5B}x & x \in [0, \frac{B}{6}] \\ \frac{9}{5B}x - \frac{1}{10} & x \in [\frac{B}{6}, \frac{B}{3}] \\ \frac{6}{5B}x + \frac{1}{10} & x \in [\frac{B}{3}, \frac{B}{2}] \\ 1 & x \in (\frac{B}{2}, B] \end{cases}, \tag{4.110}$$

where x takes values in range $1, \cdots, B$. Based on the definition, we have:

$$f(x) \leq \begin{cases} \frac{6}{5B}x & x \in [0, \frac{B}{6}] \\ \frac{9}{5B}x - \frac{1}{10} & x \in [\frac{B}{6}, \frac{B}{2}] \end{cases} \tag{4.111}$$

Suppose x_d, $d = 1, \cdots, D$, takes discrete value $1, \cdots, B_d$, and we also denote multi-dimensional weighting function by $w(x_1, x_2, \cdots, x_D) = \prod_{d=1}^{D} f(x_d)$. With the above notation, we have the following corollary for 1D discrete bin packing.

Corollary 11 *For a list of discrete-size items $L = \{a_1, \cdots, a_n\}$, where $a_i \in [1, B]$,*

and bins of size B, the number of bins used by First Fit (FF) is

$$FF(L) \leq \mathcal{W} + 2, \tag{4.112}$$

$$BF(L) \leq \mathcal{W} + 2, \tag{4.113}$$

where $\mathcal{W} = \sum_{i=1}^{n} f(a_i)$ is the summation of weighting function $f(x)$ over all items a_i.

PROOF. Packing a list of items $L = (a_1, \cdots, a_n)$ into bins of size B is equivalent to packing items $L' = (a_1/B, \cdots, a_n/B)$ into unit bins. The result immediately follows by using the result in claim 2.2.4 in [53]. □

Now, we will show the relationship between the weighting function and 2D framework algorithm EFF.

Theorem 12 *Given a list of 2D items $L = \{(x_1, y_1), \cdots, (x_n, y_n)\}$, we have $\mathcal{W} \geq EFF(L) - 2B_1 - 2$, and $\mathcal{W} \geq BFF(L) - 2B_1 - 2$, where $x_i = 1, \cdots, B_1$ and $y_i = 1, \cdots, B_2$, and $\mathcal{W} = \sum_{i=1}^{n} w(x_i, y_i)$.*

PROOF. Recall that EFF for 2D packing just repeatedly applies FF in x dimension and y dimension respectively. Therefore,

$$\mathcal{W} = \sum_{i=1}^{n} w(x_i, y_i) = \sum_{i=1}^{n} w(x_i) w(y_i) \tag{4.114}$$

$$= \sum_{s=1}^{B_1} w(s) \sum_{i=1, x_i = s}^{n} w(y_i). \tag{4.115}$$

Note that $\sum_{i=1, x_i=s}^{n} w(y_i)$ is the total weight in y dimension over all items with width s. Using corollary 11, we get

$$\sum_{i=1, x_i=s}^{n} w(y_i) \geq FF(L_s) - 2, \tag{4.116}$$

where $FF(L_s)$ is the number of strips of width s generated by EFF. Using (4.115)

and (4.116), we have:

$$\mathcal{W} \geq \sum_{s=1}^{B_1} w(s)(FF(L_s) - 2) \tag{4.117}$$

$$= \sum_{s=1}^{B_1} w(s)FF(L_s) - 2\sum_{s=1}^{B_1} w(s). \tag{4.118}$$

Note that $\sum_{s=1}^{B_1} w(s)FF(L_s)$ is the total weight of all strips in x dimension, and it should be $\geq EFF(L) - 2$. Also note that $w(s) \leq 1$ for any $s \in [1, B_1]$. Therefore,

$$\mathcal{W} \geq EFF(L) - 2B_1 - 2. \tag{4.119}$$

Same reasoning gives us: $\mathcal{W} \geq EBF(L) - 2B_1 - 2$. □

Next, we relate the above weighting function and 2D optimal packing. Before we go to the relationship, let's first look at the following lemma.

Lemma 13 *Let a list of 1D items $L = \{a_1, \cdots, a_m\}$ be packed into a bin of size B, where $a_i \leq B/2$ for each $i = 1, \cdots, m$, and $V = \sum_{i=1}^{m} a_i > B/2$. Then, resizing the list of items into two items with size $\lceil (B+1)/2 \rceil$ and $V - \lceil (B+1)/2 \rceil$ will increase the weight if $B > 18$. Precisely:*

$$\sum_{i=1}^{m} f(a_i) \leq f\left(\left\lceil \frac{B+1}{2} \right\rceil\right) + f\left(V - \left\lceil \frac{B+1}{2} \right\rceil\right). \tag{4.120}$$

Proof. We separate items a_i into two set I and II, where I has size $(0, B/6]$, and II has size $(B/6, B/2]$. Using (4.111), we have:

$$\sum_{i=1}^{m} f(a_i) \leq \sum_{a_i \in I} \frac{6a_i}{5B} + \sum_{a_i \in II} \left(\frac{9a_i}{5B} - \frac{1}{10}\right) \tag{4.121}$$

Next, we will estimate

$$\delta = f\left(\left\lceil \frac{B+1}{2} \right\rceil\right) + f\left(V - \left\lceil \frac{B+1}{2} \right\rceil\right) - \sum_{i=1}^{m} f(a_i), \tag{4.122}$$

and show that it is positive in the following three cases.

case 1: $V - \lceil (B+1)/2 \rceil \in [0, B/6]$. Using (4.121) yields:

$$\begin{aligned} \delta &\geq 1 + \frac{6}{5B}\left(V - \left\lceil \frac{B+1}{2} \right\rceil\right) - \sum_{a_i \in I} \frac{6a_i}{5B} - \sum_{a_i \in II} \left(\frac{9a_i}{5B} - \frac{1}{10}\right) && (4.123) \\ &\geq 1 - \frac{3}{5B}\sum_{a_i \in II} a_i - \frac{6}{5B}\left\lceil \frac{B+1}{2} \right\rceil + \frac{1}{10}\sum_{a_i \in II} 1. && (4.124) \end{aligned}$$

Let's assume that set II is non empty. Since $\sum_{a_i \in II} a_i \leq \lceil (B+1)/2 \rceil + B/6$, (4.124) is greater than or equal to:

$$\begin{aligned} &\geq 1 - \frac{3}{5B}(\frac{2B}{3} + 1) - \frac{6}{5B}(\frac{B}{2} + 1) + \frac{1}{10} && (4.125) \\ &\geq 1 - \frac{2}{5} - \frac{3}{5B} - \frac{3}{5} - \frac{6}{5B} + \frac{1}{10} && (4.126) \\ &= \frac{1}{10} - \frac{9}{5B}, && (4.127) \end{aligned}$$

which is greater than 0 when $B > 18$. If set II is empty, (4.124) is obviously greater than 0 when $B > 18$.

case 2: $V - \lceil (B+1)/2 \rceil \in [B/6, B/3]$. In this case,

$$\begin{aligned} \delta &\geq 1 + \frac{9}{5B}\left(V - \left\lceil \frac{B+1}{2} \right\rceil\right) - \frac{1}{10} - \sum_{a_i \in I} \frac{6a_i}{5B} - \sum_{a_i \in II} \left(\frac{9a_i}{5B} - \frac{1}{10}\right) && (4.128) \\ &\geq 1 + \frac{3}{5B}\sum_{a_i \in I} a_i - \frac{1}{10} - \frac{9}{5B}\left(\frac{B}{2} + 1\right) + \frac{1}{10}\sum_{a_i \in II} 1 && (4.129) \\ &= \frac{3}{5B}\sum_{a_i \in I} a_i - \frac{9}{5B} + \frac{1}{10}\sum_{a_i \in II} 1. && (4.130) \end{aligned}$$

If II is not empty, (4.130) ≥ 0 when $B \geq 18$. If II is empty, $\sum_{a_i \in I} a_i$ will be larger than $\frac{2B}{3}$, and (4.130) will also greater than 0 when $B \geq 18$.

case 3: $V - \lceil (B+1)/2 \rceil \in [B/3, B/2)$. In this case,

$$\begin{aligned} \delta &\geq 1 + \frac{6}{5B}\left(V - \left\lceil \frac{B+1}{2} \right\rceil\right) + \frac{1}{10} - \sum_{a_i \in I} \frac{6a_i}{5B} - \sum_{a_i \in II} \left(\frac{9a_i}{5B} - \frac{1}{10}\right) && (4.131) \\ &\geq 1 - \frac{3}{5B}\sum_{a_i \in II} a_i - \frac{6}{5B}\left(\frac{B}{2} + 1\right) + \frac{1}{10} + \frac{1}{10}\sum_{a_i \in II} 1 && (4.132) \end{aligned}$$

If II is empty, (4.132) ≥ 0 when $B \geq 18$.

If II has only one element, $\sum_{a_i \in II} a_i \leq B/2$. Then, (4.132) $\geq 3/10 - 6/5B$, and will be greater than or equal to 0 when $B > 18$.

If II has two or more elements, $\sum_{a_i \in I} a_i \leq B$, and (4.132) will be greater than or equal to:

$$\geq 1 - \frac{3}{5} - \frac{3}{5} - \frac{6}{5B} + \frac{3}{10} = \frac{1}{10} - \frac{6}{5B}, \tag{4.133}$$

which is greater than 0 when $B \geq 18$. □

Theorem 14 *For a 2D discrete packing with bin size (B_1, B_2), the total weight of all items in the bin is: $\sum_{i=1}^{m} w(x_i, y_i) \leq 3.0625$, which indicates that the total weight of optimal packing on a 2D list L is: $\mathcal{W} \leq 3.0625 OPT(L)$.*

Proof. Based on the definition of weighting function $f(x)$, we have: $f(x) \leq 3/(2B)$ if $x \in [1, B/2]$. Next, we will show the proof in the following three case.

case 1: all items in the bin has width $x_i \leq B_1/2$, and height $y_i \leq B_2/2$. In this case, we have:

$$\begin{aligned} \mathcal{W} &= \sum_{i=1}^{m} w(x_i, y_i) = \sum_{i=1}^{m} f(x_i) f(y_i) && (4.134) \\ &\leq \sum_{i=1}^{m} \frac{3x_i}{2B_1} \frac{3y_i}{2B_2} && (4.135) \\ &\leq \left(\frac{3}{2}\right)^2 \frac{1}{B_1 B_2} \sum_{i=1}^{m} x_i y_i. && (4.136) \end{aligned}$$

Note that $\sum_{i=1}^{m} x_i y_i \leq B_1 B_2$, therefore, we have:

$$\mathcal{W} \leq \left(\frac{3}{2}\right)^2 < 3.0625. \tag{4.137}$$

case 2: there is only one item that has both width $x_i > B_1/2$ and height $y_i > B_2/2$.

Without loss of generality, we assume that the item was put on the left-bottom

corner of the bin. Using lemma 13, we can add more items and re-arrange other items as shown in the Figure 31. Note that the re-arrangement will not decrease the total weight.

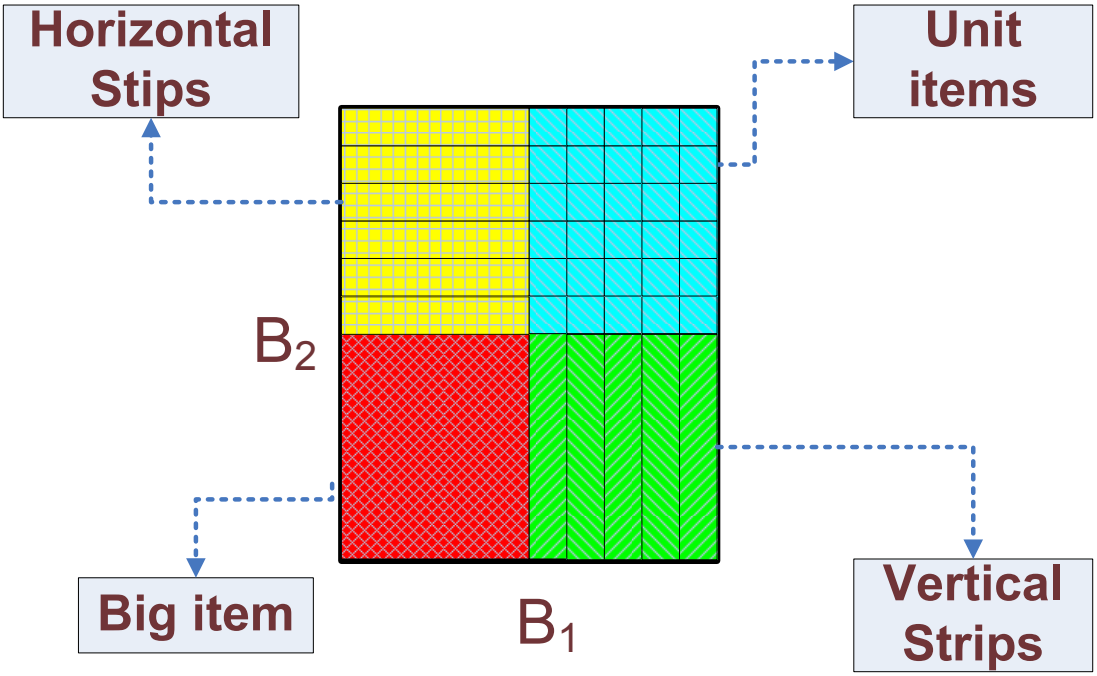

Fig. 31. Add and re-arrange items.

Using the fact $f(x) \leq 9x/(5B)$ when $x \leq B/2$, we may estimate the total weight in Figure 31 as:

$$\mathcal{W} \leq 1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 \frac{1}{4} \tag{4.138}$$

$$= 3.0625. \tag{4.139}$$

case 3: there are many items with width $x_i > B_1/2$, or height $y_i > B_2/2$. We can always use lemma 13 to combine those items, and reduce the case to case 2. □

For 2D EFF packing of item L, Theorem 14 show that the total weight for OPT is: $\mathcal{W} \leq 3.0625 OPT(L)$, and Theorem 12 shows that the total weight for EFF is:

$\mathcal{W} \geq EFF(L) - 2B_1 - 2$. Therefore, we have: $EFF(L) - 2B_1 - 2 \leq 3.0625OPT(L)$, which yields $R^{\infty}_{EFF} \leq 3.0625$. Similarly, we have: $R^{\infty}_{EBF} \leq 3.0625$.

CHAPTER V

AVERAGE-CASE ANALYSES

In this chapter, we will estimate the optimal expected waste EW_n^{OPT} of hyperbox packing, invent some non-framework-based algorithms whose expected waste is close to $\Theta(EW_n^{OPT})$, and analyze the expected waste of framework based algorithms EA. Before looking into the details, we next describe the packing system and the stochastic model.

A. Optimal Expected Waste

In this section, we first address a stochastic system of hyperbox packing, then design a stochastic model for the expected waste in the system, and finally, study the relationship between the optimal expected waste and the size distribution of arriving hyperbox-items.

Our packing system evolves with discrete time steps. At each time step t, a D-dimensional hyperbox-item comes into the system and is packed into D-dimensional hyperbox-bins with length $B_d \in Z^+$ in dimension d, $d = 1, \cdots, D$. The edge length of hyperbox-items in each dimension takes value from set $\{1, \cdots, J_d\}, J_d \leq B_d$. Note that items can not be rotated and that there are $J(D) = \prod_{d=1}^{D} J_d$ different sizes. Denote by o_i, $i = 1, \cdots, J(D)$, the size of hyperbox-items on which the length in the dth dimension is $\lfloor ((i-1) \bmod J(d))/J(d-1) \rfloor + 1$, $d = 1, \cdots, D$. At each time t, an item arrives at the system and the probability that arriving item has size o_i is λ_i, where $\sum_1^{J(D)} \lambda_i = 1$.

In order to analyze the expected waste in the above packing system, we next design a stochastic model. Note that there exist configurations c_k of items such that the items could be perfectly packed into a bin without any waste. Denote by q_{ik} the

number of items of size o_i in configuration c_k, $\mathcal{N}_k^t$ the random variable measuring the number of completely packed bins that use configuration c_k at time t. Then, at time t the number of items with size o_i in fully filled bins is:

$$\sum_{k=1}^{K} q_{ik}\mathcal{N}_k^t, \tag{5.1}$$

where K is the number of perfect-packing configurations in the system. Let us denote by $\mathcal{A}_i^t$ the number of items of size o_i in all filled bins at time t, $\mathcal{B}_i^t$ the number of items of size o_i in all partially filled bins. Obviously, we have:

$$\mathcal{A}_i^t = \sum_{k=1}^{K} q_{ik}\mathcal{N}_k^t + \mathcal{B}_i^t. \tag{5.2}$$

Representing the above equation in matrix form, we get our stochastic model as:

$$\mathcal{A}^t = Q\mathcal{N}^t + \mathcal{B}^t, \tag{5.3}$$

where $\mathcal{A}^t = (\mathcal{A}_1^t, \cdots, \mathcal{A}_{J(D)}^t)'$, $\mathcal{N}^t = (\mathcal{N}_1^t, \cdots, \mathcal{N}_K^t)'$, $\mathcal{B}^t = (\mathcal{B}_1^t, \cdots, \mathcal{B}_{J(D)}^t)'$, and $Q = (q_{ij})_{J(D)\times K}$.

1. Relationship between EW_n^{OPT}, λ and Λ

Let $\lambda = (\lambda_1, \cdots, \lambda_{J(D)})'$, and define a $J(D)$-dimensional cone as $\Lambda = \{Qf : f > 0\}$. Next, we study the asymptotic behavior of optimal expected waste EW_n^{OPT}, and show that EW_n^{OPT} asymptotically only depends on the relationship of λ and Λ.

a. λ Is Outside of Cone Λ

Theorem 15 *If λ is exterior to Λ, then the optimal expected waste is $EW_n^{OPT} = \Theta(n)$.*

PROOF. Before proving the theorem, let's first address the relationship among partially filled items $\mathcal{B}^t$, the size of bins $\vec{B} = (B_1, B_2, \cdots, B_D)$, and waste space W_t. Recall that the waste space of the system is the volume difference between filled bins and all arrived items. Obviously, the waste space is determined by $\mathcal{B}^t$ and the volume of bins. Their relationship could be shown as the following lemma.

Lemma 16 *In our hyperbox bin-packing system, the waste at time step t of any online packing algorithm is:*

$$EW_t = \Theta(E[|\mathcal{B}^t|]), \tag{5.4}$$

where $|\mathcal{B}^t| = \sum_{i=1}^{J(D)} |\mathcal{B}_i^t|$ *is the total number of hyperbox-items in partially filled bins at time t.*

PROOF. Recall that B_d is edge length of hyperbox-bins in dimension d, and that the volume of each bin is $B(D) = \prod_{d=1}^{D} B_d$. At time step t, the number of partially filled bins is in the range $(|\mathcal{B}^t|/B(D), |\mathcal{B}^t|]$, where $|\mathcal{B}^t|$ is the number of partially filled bins. For each partially filled bin, the waste space is in the range $[1, B(D))$. Therefore, we have:

$$\frac{|\mathcal{B}^t|}{B(D)} < W_t < |\mathcal{B}^t|B(D). \tag{5.5}$$

Taking expectation on both side of the above inequality and noting that $B(D)$ and D are constants, we get:

$$EW_t = \Theta(E[|\mathcal{B}^t|]). \tag{5.6}$$

□

Now let's start proving theorem 15, and first show that $W_t = \Omega(t)$. The total number of items in partially filled bins is:

$$\begin{aligned} |\mathcal{B}^t| &= |\mathcal{A}^t - Q\mathcal{N}^t| = |\mathcal{A}^t - \lambda t + \lambda t - Q\mathcal{N}^t| && (5.7)\\ &\geq |\lambda t - Q\mathcal{N}^t| - |\mathcal{A}^t - \lambda t| && (5.8)\\ &= |\lambda - Q\frac{\mathcal{N}^t}{t}|t - |\mathcal{A}^t - \lambda t|. && (5.9) \end{aligned}$$

Let $\bar{\lambda}^t = Q\mathcal{N}^t/t$ and $\alpha = \min\{|\lambda - x| : x \in \Lambda\}$. Note that λ is outside of Λ and $\alpha > 0$. Because $\mathcal{N}^t/t \geq \mathbf{0}$, $\bar{\lambda}^t$ is a random point on the cone Λ. Taking expectation on both side of (5.9), we get:

$$\begin{aligned} E\left[|\mathcal{B}^t|\right] &= E\left[|\lambda - \bar{\lambda}^t|t\right] - E\left[|\mathcal{A}^t - \lambda t|\right] && (5.10)\\ &\geq \alpha t - \sum_{i=1}^{J^D} E\left[|\mathcal{A}_i^t - \lambda_i t|\right]. && (5.11) \end{aligned}$$

Note that A_i^t follows binomial distribution with mean $\lambda_i t$ and variance $\lambda_i(1-\lambda_i)t$. Therefore,

$$E\left[|\mathcal{A}_i^t - \lambda_i t|\right] \leq \sqrt{Var(A_i^t)} = \Theta(\sqrt{t}). \quad (5.12)$$

Combining this with (5.11) and (5.4) gives:

$$EW_t^{OPT} = \Omega(t). \quad (5.13)$$

Obviously, $EW_t = O(t)$. Therefore,

$$EW_t^{OPT} = \Theta(t), \quad (5.14)$$

which yields the desired results for $EW_n^{OPT} = \Theta(n)$. $\square$

b. λ Is on the Boundary of Cone Λ

Theorem 17 *If λ is on the boundary of Λ, then the optimal expected waste is $EW_n^{OPT} = \Theta(\sqrt{n})$.*

PROOF. In order to show the tight bound, we need to show $E[W_t^{OPT}] = \Omega(\sqrt{t})$ and $E[W_t^{OPT}] = O(\sqrt{t})$. The lower bound and the upper bound are shown in STEP 1 and STEP 2 respectively.

STEP 1. Note that Λ is a $J(D)$-dimensional cone. Denote by $h = (h_1, \cdots, h_{J^D})'$ a unit-length vector that is normal to the boundary of the hyperplane of Λ. Also note that if $h'\mathcal{A}^t > 0$, $h'\mathcal{A}^t$ can be treated as the distance between point $\mathcal{A}^t$ and the boundary of the hyperplane of Λ. Let $||\mathcal{B}_t|| = \sqrt{\sum_{i=1}^{J(D)} (B_i^t)^2}$. We have,

$$||\mathcal{B}_t|| = ||\mathcal{A}^t - Q\mathcal{N}^t|| \geq \max(h'\mathcal{A}^t, 0). \tag{5.15}$$

Next, we examine the distribution of random variable $h'\mathcal{A}^t$. Let X_i^t, $i = 1, \cdots, J(D)$, be a Bernoulli random variable measuring if the arriving item at time t has size o_i. By the previous description of our packing system, random variables $X_i^1, \cdots, X_i^t$ are independent and identically distributed with PMF:

$$P[X_i^t = 1] = \lambda_i. \tag{5.16}$$

Obviously, we have $\mathcal{A}_i^t = \sum_{j=1}^{t} X_i^j$. Therefore,

$$h'\mathcal{A}^t = \sum_{i=1}^{J(D)} h_i \mathcal{A}_i^t = \sum_{i=1}^{J(D)} h_i \sum_{j=1}^{t} X_i^j \tag{5.17}$$

$$= \sum_{j=1}^{t} \left(\sum_{i=1}^{J(D)} h_i X_i^j \right). \tag{5.18}$$

Let $Z^t = \left(\sum_{i=1}^{J(D)} h_i X_i^t\right)$. Because $X_i^1, \cdots, X_i^t$ are i.i.d random variables, $Z^1, \cdots, Z^t$ are also independent and identically distributed. The expectation of Z^t is:

$$E[Z^t] = E\left[\sum_{i=1}^{J(D)} h_i X_i^t\right] = h'\lambda. \tag{5.19}$$

Note that λ is on the boundary of Λ. We have $h'\lambda = 0$, therefore, $E[Z^t] = 0$. The variance of Z^t is:

$$\begin{aligned} Var(Z^t) &= Var\left(\sum_{i=1}^{J(D)} h_i X_i^t\right) && (5.20)\\ &= \sum_{i=1}^{J(D)} h_i^2 Var(X_i^t) = \sum_{i=1}^{J(D)} h_i^2 \lambda_i(1-\lambda_i). && (5.21) \end{aligned}$$

Letting $\sigma^2 = \sum_{i=1}^{J^D} h_i^2 \lambda_i(1-\lambda_i)$ and applying central limit theorem on Z^t, we get that random variable $\sum_{j=1}^{t} Z^t/(\sigma\sqrt{t})$ follows standard normal distribution as $t \to \infty$. Using (5.18), we conclude that $h'\mathcal{A}^t$ follows normal distribution $N(0, \sigma\sqrt{t})$ as $t \to \infty$. Therefore,

$$E[(h'\mathcal{A}^t)^+] = \int_0^\infty x \frac{1}{\sqrt{2\pi}\sigma\sqrt{t}} e^{-\frac{x^2}{2\sigma^2 t}} dx = \frac{\sigma\sqrt{t}}{\sqrt{2\pi}}, \tag{5.22}$$

as $t \to \infty$. Combining the above equation with (5.15), we get:

$$E[||B_t||] = \Omega(\sqrt{t}). \tag{5.23}$$

Noting that $|B_t| \geq ||B_t||$ and using (5.4), we get desired result as:

$$EW_t^{OPT} = \Omega(\sqrt{t}). \tag{5.24}$$

STEP 2: In this step, we present an algorithm that has expected waste $EW_t = O(\sqrt{t})$. Suppose that we have K perfect configurations. Because λ is on the boundary of Λ, there exists a K-dimensional non-negative vector $f = (f_1, \cdots, f_K)'$ such that $f_k \in [0,1]$, $k = 1, \cdots, K$ and $Qf = \lambda$, where Q is a $J(D) \times K$ perfect configuration

matrix. Our algorithm proceeds in discrete phases H^t. Initially at time step 0, $H^0 = 0$ and there are no opened bins or arrived items. At each following time t, a random item a_t with size o_t comes to the system. Our packing algorithm first sets the current phase number $H^t = H^{t-1}$, then packs a_t by the following rule. If there exists any opened bins with configurations that have unoccupied slots for size o_t, a_t will be packed into one such slot. Otherwise, it keeps opening bins by the following process until there is a vacant slot for size o_t. In the bins-opening process, the algorithm sets $H^t = H^t + 1$ and opens enough new bins with certain configurations such that there are exactly $\lceil f_k H^t \rceil$ bins having configuration c_k in the system.

Next, we show that this algorithm has expected waste $EW_t = O(\sqrt{t})$. Denote by q_{ik}, $i = 1, \cdots, J(D)$, $k = 1, \cdots, K$, the number of size o_i in configuration c_k, by S_i^t the number of size-o_i slots among opened bins at time t, by $\mathcal{A}_i^t$ the number of size-o_i items at time t. The expected waste at time t is:

$$E[W_t] = E\left[\sum_{i=1}^{J(D)} \left(S_i^t - \mathcal{A}_i^t\right)\right] = \sum_{i=1}^{J(D)} E[S_i^t] - \sum_{i=1}^{J(D)} \lambda_i t. \tag{5.25}$$

Noting that

$$S_i = \sum_{k=1}^{K} q_{ik} \lceil f_k H^t \rceil \leq \sum_{k=1}^{K} q_{ik} f_k H^t + \sum_{k=1}^{K} q_{ik}, \tag{5.26}$$

and that $\sum_{k=1}^{K} q_{ik} f_k = \lambda_i$, we have:

$$E[S_i^t] \leq \lambda_i E[H^t] + \sum_{k=1}^{K} q_{ik}. \tag{5.27}$$

Applying (5.27) into (5.25) and using that $\sum_{i=1}^{J(D)} \lambda_i = 1$, we have:

$$E[W_t] \leq E[H^t] + |Q| - t, \tag{5.28}$$

where $|Q| = \sum_{i=1}^{J(D)} \sum_{k=1}^{K} q_{ik}$ is a constant that does not depend on t. Next, we compute the expectation of phases H^t. By the description of our algorithm, H^t is the

minimum integral phase that ensures that all $\mathcal{A}_i^t$ items have enough slots to sit in all opened bins. Therefore,

$$\begin{aligned} H^t &= \max_{i=1}^{J(D)} \left[\inf\{h:\ h \in \mathcal{Z}, S_i^t \geq \mathcal{A}_i^t\}\right] && (5.29)\\ &= \max_{i=1}^{J(D)} \left[\inf\{h:\ h\in\mathcal{Z}, \sum_{k=1}^{K} q_{ik}\lceil f_k h\rceil \geq \mathcal{A}_i^t\}\right] && (5.30)\\ &\leq \max_{i=1}^{J(D)} \left[\inf\{h:\ h\in\mathcal{Z}, \lambda_i h \geq \mathcal{A}_i^t\}\right] && (5.31)\\ &\leq \max_{i=1}^{J(D)} \left[\frac{\mathcal{A}_i^t}{\lambda_i}+1\right] = \max_{i=1}^{J(D)} \left[\frac{\mathcal{A}_i^t}{\lambda_i}\right]+1. && (5.32) \end{aligned}$$

Note that $\mathcal{A}_i^t$ follows binomial distribution with mean $\lambda_i t$ and variance $t\lambda_i(1-\lambda_i)$. $\mathcal{A}_i^t/\lambda_i$ follows binomial distribution $B(t, t(1-\lambda_i)/\lambda_i)$. We have:

$$\begin{aligned} E\left[\max\left[\frac{\mathcal{A}_i^t}{\lambda_i}\right]\right] &= E\left[\max\left[\frac{\mathcal{A}_i^t}{\lambda_i}\right]-t\right]+t && (5.33)\\ &\leq E\left[\sum_{i=1}^{J(D)} |\frac{\mathcal{A}_i^t}{\lambda_i}-t|\right]+t. && (5.34) \end{aligned}$$

By Cauchy-Schwarz's inequality, we have:

$$E\left[|\frac{\mathcal{A}_i^t}{\lambda_i}-t|\right] \leq \sqrt{Var\left(\frac{\mathcal{A}_i^t}{\lambda_i}\right)} = \sqrt{t}\cdot\sqrt{\frac{1-\lambda_i}{\lambda_i}}. \quad (5.35)$$

Applying (5.35) into (5.34) yields:

$$E\left[\max\left[\frac{\mathcal{A}_i^t}{\lambda_i}\right]\right] \leq t+\sqrt{t}\sum_{i=1}^{J(D)}\sqrt{\frac{1-\lambda_i}{\lambda_i}}. \quad (5.36)$$

Combining (5.36) with (5.32), (5.28), we have $E[W_t] = O(\sqrt{t})$.

Combining the results in **STEP 1** and **STEP 2**, we have: $EW_n^{OPT} = \Theta(\sqrt{n})$ when λ is on the boundary of Λ.

□

c. λ Is Inside of Cone Λ

Theorem 18 *If λ is interior to Λ, then the optimal expected waste is $EW_n^{OPT} = O(1)$.*

PROOF. The algorithm in [34] can be directly applied to our hyperbox packing. By using the same reasoning as in [34], it is trivial to show that the algorithm has expected waste $EW_n = O(1)$. □

2. Multi-Dimensional Perfect Packing Theorem

Theorem 19 *Suppose R, J, and B are positive integers with $J \leq B$. We have an infinite number of D-dimensional bins, which are hypercubes with edge length B, and R lists of hyperbox-items $\mathcal{S}_{J,D}$, each of which consists of J^D hyperbox-items with edge length $l_d = 1, \cdots, J$, $d = 1, \cdots, D$ in the dth dimension. The RJ^D items can be perfectly packed into the bins* ***if and only if*** *$R(J(J+1)/2)^D$ can be divided by B^D.*

PROOF. Show necessity first. Note that the volume of each bin is B^D and the total volume of the one item list $\mathcal{S}_{J,D}$ is:

$$Vol(S_{J,D}) = \sum_{l_1=1}^{J} \sum_{l_2=1}^{J} \cdots \sum_{l_{D-1}=1}^{J} \sum_{l_D=1}^{J} l_1 l_2 \cdots l_D \tag{5.37}$$

$$= \sum_{l_1=1}^{J} \sum_{l_2=1}^{J} \cdots \sum_{l_{D-1}=1}^{J} l_1 l_2 \cdots l_{D-1} \sum_{l_D=1}^{J} l_D \tag{5.38}$$

$$= \sum_{l_1=1}^{J} \sum_{l_2=1}^{J} \cdots \sum_{l_{D-1}=1}^{J} l_1 l_2 \cdots l_{D-1} \frac{J(J+1)}{2} \tag{5.39}$$

$$= \left(\frac{J(J+1)}{2} \right)^D. \tag{5.40}$$

Obviously, if the R copies of $S_{J,D}$ can be perfectly packed into hypercube bins, the total volume of the RJ^D items has to be divisible by the volume of one bin. Therefore, $R(J(J+1)/2)^D$ must be divisible by B^D.

Now prove the sufficiency by induction.

As shown in [31], the packing theorem holds for one dimensional case.

Assume the sufficiency holds for $\bar{D}$-dimensional case ($\bar{D} \geq 1$), in other words, if $R(J(J+1)/2)^{\bar{D}}$ can be divided by $B^{\bar{D}}$, then R copies of $\mathcal{S}_{J,\bar{D}}$ can be packed into hypercubes with length B.

We next show that the theorem holds for $\bar{D}+1$. As $R\left(J(J+1)/2\right)^{\bar{D}+1}$ is divisible by $B^{\bar{D}+1}$, we have,

$$R\left(\frac{J(J+1)}{2}\right)^{\bar{D}+1} = cB^{\bar{D}+1}, \tag{5.41}$$

where $c \in Z^+$, which implies:

$$\frac{R^{\frac{J(J+1)}{2}}}{B} = \left(R^{\bar{D}}c\right)^{\frac{1}{\bar{D}+1}}. \tag{5.42}$$

Lemma 20 *The number* $\left(R^{\bar{D}}c\right)^{1/(\bar{D}+1)}$ *in (5.42) is a positive integer.*

PROOF. Note that $R, \bar{D}, J, B \in Z^+$ and the left hand side of (5.42) is a positive rational number. Therefore, $\left(R^{\bar{D}}c\right)^{1/(\bar{D}+1)} \in Q$.

Assume that $\left(R^{\bar{D}}c\right)^{1/(\bar{D}+1)}$ is a non-integer rational number. Then, $R^{\bar{D}}c$ is also a non-integer rational number, which contradicts the fact $R, \bar{D}, c \in Z^+$. $\square$

Next, we show how to perfectly pack the $RJ^{\bar{D}+1}$ hyperbox-items into hypercube-bins. Divide the $RJ^{\bar{D}+1}$ hyperbox-items into $J^{\bar{D}}$ groups so that hyperbox-items in each group have the same edge length in dimensions from 1 to $\bar{D}$. Obviously, each group has RJ hyperbox-items and their length in the $(\bar{D}+1)$th dimension increases from 1 to J. Denote by $l_{d,m}$ the length in the dth, $d = 1, \cdots, \bar{D}$ dimension of group m, $m = 1, \cdots, J^{\bar{D}}$. Note that the problem of packing the hyperbox-items in group m into hyper-box with size $(l_{1,m}, \cdots, l_{\bar{D},m}, B)$ is equivalent to the problem of packing R lists of one-dimensional item $\mathcal{S}_{J1}$ into bins with size B. Lemma 20 and one-dimensional

perfect packing theorem in [31] indicates that hyper-boxes in group m can be piled into $(R^{\bar{D}}c)^{1/(\bar{D}+1)}$ bigger hyper-boxes with size $(l_{1,m}, \cdots, l_{\bar{D},m}, B)$. Consider the problem of packing the resulting bigger hyper-boxes in each group m into $(\bar{D}+1)$-dimensional hypercube with length B. As $l_{d,m} = 1, \cdots, J$ for $d = 1, \cdots, \bar{D}, m = 1, \cdots, J^{\bar{D}}$, the problem is equivalent to the problem of packing $(R^{\bar{D}}c)^{1/(\bar{D}+1)}$ lists $\mathcal{S}_{J,\bar{D}}$ into $\bar{D}$-dimensional hyper-boxes with length B. By (5.42), we have:

$$(R^{\bar{D}}c)^{\frac{1}{\bar{D}+1}} \left(\frac{\frac{J(J+1)}{2}}{B} \right)^{\bar{D}} = c. \tag{5.43}$$

The induction hypothesis implies that the resulting bigger hyper-boxes in each group can be perfectly packed into $(\bar{D}+1)$-dimensional hypercubes with length B. □

3. Discrete Uniform Distribution

Theorem 21 *In the online and discrete hyperbox packing, random items come to the system and their edge lengths in the dth, $d = 1, \cdots, D$, dimension independently follow uniform distribution with PMF:*

$$P[L^d = x] = 1/J_d, \tag{5.44}$$

where $x = 1, \cdots, J_d$, $J_d \leq B_d - 2$, and B_d is the edge length of hyperbox-bins in the d-th dimension. Then, the optimal expected waste is:

$$EW_n^{OPT} = \Theta(1). \tag{5.45}$$

Proof. Before proving the theorem, we next address the following important lemma.

Lemma 22 *Let $c_k = (c_k[1], \cdots, c_k[J(D)])'$ be a configuration that precisely packs a list $\mathcal{S}_{J,D}$ of D-dimensional hyperbox-items into hyperbox-bins with edge length B_d in*

dimension d, where $J(D) = \prod_{d=1}^{D} J_d$, $d = 1, \cdots, D$, $c_k[i]$ *is the number of items with size* o_i *in the list* $\mathcal{S}_{J,D}$. *Denote by* $q_{ik} = c_k[i]$ *the number of items with size* o_i *in configuration* c_k. *Then, there exists a square matrix* $Q_D = (q_{ik})_{J(D)\times J(D)}$ *such that* $J(D)$*-dimensional vector* $e_D = (1/J(D), \cdots, 1/J(D))'$ *is in the interior of the cone* $\Lambda_D = \{Q_D f_D : f_D \geq 0)$, *where* f_D *is any positive vector with* $J(D)$*-dimensions.*

Note that Theorem 21 follows by applying this lemma to the stability property of D-dimensional hyper-box packing. Therefore, we just need to prove Lemma 22.

We prove it by induction. Using our previous notation, D-dimensional hyper-boxes with size o_i have length $\lfloor((i-1) \bmod J(d))/J(d-1)\rfloor + 1$ in the dth dimension, where $d = 1, \cdots, D$.

STEP 1. When $D = 1$, we have $o_i = i$ where $i = 1, \cdots, J_1$. By [31], there exist $r_k, s_k \in Z^+$ and J_1 perfect configurations c_k, $k = 1, \cdots, j$, such that $c_k[i] = r_k$ for $i \neq k$ and $c_k[k] = s_k + r_k$. Therefore, we have:

$$Q_1 = \begin{pmatrix} r_1+s_1 & r_2 & \cdots & r_k & \cdots & r_{J_1} \\ r_1 & r_2+s_2 & \cdots & r_k & \cdots & r_{J_1} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ r_1 & r_2 & \cdots & r_k+s_k & \cdots & r_{J_1} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ r_1 & r_2 & & r_k & & r_{J_1}+s_{J_1} \end{pmatrix}, \tag{5.46}$$

in which each column is a perfect configuration. To make sure that $e_1 = (1/J_1, \cdots, 1/J_1)'$ is in the interior of cone Λ_1, we next show that e_1 and its ϵ_1-neighborhood for sufficiently small ϵ_1 are on the cone. By the definition of cone, if matrix equation

$$Q_1 f_1 = e_1, \tag{5.47}$$

has positive solution $f_1 > 0$ then e_1 is in cone Λ_1. Using Gaussian Elimination and letting the ith row $Q_1[i] = (Q_1[i] - Q_1[i+1])/s_i$, $i = 1, \cdots, J_1 - 1$, we get:

$$\left(\begin{array}{ccccc|c} 1 & -\frac{s_2}{s_1} & 0 & \cdots & 0 & 0 \\ 0 & 1 & -\frac{s_3}{s_2} & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & -\frac{s_{J_1}}{s_{J_1-1}} & 0 \\ r_1 & r_2 & r_3 & \cdots & r_{J_1}+s_{J_1} & \frac{1}{J_1} \end{array}\right)\left(\begin{array}{c} f_1[1] \\ f_1[2] \\ \vdots \\ f_1[J_1-1] \\ f_1[J_1] \end{array}\right). \tag{5.48}$$

Repeatedly replace the J_1th row $Q_1[J_1]$ by $Q_1[J_1] - E_i * Q_1[i]$, where $i = 1, \cdots, J_1 - 1$ and

$$E_1 = r_1, \tag{5.49}$$

$$E_{i+1} = r_{i+1} + \frac{s_{i+1}}{s_i} E_i. \tag{5.50}$$

The above row operations will eliminate the first $J_1 - 1$ elements of row J_1. Solving matrix equation (5.48) by backward substitution, we get:

$$f_1[i] = f_1[i+1]\frac{s_{i+1}}{s_i}, i = 1, \cdots, J_1 - 1 \tag{5.51}$$

$$f_1[J_1] = \frac{1}{J_1}\frac{1}{E_{J_1} + s_{J_1}}. \tag{5.52}$$

Note that the E_i are all positive rational numbers. Therefore, $f_1 > \mathbf{0}$ and e_1 is in Λ_1. Now we show that $\exists \epsilon_1 > 0$ such that ϵ_1-neighborhood of e_1 is also on Λ_1. Let $e_1^* = (1/J_1 + \delta_1, \cdots, 1/J_1 + \delta_{J_1})$ be any point in the ϵ_1-neighborhood, i.e. $\sqrt{\sum_{i=1}^{J_1} \delta_i^2} \leq \epsilon_1$. Our objective is to find proper ϵ_1 such that matrix equation

$$Q_1 f_1^* = e_1^*, \tag{5.53}$$

has non-negative solution, i.e. $f_1^* \geq \mathbf{0}$, which would imply e_1^* is also on the cone Λ_1. Using the above Gaussian Elimination technique, we get matrix equation (5.54) as

in Figure 32. Backward substitution of $f_1^*[J_1]$ into matrix equation (5.54) from row

$$\left(\begin{array}{ccccc|c} 1 & -\frac{s_2}{s_1} & 0 & \cdots & 0 & (\delta_1-\delta_2)/s_1 \\ 0 & 1 & -\frac{s_3}{s_2} & \cdots & 0 & (\delta_2-\delta_3)/s_2 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ 0 & 0 & 0 & 1 & -\frac{s_{J_1}}{s_{J_1-1}} & (\delta_{J_1-1}-\delta_{J_1})/s_{J_1-1} \\ 0 & 0 & 0 & \cdots & E_{J_1}+s_{J_1} & \frac{1}{J_1}+\delta_{J_1}-\sum_{i=1}^{J_1-1}E_i\frac{\delta_i-\delta_{i-1}}{s_i} \end{array}\right)\left(\begin{array}{c} f_1^*[1] \\ f_1^*[2] \\ \vdots \\ f_1^*[J_1-1] \\ f_1^*[J_1] \end{array}\right) \tag{5.54}$$

Fig. 32. Matrix equation $Q_1 e_1^* = f_1^*$ after Gaussian elimination.

$J_1 - 1$ to row 1 gives:

$$f_1^*[i] = \frac{s_{J_1}}{s_i} f_1^*[J_1] + \frac{\delta_i - \delta_{J_1}}{s_i}, \tag{5.55}$$

where $i = 1, \cdots, J_1 - 1$. Note that $s_{J_1} \geq 1$ and $\delta_i - \delta_{J_1} \geq -2\epsilon_1$. Equation (5.55) implies that $f_1^*[J_1] \geq 2\epsilon_1 > 0$ will ensure $f_1^* \geq \mathbf{0}$. By the last row of matrix equation (5.54), we get $f_1^*[J_1]$:

$$f_1^*[J_1] = \frac{\frac{1}{J_1} + \delta_{J_1} - \sum_{i=1}^{J_1-1} E_i \frac{\delta_i - \delta_{i-1}}{s_i}}{(E_{J_1} + s_{J_1})}, \tag{5.56}$$

which would be larger than or equal to $2\epsilon_1$ if the following inequality holds:

$$\frac{1}{J_1} - \epsilon_1 - \sum_{i=1}^{J_1-1} E_i \frac{2\epsilon_1}{s_i} \geq (E_{J_1} + s_{J_1}) 2\epsilon_1. \tag{5.57}$$

Solving the inequality, we get that when

$$\epsilon_1 \leq \frac{1}{J_1} \frac{1}{1 + 2\sum_{i=1}^{J_1-1} \frac{E_i}{s_i} + 2(E_{J_1} + s_{J_1})}, \tag{5.58}$$

(5.57) holds $\Rightarrow f_1^*[J_1] \geq 2\epsilon_1 \Rightarrow f_1^* \geq \mathbf{0}$, therefore, ϵ_1-neighborhood of e_1 is in Λ_1. Combining this with previous result that e_1 is on cone Λ_1, we conclude that e_1 is in the interior of cone Λ_1.

STEP 2. Assume that the lemma is true for $\bar{D}$ dimensional hyper-box packing. In other words, there exists $J(\bar{D}) \times J(\bar{D})$ matrix $Q_{\bar{D}}$ such that: (i). each column of $Q_{\bar{D}}$ is a perfect packing configuration; (ii). $e_{\bar{D}}$ is inside of the cone $\Lambda_{\bar{D}}$ spanned by $Q_{\bar{D}}$. In the next step, we show that for $(\bar{D}+1)$-dimensional hyper-boxes, the lemma also holds.

STEP 3. We claim that $J(\bar{D}+1) \times J(\bar{D}+1)$ square matrix $Q_{\bar{D}+1}$ in (5.59) of Figure 33 meets our needs, where r_k and s_k are the same as those in matrix Q_1.

$$\begin{pmatrix} (r_1+s_1)Q_{\bar{D}} & r_2Q_{\bar{D}} & \cdots & r_kQ_{\bar{D}} & \cdots & r_{J_{\bar{D}+1}}Q_{\bar{D}} \\ r_1Q_{\bar{D}} & (r_2+s_2)Q_{\bar{D}} & \cdots & r_kQ_{\bar{D}} & \cdots & r_{J_{\bar{D}+1}}Q_{\bar{D}} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ r_1Q_{\bar{D}} & r_2Q_{\bar{D}} & \cdots & (r_k+s_k)Q_{\bar{D}} & \cdots & r_{J_{\bar{D}+1}}Q_{\bar{D}} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ r_1Q_{\bar{D}} & r_2Q_{\bar{D}} & & r_kQ_{\bar{D}} & & (r_{J_{\bar{D}+1}}+s_{J_{\bar{D}+1}})Q_{\bar{D}} \end{pmatrix}, \quad (5.59)$$

Fig. 33. Perfect configuration matrix in $(\bar{D}+1)$-dimensional hypercube packing.

First, we show that each column of matrix (5.59) is a perfect configuration. Note that the kth column of matrix $Q_{\bar{D}+1}$ consists of $J_{\bar{D}+1}$ sub-matrices $C_k[i]Q_{\bar{D}}$, where $i = 1, \cdots, J_{\bar{D}+1}$ and

$$C_k[i] = \begin{cases} r_k, & \text{if } i \neq k \\ s_k + r_k, & \text{if } i = k. \end{cases} \quad (5.60)$$

Also note that $C_k[i]Q_n[x][y]$ represents the number of $(\bar{D}+1)$-dimensional hyper-boxes with size $o_{(i-1)J(\bar{D})+x}$ in configuration $(k-1)J(\bar{D})+y$. By the assumption in STEP 2, we can use sub-matrix $C_k[i]Q_{\bar{D}}$ to pack hyperbox-items with size from $o_{(i-1)J(\bar{D})+1}$ to $o_{iJ(\bar{D})}$ into bigger hyperbox that have edge length B_d in dimensions $d = 1, \cdots, \bar{D}$

and edge length i in the $(\bar{D}+1)$th dimension. Packing these bigger hyper-boxes into hyperbox-bins is equivalent to packing 1-dimensional items. Reusing the result in [31], we know that each column in $Q_{\bar{D}+1}$ is a perfect configuration.

Next, we show that $e_{\bar{D}+1} = (1/J(\bar{D}+1), \cdots, 1/J(\bar{D}+1))'$ and its $\epsilon_{\bar{D}+1}$-neighborhood are on the cone $\Lambda_{\bar{D}+1}$ spanned by $Q_{\bar{D}+1}$. It is easy to check that there exists non-negative vector $f_{\bar{D}+1} = (f'_{\bar{D}} f_1[1], \cdots, f'_{\bar{D}} f_1[J_{\bar{D}+1}])' > 0$ such that:

$$Q_{\bar{D}+1} f_{\bar{D}+1} = e_{\bar{D}+1}. \tag{5.61}$$

Therefore, $e_{\bar{D}+1}$ is on the cone $\Lambda_{\bar{D}+1}$. Now we show that the $\epsilon_{\bar{D}+1}$-neighborhood of $e_{\bar{D}+1}$ is also on $\Lambda_{\bar{D}+1}$. Let $e^*_{\bar{D}+1} = e_{\bar{D}+1} + (\Delta_1, \cdots, \Delta_{J_{\bar{D}+1}})'$ be any point in the neighborhood, where

$$\Delta_i = (\delta_i[1], \cdots, \delta_i[J(\bar{D})])', \tag{5.62}$$

and $\sum_{i=1}^{J_{\bar{D}+1}} \|\Delta_i\|^2 \leq \epsilon^2_{\bar{D}+1}$. The positive solution $f^*_{\bar{D}+1} > 0$ of matrix equation:

$$Q_{\bar{D}+1} f^*_{\bar{D}+1} = e^*_{\bar{D}+1}, \tag{5.63}$$

would imply that $e^*_{\bar{D}+1}$ is in cone $\Lambda_{\bar{D}+1}$. Applying the Gaussian Elimination technique in STEP 1 to the sub-matrices in $Q_{\bar{D}+1}$ rather than each elements, we get the matrix as in Figure 34, where $\Delta^*_{J_{\bar{D}+1}} = J_{\bar{D}+1}\left(\Delta_{J_{\bar{D}+1}} - \sum_{i=1}^{J_{\bar{D}+1}-1} E_i(\Delta_i - \Delta_{i-1})/s_i\right)$. Applying backward substitution in matrix in Figure 34, we get:

$$Q_{\bar{D}} f^*_{\bar{D}+1}[J_{\bar{D}+1}] = \frac{1}{J_{\bar{D}+1}(E_{J_{\bar{D}+1}} + s_{J_{\bar{D}+1}})}(e_{\bar{D}} + \Delta^*_{J_{\bar{D}+1}}), \tag{5.64}$$

$$Q_{\bar{D}} f^*_{\bar{D}+1}[i] = \frac{s_{J_{\bar{D}+1}}}{s_i J_{\bar{D}+1}(E_{J_{\bar{D}+1}} + s_{J_{\bar{D}+1}})}(e_{\bar{D}} + \Delta^*_i), \tag{5.65}$$

$$\begin{pmatrix} Q_{\bar{D}} & -\frac{s_2}{s_1}Q_{\bar{D}} & \mathbf{0} & \cdots & \mathbf{0} & | & (\Delta_1-\Delta_2)/s_1 \\ \mathbf{0} & Q_{\bar{D}} & -\frac{s_3}{s_2}Q_{\bar{D}} & \cdots & \mathbf{0} & | & (\Delta_2-\Delta_3)/s_2 \\ \vdots & \ddots & \ddots & \ddots & & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & Q_{\bar{D}} & -\frac{s_{J_{\bar{D}+1}}}{s_{J_{\bar{D}+1}-1}}Q_{\bar{D}} & | & \frac{\Delta_{J_{\bar{D}+1}-1}-\Delta_{J_{\bar{D}+1}}}{s_{(J_{\bar{D}+1}-1)}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & (E_{J_{\bar{D}+1}}+s_{J_{\bar{D}+1}})Q_{\bar{D}} & | & \frac{e_{\bar{D}}+\Delta^*}{J_{\bar{D}+1}} \end{pmatrix} \begin{pmatrix} f^*_{\bar{D}+1}[1] \\ f^*_{\bar{D}+1}[2] \\ \vdots \\ f^*_{\bar{D}+1}[J_{\bar{D}+1}-1] \\ f^*_{\bar{D}+1}[J_{\bar{D}+1}] \end{pmatrix},$$

Fig. 34. Matrix equation $Q_{\bar{D}+1}e^*_{\bar{D}+1} = f^*_{\bar{D}+1}$ after Gaussian elimination.

where $\Delta^*_i = \Delta^*_{J_{\bar{D}+1}} + J_{\bar{D}+1}(E_{J_{\bar{D}+1}} + s_{J_{\bar{D}+1}})(\Delta_i - \Delta_{J_{\bar{D}+1}})/s_{J_{\bar{D}+1}}$, and $i = 1, \cdots, J_{\bar{D}+1}-1$. By the assumptions in STEP 2, the following inequalities will guarantee $f^*_{\bar{D}+1} \geq 0$:

$$||\Delta^*_{J_{\bar{D}+1}}|| \leq \epsilon_{\bar{D}}, \tag{5.66}$$

$$||\Delta^*_i|| \leq \epsilon_{\bar{D}}. \tag{5.67}$$

Solving the above inequalities, we get:

$$\epsilon_{\bar{D}+1} \leq \frac{\epsilon_{\bar{D}}}{2J(\bar{D}+1)(1+\sum_{i=1}^{J_{\bar{D}+1}}\frac{E_i}{s_i})}. \tag{5.68}$$

□

B. Theoretical Algorithms of Discrete Hyperbox Packing

Recall that the expected waste is the major metric for us to measure the average-case performance of hyperbox packing algorithms, and our previous results show that optimal expected waste EW_n^{OPT} asymptotically takes three forms $\Theta(1)$, $\Theta(\sqrt{n})$, and $\Theta(n)$ according to the relationship between λ and Λ. We are interested in inventing hyperbox packing algorithms that asymptotically have sub-linear expected waste. In this section, we present two hyperbox bin-packing algorithms that have sub-linear

expected waste $O(\sqrt{n})$ if λ is on the boundary of cone Λ.

1. Grouping Packing (GP) Algorithm

Input: It proceeds with n steps. At each time step from 1 to n, one item with size o_i randomly arrives with rational probability λ_i, $i = 1, \cdots, J$. The input also includes infinite number of closed bins.

Output: A couple of opened bins that contain the n random arrival items. The total size of item in each opened bin does not exceed each bin's size B.

1. According to the items size $o_1, \cdots, o_J$ and the bins size B, find the perfect configurations and calculate the perfect packing matrix Q.
2. Let $f = Q^{-1}\lambda$, where $f = (f_1, \cdots, f_K)'$.
3. Find a positive integer T s.t. $f_1T, \cdots, f_KT$ are all integers.
4. *For* each random arrival item with size o_i *do*
5. *If* among the opened bins, there exist available slots for size o_i, put the item into one of those slots.
6. *Else* open a group of $T\sum_{k=1}^{K} f_k$ bins, in which Tf_k bins are assigned configuration c_k. Then, put the item with size o_i into one of available slots.
7. *End For*

Fig. 35. Grouping bin packing algorithm.

Let's first deal with a special case of the hyperbox packing problem, in which the items are of size $o_1, o_2, \cdots, o_J$, and the size arriving probability λ_i, $i = 1, \cdots, J$, are rational.

Our algorithm to solve this problem is called Grouping Packing (GP) algorithm. Note that the expected number of size o_i at time t is $t\lambda_i$, $i = 1, \ldots, J$. The key idea of our algorithm is to perfectly pack these expected items using grouping technique.

Note that λ is on the cone Λ. There exists a vector $f = (f_1, \cdots, f_K)' \geq \mathbf{0}$ such that $Qf = \lambda$. As each element in matrix Q is an integer, therefore, $f = Q^{-1}\lambda$ is a rational vector. Let T be a positive integer such that $f_k T$, $k = 1, \cdots, K$, are integers. Then, equation

$$QfT = \lambda T, \tag{5.69}$$

implies that $\lambda_i T$, $i = 1, \cdots, J$, are also integers. By the definition of perfect packing matrix Q, we could use $f_k T$ bins with assigned configuration c_k, $k = 1, \cdots, K$, to perfectly pack $\lambda_i T$ items with size o_i, $i = 1, \cdots, J$. Note that $\lambda_i T$ is the expected number of size o_i at time step T. Our algorithm proceeds as follows. When an item with size o_i comes into the system, it checks if there are available slots for that size. If it is yes, the item will be put into one of those slots. Otherwise, it opens $\sum_{k=1}^{K} f_k T$ bins, in which $f_k T$ bins are assigned configuration c_k, $k = 1, \cdots, K$, and pack the item accordingly. The detailed description of the algorithm is shown as in Figure 35.

Obviously, the above algorithm is polynomial to the number of packed items n. Next, we show that its expected waste is $O(\sqrt{n})$.

Note that in a newly opened bin with assigned configuration c_k, there are q_{ik} available slots for size o_i, where q_{ik} is the (i,k)th element of matrix Q. By line 6 in the above algorithm, a group of Tf_k bins with configuration c_k, $k = 1, \cdots, K$, is opened whenever no slots are available for the arriving item. In the group of opened bins, the number of available slots for size i is:

$$\sum_{k=1}^{K} T f_k q_{ik} = T\lambda_i. \tag{5.70}$$

Let $\mathcal{A}_i^t$ be random variable measuring the number of arrival size o_i at time t. Then, the number of groups required by size o_i is $\mathcal{A}_i^t/(T\lambda_i)$. Let $G = B\sum_{k=1}^{K} Tf_k$ be the total size of bins in a group. The expected waste of the algorithm is:

$$\begin{aligned} EW_t &\leq GE\left[\max\{\frac{\mathcal{A}_i^t}{T\lambda_i}, i=1,\cdots,K\} - \min\{\frac{\mathcal{A}_i^t}{T\lambda_i}, i=1,\cdots,K\} + 1\right] && (5.71)\\ &\leq \frac{G}{T}E\left[\max\{\frac{\mathcal{A}_i^t}{\lambda_i}, i=1,\cdots,K\} - \min\{\frac{\mathcal{A}_i^t}{\lambda_i}, i=1,\cdots,K\}\right] + G && (5.72)\\ &\leq \frac{G}{T}E\left[\sum_{i=1}^{J}\sum_{j=1}^{J}|\frac{\mathcal{A}_i^t}{\lambda_i} - \frac{\mathcal{A}_j^t}{\lambda_j}|\right] + G && (5.73)\\ &\leq \frac{G}{T}\sum_{i=1}^{J}\sum_{j=1}^{J}E\left[|\frac{\mathcal{A}_i^t}{\lambda_i} - \frac{\mathcal{A}_j^t}{\lambda_j}|\right] + G && (5.74)\\ &\leq \frac{G}{T}\sum_{i=1}^{J}\sum_{j=1}^{J}E\left[|\frac{\mathcal{A}_i^t}{\lambda_i} - t| + |\frac{\mathcal{A}_j^t}{\lambda_j} - t|\right] + G. && (5.75) \end{aligned}$$

Note that $\mathcal{A}_i^t$ follows binomial distribution $B(\lambda_i t, \lambda_i(1-\lambda_i)t)$, $\mathcal{A}_i^t/\lambda_i$ follows binomial distribution with mean t, and variance $t(1-\lambda_i)/\lambda_i$. By Schwarz-Cauchy inequality, we have:

$$E\left[|\frac{\mathcal{A}_i^t}{\lambda_i} - t|\right] \leq \sqrt{Var(\mathcal{A}_i^t/\lambda_i)} \leq \sqrt{t}\cdot\sqrt{\frac{1-\lambda_i}{\lambda_i}}. \quad (5.76)$$

Combining the above equation with (5.75), we have:

$$EW_t \leq \frac{2G}{T}J\sqrt{t}\sum_{i=1}^{J}\sqrt{\frac{1-\lambda_i}{\lambda_i}} + G. \quad (5.77)$$

As G, T,and J do not depend on time step t, therefore, the expected waste at time step t is $O(\sqrt{t})$.

2. Accumulative Packing (AP) Algorithm

Note that this grouping packing (GP) algorithm restricts the size arriving probability to be rational. Next, we present another algorithm that removes this restriction.

Suppose that we have K perfect configurations. Because λ is on the boundary of Λ, there exists a K-dimensional non-negative vector $f = (f_1, \cdots, f_K)'$ such that $f_k \in [0, 1]$, $k = 1, \cdots, K$ and $Qf = \lambda$, where Q is a $J^D \times K$ perfect configuration matrix. Our algorithm proceeds in discrete phases H^t. In time step 0, $H^0 = 0$ and there are no opened hypercubes (bins) or arrived hyper-boxes(items). At each time step t, a random hyper-box (item) a_t with size o_t comes to the system. Our packing algorithm first sets the current phase number as $H^t = H^{t-1}$, then packs a_t by the following rule. If there exists any opened bins with configurations that have unoccupied slots for size o_t, a_t will be packed into one such slot. Otherwise, it keeps opening bins by the following process until there is a vacant slot for size o_t. In the bins-opening process, the algorithm sets $H^t = H^t + 1$ and opens enough new bins with certain configurations such that there are exactly $\lceil f_k H^t \rceil$ bins having configuration c_k in the system.

Note that the algorithm is the same as the algorithm in STEP 2 of proof of theorem 17. As shown in the proof of the theorem, the expected waste of AP algorithm is: $O(\sqrt{n})$. Since the λ is on the boundary of Λ, and $EW_n^{OPT} = \Theta(\sqrt{n})$, we have $EW_n^{AP} = \Theta(\sqrt{n})$ as well.

C. Average-Case Analysis of Framework-Based Algorithms

Recall that there are many existing results on average-case analysis of 1D packing algorithms, and it would be nice to extend those 1D results to hyperbox packing. Next, we will show the expected waste relationship between 1D packing algorithm A and framework based algorithm EA.

Theorem 23 *Denote by A a 1D packing algorithms, and there are D different 1-dimensional bin packing problems where bins' sizes are $B_1, \cdots, B_D$. Let $\Theta(f(n, d))$*

be the expected waste of algorithm A that packs n items into bins of size B_d, $d = 1, \cdots, D$, and items' sizes follow discrete distribution $\mathcal{F}_d$. For the hyperbox packing problem where bins' size is $\vec{B} = (B_1, \cdots, B_D)$, and items' edge length in dimension d independently follow distribution $\mathcal{F}_d$, the expected waste of the framework algorithm EA for the packing is dominated by the maximum of $\Theta(f(n,d))$ for $d = 1, \cdots, D$.

PROOF. Recall that in framework algorithms, all items packed in the dth dimension have the same edge length in dimensions $1, \cdots, d-1$, and items packed in dth dimension will form a strip. Note that the waste area in a partially-packed bin could be categorized into D types, where the ith type are those hyperbox areas with size: $(l_1, l_2, \cdots, l_{i-1}, w_i^{l_1,\cdots,l_{i-1}}, B_{i+1}, B_D)$, where l_d, $d = 1, \cdots, i-1$, are the edge lengths of packed i-pile in dimension d, $w_i^{l_1,\cdots,l_{i-1}}$ is the total waste of length in i-pile of size $(l_1, \cdots, l_{i-1})$ in dimensions 1 to $i-1$, and B_d, $d = i+1, \cdots, D$, is bins' edge length in dimension d. Figure 36 gives an example of the waste types in 2D packing. Since the edge length in dimension d of hyperbox bins independently follows distribution $\mathcal{F}_d$, $w_i^{l_1,\cdots,l_{i-1}}$ is corresponding to the waste of algorithm A in the 1D packing.

Denote by $W[i]$ the waste space in the ith dimension. Obviously, the total waste of a packing instance is:

$$W = \sum_{d=1}^{D} W[d]. \tag{5.78}$$

Also note that

$$W[i] = \sum_{l_1=1}^{B_1} \cdots \sum_{l_{i-1}=1}^{B_{i-1}} \left[\left(\prod_{d=1}^{i-1} l_d \right) w_i^{l_1,\cdots,l_i-1} \left(\prod_{d=i+1}^{D} B_d \right) \right]. \tag{5.79}$$

Therefore, W will be dominated by the biggest $w[i]$ in all dimensions from 1 to D. □

Using the results form theorem 23, we have:

Corollary 24 *In a hyperbox packing, let the edge length of hyperbox-item indepen-*

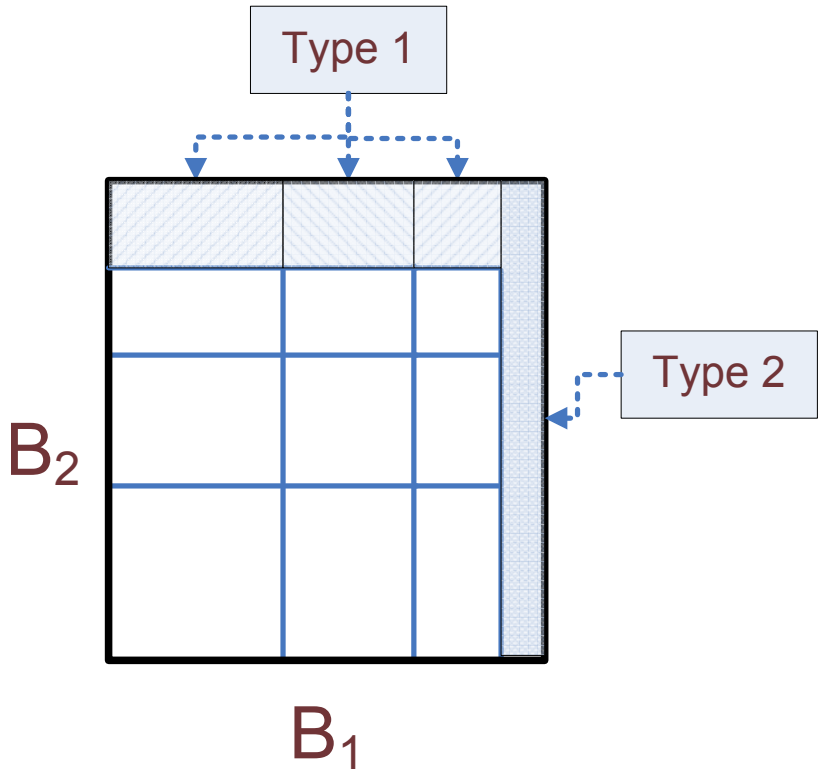

Fig. 36. The waste type in 2D packing.

dently follows distribution $\mathcal{F}$*, and denote by* EW_n^{OPT} *the optimal expect waste for corresponding 1D packing under distribution* $\mathcal{F}$*. Then, we have:*

$$EW_n^{ESS} = \begin{cases} \Theta(EW_n^{OPT}) & \text{if } EW_n^{OPT} \neq \Theta(1) \\ O(\log n) & \text{otherwise} \end{cases}. \tag{5.80}$$

Corollary 25 *Framework based algorithms EBF and EFF for hyperbox packing with bins' size* $\vec{B} = (B_1, \cdots, B_D)$ *have bounded expected waste* $\Theta(1)$ *if all the edge length of items independently follows uniform distribution* $U\{B_d - 2, B_d\}$*, where* $d = 1, \cdots, D$*.*

D. Optimal Expected Waste of Continuous Hyperbox Packing

We are also interested in comparing the optimal expected waste between the continuous hyperbox packing and discrete packing. Next, we show that the optimal expected waste for continuous hyperbox packing is $\Omega(n^{1/2})$ if the edge length of items independently follow some uniform distribution.

Theorem 26 *In the online and continuous D-dimensional hyperbox packing, the edge lengths of hyperboxes-items in the dth, $d = 1, \cdots, D$, dimension independently follow uniform distribution $U(0, u]$, where $u \in (0, 1]$, and bins are unit hypercube with edge length 1. Then, the optimal expected waste is $EW_n^{OPT} = \Omega(n^{1/2})$, where n is the number of items to be packed.*

Proof. Let a_{t+1} be the $(t+1)$th item (hyper-box), where $t = 0, \cdots, n-1$ is the index of all items. Denote by $L^d_{a_{t+1}}$, $d = 1, \cdots, D$, the random variable measuring the edge length of hyper-box a_{t+1} in the dth dimension. We next estimate the lower bound of expected waste for any online packing algorithm A.

Denote by $\omega(t)$ the random variable measuring the waste space after t items (hyper-boxes) have been packed into bins. Let $\mathcal{L}^d_{a_i}$, $i = 1, \cdots, t$, $d = 1, \cdots, D$, be probability space of random variable $L^d_{a_i}$. By conditional expectation, we have:

$$E[\omega(t)] \quad = \quad E\Big[\ E\big[\omega(t)|\{\mathcal{L}^1_{a_1}, \cdots, \mathcal{L}^d_{a_1}\}, \cdots, \{\mathcal{L}^1_{a_t}, \cdots, \mathcal{L}^d_{a_t}\}\big]\ \Big]. \tag{5.81}$$

Denote by $F^t_{a_s}$ a Bernoulli random variable indicting whether item a_s is last in a bin at time step t, $t \geq s$, by $U_{a_{t+1}}$ a random variable measuring the volume of unused space in the bin, in which item a_{t+1} is just packed by algorithm A. Let $\delta = u^{D+1} n^{-1/2} / 2^{D+2}$. Then, the inner expectation of the right hand side in (5.81) can be computed as:

$$E\Big[\omega(t)|\{\mathcal{L}^1_{a_1}, \cdots, \mathcal{L}^d_{a_1}\}, \cdots, \{\mathcal{L}^1_{a_t}, \cdots, \mathcal{L}^d_{a_t}\}\Big] \tag{5.82}$$

$$= \sum_{s=0}^{t-1} E\left[F^t_{a_{s+1}} U_{a_{s+1}} | \mathcal{L}^1_{a_{s+1}}, \cdots, \mathcal{L}^d_{a_{s+1}}\right] \tag{5.83}$$

$$\geq \sum_{s=0}^{t-1} (\delta \prod_{d=2}^{D} L^d_{a_{s+1}}) P\left[F^t_{a_{s+1}}=1, U_{a_{s+1}} \geq \delta \prod_{d=2}^{D} L^d_{a_{s+1}}\right]. \tag{5.84}$$

Applying (5.84) into (5.81) and using that $L^d_{a_{s+1}}$ follows uniform distribution $U(0,u]$,, we get:

$$\begin{aligned}
& E\left[\omega(t)\right] \\
\geq\ & E\left[\ \sum_{s=0}^{t-1} \delta \prod_{d=2}^{D} L^d_{a_{s+1}} P\left[F^t_{a_{s+1}}=1, U_{a_{s+1}} \geq \delta \prod_{d=2}^{D} L^d_{a_{s+1}}\right]\ \right] \\
=\ & \delta\left(\tfrac{u}{2}\right)^{D-1} \sum_{s=0}^{t-1} P\left[F^t_{a_{s+1}}=1, U_{a_{s+1}} \geq \delta \prod_{d=2}^{D} L^d_{s+1}\right] \\
\geq\ & \delta\left(\tfrac{u}{2}\right)^{D-1} \sum_{s=0}^{t-1} \left(P\left[F^t_{a_{s+1}}=1\right] - P\left[U_{a_{s+1}} \leq \delta \prod_{d=2}^{D} L^d_{s+1}\right]\right).
\end{aligned} \tag{5.85}$$

To estimate the lower bound of of (5.85), we first compute the first term in the summation. Let S^t be the volume summation of first t items. Note that the number of partially filled bins is at least S^t after the first t items having been packed. Therefore,

$$\sum_{s=0}^{t-1} P\left[F^t_{a_{s+1}} = 1\right] \tag{5.86}$$

$$\geq E[S_t] = E\left[\sum_{s=0}^{t-1} \prod_{d=1}^{D} L^d_{a_{s+1}}\right] = t\left(\frac{u}{2}\right)^D. \tag{5.87}$$

To estimate the second term inside the summation of (5.85), we use the following lemma.

Lemma 27 *Let's use the same random variable $U_{a_{t+1}}$ as mentioned before and denote by $v(t)$ a random variable measuring the number of such partially filled bins at time step t that consist of at least one unused hyper-box room with the edge length in the first dimension larger than or equal to $u^{D+1}/2^{D+2}$, the edge length in the dth dimension,*

$d = 2, \cdots, D$, larger than or equal to $L^d_{a_{t+1}}$. Then, we have:

$$P\left[U_{a_{t+1}} \le \delta \prod_{d=2}^{D} L^d_{a_{t+1}}\right] \le \frac{u^D}{2^{D+1}} + P\left[v(t) \ge n^{1/2}\right], \tag{5.88}$$

PROOF.

$$\begin{aligned} & P \quad \left[U_{a_{t+1}} \le \delta \prod_{d=2}^{D} L^d_{a_{t+1}}\right] && (5.89)\\ \le \quad & P\left[L^1_{a_{t+1}} \le \frac{u^{D+1}}{2^{D+2}}\right] \\ + \quad & P\left[L^1_{a_{t+1}} \ge \frac{u^{D+1}}{2^{D+2}}, U_{a_{t+1}} \le \delta \prod_{d=2}^{D} L^d_{a_{t+1}}\right]. && (5.90) \end{aligned}$$

Note that $L^1_{a_{t+1}}$ follows uniform distribution $U(0, u]$. Therefore, the first term of (5.90) is:

$$P\left[L^1_{a_{t+1}} \le \frac{u^{D+1}}{2^{D+2}}\right] = \frac{u^D}{2^{D+2}}. \tag{5.91}$$

The second term of (5.90) is:

$$\begin{aligned} & P\left[L^1_{a_{t+1}} \ge \frac{u^{D+1}}{2^{D+2}}, U_{a_{t+1}} \le \delta \prod_{d=2}^{D} L^d_{a_{t+1}}\right] && (5.92)\\ \le \quad & P\left[v(t) \le n^{1/2}, L^1_{a_{t+1}} \ge \tfrac{u^{D+1}}{2^{D+2}}, U_{a_{t+1}} \le \delta \textstyle\prod_{d=2}^{D} L^d_{a_{t+1}}\right] \\ + \quad & P\left[v(t) \ge n^{1/2}\right] && (5.93) \end{aligned}$$

Recall that $U_{a_{t+1}}$ is the unused space of the partially filled bin, in which item a_{t+1} is filled by algorithm A and note that if $L^1_{a_{t+1}} \ge u^{D+1}/2^{D+2}$, hyper-box a_{t+1} can only be packed in one of the above $v(t)$ bins. Let B_i, $i = 1, \cdots, v(t)$, be these partially filled bins. Then the first term of equation (5.93) is less than or equal to:

$$\sum_{i=1}^{v(t)} P\left[v(t) \le n^{1/2}, U_{a_{t+1}} \le \delta \textstyle\prod_{d=2}^{D} L^d_{a_{t+1}} \mid \text{ pack } a_{t+1} \text{ into } B_i \ \right]. \tag{5.94}$$

If algorithm A packs hyper-box a_{t+1} into bin B_i and leaves unused space less than or equal to $\delta \prod_{d=2}^{D} L^d_{a_{t+1}}$, then there exists a positive number $R_i \ge \delta$ such that $R_i - \delta \le$

$L^1_{a_{t+1}} \leq R_i$. Therefore, (5.94) is less than or equal to:

$$\sum_{i=1}^{v(t)} P\big[v(t)\leq n^{1/2}, R_i-\delta\leq L^1_{a_{t+1}}\leq R_i |\ \text{ pack } a_{t+1} \text{ into } B_i\ \big]. \tag{5.95}$$

Using that $L^1_{a_{t+1}}$ follows uniform distribution $U(0,u]$ and $\delta = u^{D+1}n^{-1/2}/2^{D+2}$, we get that (5.95) is less than or equal to:

$$n^{1/2}\frac{\delta}{u} = \frac{u^D}{2^{D+2}}. \tag{5.96}$$

Wrapping up (5.92), (5.93), (5.94), (5.95), (5.96), we get that the second term of (5.90) as:

$$P\left[L^1_{a_{t+1}} \geq \frac{u^{D+1}}{2^{D+2}}, U_{a_{t+1}} \leq \delta \prod_{d=2}^{D} L^d_{a_{t+1}}\right] \tag{5.97}$$

$$\leq \quad \frac{u^D}{2^{D+2}} + P\left[v(t)\geq n^{1/2}\right]. \tag{5.98}$$

Combining the above inequality (5.98) with (5.91), (5.90), and (5.89), we get:

$$P\left[U_{a_{t+1}} \leq \delta \prod_{d=2}^{D} L^d_{a_{t+1}}\right] \leq \frac{u^D}{2^{D+1}} + P\left[v(t)\geq n^{1/2}\right]. \tag{5.99}$$

□

Applying (5.87) and (5.88) into (5.85), we get:

$$E\left[\omega(t)\right] \tag{5.100}$$

$$\geq \quad \delta\left(\frac{u}{2}\right)^{D-1}\left(\frac{tu^D}{2^{D+1}} - \sum_{s=0}^{t-1} P\left[v(s)\geq n^{1/2}\right]\right). \tag{5.101}$$

Next, we discuss how the value of $\sum_{s=0}^{n-1} P[v(s)\geq n^{1/2}]$ affects the expected waste $E[\omega(t)]$ at time step t. On one hand, if it is less than or equal to $nu^D/(3\cdot 2^{D+1})$, applying $\delta = u^{D+1}n^{-1/2}/2^{D+2}$ into (5.101) will imply that when $t \geq n/2$:

$$E\left[\omega(t)\right] \quad \geq \quad \delta\left(\frac{u}{2}\right)^{D-1}\left(\frac{tu^D}{2^{D+1}} - \frac{nu^D}{3\cdot 2^{D+1}}\right) \tag{5.102}$$

$$= \frac{u^{3D} n^{1/2}}{3 \cdot 2^{3D+3}}. \tag{5.103}$$

Note that $\omega(t) \geq 0$. (5.103) further implies:

$$E\left[\frac{1}{n}\sum_{t=1}^{n}\omega(t)\right] \geq E\left[\frac{1}{n}\sum_{t=n/2}^{n}\omega(t)\right] = \frac{u^{3D} n^{1/2}}{3 \cdot 2^{3D+4}}. \tag{5.104}$$

On the other hand, if $\sum_{s=0}^{n-1} P\left[v(s) \geq n^{1/2}\right] \geq \frac{nu^D}{3 \cdot 2^{D+1}}$, we have:

$$E\left[\frac{1}{n}\sum_{t=1}^{n}\omega(t)\right] \tag{5.105}$$

$$= E\left[\frac{1}{n}\sum_{t=1}^{n} E\left[\omega(t) \mid \mathcal{L}^1_{a_{t+1}}, \cdots, \mathcal{L}^d_{a_{t+1}}\right]\right]. \tag{5.106}$$

By the definition of random variable $v(t)$, we have:

$$E\left[\omega(t) \mid \mathcal{L}^1_{a_{t+1}}, \cdots, \mathcal{L}^d_{a_{t+1}}\right] \tag{5.107}$$

$$\geq P\left[v(t) \geq n^{1/2}\right] n^{1/2} \frac{u^{D+1}}{2^{D+2}} \prod_{d=2}^{D} L^d_{a_{t+1}}. \tag{5.108}$$

Applying (5.108) into (5.106) and noting that $L^d_{a_{t+1}}$ follows uniform distribution $U(0, u]$, as well as $\sum_{s=0}^{n-1} P\left[v(s) \geq n^{1/2}\right] \geq nu^D/(3 \cdot 2^{D+1})$, we get:

$$E\left[\frac{1}{n}\sum_{t=1}^{n}\omega(t)\right] \geq \frac{u^{3D} n^{1/2}}{3 \cdot 2^{3D+2}}. \tag{5.109}$$

Combining (5.104) and (5.109), we get:

$$E\left[\frac{1}{n}\sum_{t=1}^{n}\omega(t)\right] \geq \frac{u^{3D} n^{1/2}}{3 \cdot 2^{3D+4}}, \tag{5.110}$$

which implies that $E[\omega(n)]$ is $\Omega(n^{1/2})$. $\square$

According to our results, the distribution $\mathcal{F}$ of the edge lengths of arriving hyperbox-items affects the optimal expected waste of hyperbox packing. In the case of uniform distribution, discrete distribution $U\{J, B\}$ has asymptotically smaller expected waste than continuous distribution $U(0, u]$. We next discuss the expected

waste of a hyperbox packing with the mixed above two distributions, under which edge lengths in some dimensions follow discrete distribution $U\{J, B\}$ while others follow continuous distribution $U(0, u]$, where $u = J/B$. Without loss of generality, we assume that the first D_c dimensions follow continuous distribution $U(0, u]$. By applying the similar proof as in theorem 26, we may get the following corollary.

Corollary 28 *Given a D-dimensional hypercube packing problem, if the edge lengths of arriving hyper-boxes in first D_c, $D > D_c \geq 1$, dimensions follow continuous distribution $U(0, J/B]$, $J \leq B$, and the edge lengths in the rest $D - D_c$ dimensions follow discrete distribution $U\{J, B\}$, then the optimal expected waste of this hypercube packing is at least $EW_n^{OPT} = \Omega(n^{1/2})$.*

CHAPTER VI

SUMMARY AND FUTURE WORK

A. Summary

In this dissertation, we mainly focused on extending 1D bin packing algorithms to hyperbox packing and analyzed the performance of the extended algorithms. Because of the complexity of the problem, we currently limited the items' and bins' sizes to be discrete. The dissertation mainly consists of 5 chapters.

In the first chapter, we briefly introduced the classical bin packing problem and its variants and explained the motivation of studying the problem. We stated the complexity of the problem, illustrated a few real-world applications for the problem, and showed the importance of extending classical 1D packing algorithm to hyperbox packing.

In the second chapter, we reviewed the related past works on classical 1D packing algorithms, which included Next Fit (NF), First Fit (FF), Best Fit (BF), Sum of Squares (SS), First Fit Decreasing (FFD), and Best Fit Decreasing (BFD). After reviewing how the algorithms work, we revisited the worst-case performance of the algorithms via metric worst-case performance ratio, and briefly showed lower-bounds and upper-bounds of the ratios. The techniques and notations used in this section are referenced in chapter IV. We also reviewed the past results on average-case analysis of classical 1D packing algorithms.

In the third chapter, we presented a framework algorithm that converted the problem of hyperbox packing to 1D packing. The framework took two inputs: a 1D packing algorithm and a hyperbox packing instance and output a hyperbox packing algorithm. In this chapter, we presented notations and definitions, listed the key

points, specified in detail how the framework works, and illustrated some examples of framework algorithms.

Since the framework significantly enriched the family of hyperbox packing algorithms, the analysis of the performance of the algorithms became an issue. In chapter IV, we focused on framework-based algorithms ENF, ESS, EFF, EBF, EFFD, and EBFD that are the extensions of 1D algorithms Next Fit, Sum of Squares, First Fit, Best Fit, First Fit Decreasing, and Best Fit Decreasing. We also used metric **worst-case performance ratio**, and analyzed the worst-case performance of the algorithms from the following three perspectives. First, we focused on studying the relationship between the worst-case performance ratio of 1D algorithms and that of extended algorithms, and revealed that the ratios were closely related if certain conditions were met. Secondly, we used strip optimal algorithm (SOPT) as a baseline, compared how well the extended algorithms perform compared to SOPT, and showed the lower bounds and upper bounds for the ratios based on SOPT. Lastly, we used the optimal algorithm as a baseline, and estimated the worst-case performance ratio for algorithms EFF, EBF, EFFD, and EBFD.

Chapter V mainly dealt with the average-case analysis of hyperbox packing algorithms by using metric **expected waste**, and consisted of the following four parts. The first part was related to optimal expected waste, which is the expected waste for optimal algorithms. We showed that optimal expected waste for hyperbox packing asymptotically took three forms depending on the hyperbox-items' size distribution and hyperbox-boxes size. In the first part, we also showed an extended version of the perfect packing theorem, and that optimal expected waste was $\Theta(1)$ when the hyperbox-items' edge lengths were independently drawn from some discrete uniform distribution. In the second part, we presented two theoretical hyperbox packing algorithms, and showed that their expected waste was $\Theta(\sqrt{n})$ if optimal expected waste

was $\Theta(1)$ or $\Theta(\sqrt{n})$. The third part focused on the analysis of the expected waste for the framework-based algorithms ESS, EBF, and EFF. The last part of the chapter dealt with the optimal expected waste for 2D bin packing where hyperbox-items' edge lengths were continuous.

B. Future Work

Future work on our framework could include the following aspects. First of all, our current framework is limited to discrete packing, where hyperbox-items' sizes are discrete. Future work may consider extending the current framework to continuous hyperbox packing, which could be done by converting items edge-lengths from continuous to discrete. For continuous edge length x, $x \in (0, 1]$, of hyperbox-items, we could use some rounding technique to make the conversion happen. For example, we may first choose a very large integer M, and round up the edge-length x with the following function:

$$r(x) = \begin{cases} 1/2^M & \text{if } x \in (0, 1/2^M] \\ 1/2^i & \text{if } x \in (1/2^{i+1}, 1/2^i] \text{ and } 1 \leq i < M \end{cases} . \tag{6.1}$$

Since we round up the edge length and increase the to-be-packed items' sizes, the framework algorithm for the discrete items should also work for the continuous items. Then, the continuous bin packing problem becomes discrete packing, and we may apply our framework to construct an algorithm for it. The challenge of extending the framework to continuous packing is to find right rounding functions, and compare how the rounding functions affect the performance of continuous framework-based algorithms. Analyzing the worst-case and the average-case performance for continuous framework algorithms is also much harder than the analysis in discrete framework packing.

Recall that bin packing with rotation that allows the rotation of hyperbox-items is one of the variants of classical bin packing problems. Our current framework does not allow the rotation of hyperbox-items, and future work may consider extending the framework to this variant or any other variants as well. The challenge of allowing rotation in framework-based algorithms is to find proper heuristic in rotation process. Analyzing the performance of those framework-based algorithms with rotation will be very difficult.

In the worst-case analysis of framework-based algorithms EFFD, we estimated the upper bound of worst-case performance ratio by using the following equation:

$$R_{EFFD}^{\infty} = \sup_{B \in \mathcal{R}} \limsup_{OPT(L) \to \infty} \frac{EFFD(L, \vec{B})}{SOPT(L, \vec{B})} \frac{SOPT(L, \vec{B})}{OPT(L, \vec{B})} \tag{6.2}$$

where $EFFD(L, \vec{B})$, $SOPT(L, \vec{B})$, and $OPT(L, \vec{B})$ are the number of bins used by algorithms EFFD, SOPT, and OPT respectively w.r.t packing instance L. By taking advantage of the above equation, we reduced the complexity of the problem by estimating the upper bounds of $\sup_{B \in \mathcal{R}} \limsup_{SOPT(L) \to \infty} EFFD(L, \vec{B})/SOPT(L, \vec{B})$, and $\sup_{B \in \mathcal{R}} \limsup_{OPT(L) \to \infty} SOPT(L, \vec{B})/OPT(L, \vec{B})$ respectively. Since we did not do the case analysis for EFFD, the upper bound may not be tight enough, and future work may consider approaching the upper bound via case analysis and improving the upper bounds for EFFD and EBFD. Note that doing case analysis for 1D FFD algorithm is already very difficult [6], [52], and Johnson [52] took about 100 pages in his dissertation for the case analysis. Case analysis in multiple-dimensional hyperbox packing would be much harder.

In the worst-case analysis of framework-based algorithm EFF and EBF, we estimated the upper bound of worst-case performance ratio by case analysis, but because of the complexity of the analysis, we only got results for 2D hyperbox packing. Future work may consider a simpler case analysis method to reduce the complexity of

the problem, and getting results for higher dimension hyperbox packing. Even in 2D hyperbox packing, the upper bound could be possibly improved in the future work.

In the average-case analysis of hyperbox packing, we estimated the optimal expect waste when the edge lengths of hyperbox-items independently follow discrete uniform distribution. In the future work, we may consider some other distributions for the edge length, or consider the hyperbox-items sizes' distribution where the edge-lengths' distributions are correlated. Note that we converted the average-case analysis problem to the counterpart problem in 1D packing in this dissertation, and it significantly reduced the complexity. When future work considers the case where edge lengths distributions are not independent, it will be hard to re-use the results in 1D packing and challenging to do case analysis in hyperbox packing for the average-case analysis.

REFERENCES

[1] S. Alberts, M. Mitzenmacher, "Average-Case Analyses of First Fit and Random Fit Bin Packing," in *ACM SODA*, 1998, pp. 290-299.

[2] A. C. F. Alvim, C. C. Ribeiro, F. Glover, and D. J. Aloise, "A Hybrid Improvement Heuristic for the One-Dimensional Bin Packing Problem," *Journal of Heuristics*, vol. 10, no. 2, pp. 205-229, March 2004.

[3] E. Anderson, S. Spence, R. Swaminathan, M. Kallahalla, and Q. Wang, "Quickly Finding Near-Optimal Storage Designs," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 4, pp. 337-374, November 2005.

[4] E. Asgeirsson, U. Ayesta, E. Coffman, J. Etra, P. Momcilovic, D. Phillips, V. Vokhshoori, Z. Wang, and J. Wolfe, "Closed On-Line Bin Packing," *Acta Cybernetica*, vol. 15, no. 3, pp. 361-367, September 2002.

[5] Y. Azar and O. Regev, "On-Line Bin-Stretching," *Theoretical Computer Science*, vol. 268, no. 1, pp. 17-41, October 2001.

[6] B. S. Baker, "A New Proof for the First-Fit Decreasing Bin-Packing Algorithm," *Journal of Algorithms*, vol. 6, pp. 49-70, 1983.

[7] N. Bansal and M. Sviridenko, "New Approximability and Inapproximability Results for 2-dimensional Bin Packing," in *ACM SODA*, 2004, pp. 196-203.

[8] N. Bansal, J.R. Correa, C. Kenyon, and M. Sviridenko, "Bin Packing in Multiple Dimensions: Inapproximability Results and Approximation Schemes," *Mathematics of Operations Research*, vol. 31, no. 1, pp. 31-49, 2006.

[9] A. Bar-Noy, R. E. Ladner, and T. Tamir, "Windows Scheduling as a Restricted Version of Bin Packing," in *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004, pp. 224-233.

[10] A. Bar-Noy, R. E. Ladner, T. Tamir, and T. VanDeGrift, "Windows Scheduling of Arbitrary Length Jobs on Parallel Machines," in *Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2005, pp. 56-65.

[11] A. Bar-Noy, R. E. Ladner, and T. Tamir, "Windows Scheduling as a Restricted Version of Bin Packing," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 3, pp. 28-28, August 2007.

[12] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems," *IEEE Design and Test of Computers*, vol. 17, no. 1, pp. 68-83, January 2000.

[13] W. Bein, J. R. Correa, and Xin Han, "A Fast Asymptotic Approximation Scheme for Bin Packing with Rejection," *Theoretical Computer Science*, vol. 393, no. 1-3, pp. 14-22, March 2008.

[14] R. Berghammer and F. Reuter, "A Linear Approximation Algorithm for Bin Packing with Absolute Approximation Factor 3/2," *Science of Computer Programming*, vol. 48, no. 1, pp. 67-80, July 2003.

[15] P. Berman, J. Jeong, S. P. Kasiviswanathan, and B. Urgaonkar, "Packing to Angles and Sectors," in *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, 2007, pp. 171-180.

[16] S. Bischof and E. W. Mayr, "On-Line Scheduling of Parallel Jobs with Runtime

Restrictions," *Theoretical Computer Science*, vol. 268, no. 1, pp. 67-90, October 2001.

[17] M. A. Boschetti, "New Lower Bounds for the Three-Dimensional Finite Bin Packing Problem," *Discrete Applied Mathematics*, vol. 140, no. 1-3, pp. 241-258, May 2004.

[18] J. Boyan and A. W. Moore, "Learning Evaluation Functions to Improve Optimization by Local Search," *The Journal of Machine Learning Research*, vol. 1, pp. 77-112, September 2001.

[19] J. Boyar, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen, "The Competitive Ratio for On-Line Dual Bin Packing with Restricted Input Sequences," *Nordic Journal of Computing*, vol. 8, no. 4, pp. 463-472, December 2001.

[20] M. Brehob, E. Torng, and P. Uthaisombut, "Applying Extra-Resource Analysis to Load Balancing," in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, 2000, pp. 560-561.

[21] A. Caprara, "Packing 2-dimensional Bins in Harmony," in *IEEE FOCS*, 2002, pp. 490-499.

[22] A. Caprara and U. Pferschy, "Modified Subset Sum Heuristics for Bin Packing," *Information Processing Letters*, vol. 96, no. 1, pp. 18-23, October 2005.

[23] A. Caprara, "Packing d-Dimensional Bins in d Stages," *Mathematics of Operations Research*, vol. 33, no. 1, pp. 203215, February 2008.

[24] E. Chang, W. Wang, and M. S. Kankanhalli, "Multidimensional On-Line Bin Packing: An Algorithm and its Average-Case Analysis," *Information Processing Letters*, vol. 48, pp. 121-125, 1993.

[25] F. Chung, R. Graham, R. Bhagwan, S. Savage, and G. M. Voelker, "Maximizing Data Locality in Distributed Systems," *Journal of Computer and System Sciences*, vol. 72 no. 8, pp. 1309-1316, December 2006.

[26] B. Codenotti, G. D. Marco, M. Leoncini, M. Montangero, M. Santini, "Approximation Algorithms for a Hierarchically Structured Bin Packing Problem," *Information Processing Letters*, vol. 89, no. 5, pp. 215-221, March 2004.

[27] E. G. Coffman, C. A. Courcoubetis, M. R. Garey, D. S. Johnson, L. A. Mcgeoch, P. W. Shor, R. R. Weber, and M. Yannakakis, "Fundamental Discrepancies between Average-case Analyses Under Discrete and Continuous Distributions - A Bin Packing Case Study," in *ACM STOC*, 1991, pp. 230-240.

[28] E. G. Coffman, D. S. Johnson, G. S. Lueker, and P. W. Shor, "Asymptotic Probabilistic Analysis of Packing and Related Partitioning Problems," *Statistical Sciences*, vol. 8, pp. 40-47, 1993.

[29] E. G. Coffman, Jr, D. S. Johnson, P. W. Shor, and R. R. Weber, "Markov Chains, Computer Proofs, and Best Fit Bin Packing," in *ACM STOC*, 1993, pp. 412-421.

[30] E. G. Coffman, M. R. Garey, and D. S. Johnson, "Bin Packing Approximation Algorithms: A Survey," in *Approximation Algorithms for NP-Hard Problems*, D. Hochbaum (ed.), PWS Publishing Co. Boston, MA, pp. 46-93, 1996.

[31] E. G. Coffman, Jr., C. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor, R. R. Weber, and M. Yannakakis, "Bin Packing with Discrete Item Sizes, Part I: Perfect Packing Theorems and the Average Case Behavior of Optimal Packings," *SIAM Journal on Applied Mathematics (SIAP)*, vol. 13, pp. 384-402, 2000.

[32] E. G. Coffman Jr., C. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor, R.

R. Weber, and M. Yannakakis, "Perfect Packing Theorems and the Average-case Behaviour of Optimal and Online Bin Packing," *SIAM Review*, vol. 44, no. 1, pp. 95-108, 2002.

[33] C. Courcoubetis and R. R. Weber, "A Bin-Packing System for Objects with Sizes from a Finite Set: Necessary and Sufficient Conditions for Stability and Some Applications," in *Proceedings of the 25th IEEE Conference on Decision and Control*, 1986, pp. 1686-1691.

[34] C. Courcoubetis and R. R. Weber. "Necessary and Sufficient Conditions for Stability of a Bin-packing System," *Journal of Applied Probability*, vol. 23, pp. 989-999, 1986.

[35] C. Courcoubetis and R. R. Weber, "Stability of On-line Bin Packing with Random Arrivals and Long-run Average Constraints," *Probability in the Engineering and Informational Sciences*, vol. 4, pp. 447-460, 1990.

[36] T. G. Crainic, G. Perboli, M. Pezzuto, and R. Tadei, "New Bin Packing Fast Lower Bounds," *Computers and Operations Research*, vol. 34, no. 11, pp. 3439-3457, November 2007.

[37] J. Csirik, D. S. Johnson, C. Kenyon, P. W. Shor, and R. R. Weber, "A Self Organizing Bin Packing Heuristic," in *Proceeding of the 1999 Workshop on Algorithm Engineering and Experimentation*, 1999, pp. 246-265.

[38] J. Csirik, D. S. Johnson, C. Kenyon, J. B. Orlin, P. W. Shor, and R. R. Weber, "On the Sum-of-Squares Algorithm for Bin Packing," in *ACM STOC*, 2000, pp. 208-217.

[39] J. Csirik, D. S. Johnson, and C. Kenyon, "On the Worst-Case

Performance of the Sum-of-Squares Algorithm for Bin Packing," http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0509031, 2005.

[40] M. Dawande, S. Kumar, and C. Sriskandarajah, "Performance Bounds of Algorithms for Scheduling Advertisements on a Web Page," *Journal of Scheduling*, vol. 6, no. 4, pp. 373-393, July 2003.

[41] X. Décoret, F. Durand, F. X. Sillion, and J. Dorsey, "Billboard Clouds for Extreme Model Simplification," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 3, pp. 689-696, July 2003.

[42] G. Dósa and Y. He, "Bin Packing Problems with Rejection Penalties and Their Dual Problems," *Information and Computation*, vol. 204, no. 5, pp. 795-815, May 2006.

[43] A. Ejnioui and N. Ranganathan, "Multi-Terminal Net Routing for Partial Crossbar-Based Multi-FPGA Systems," in *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, 1999, pp. 176-185.

[44] A. Ejnioui and N. Ranganathan, "Multiterminal Net Routing for Partial Crossbar-Based Multi-FPGA Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 1, pp. 71-78, February 2003.

[45] A. El-Haj-Mahmoud, A. S. AL-Zawawi, A. Anantaraman, and E. Rotenberg, "Virtual Multiprocessor: an Analyzable, High-Performance Architecture for Real-Time Computing," in *Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2005, pp. 213-224.

[46] L. Epstein and L. M. Favrholdt, "On-Line Maximizing the Number of Items

Packed in Variable-Sized Bins," *Acta Cybernetica*, vol. 16, no. 1, pp. 57-66, January 2003.

[47] L. Epstein and R. V. Stee, "Optimal Online Bounded Space Multidimensional Packing," in *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004, pp. 214-223.

[48] M. R. Garey, R. L. Graham, and J. D. Ullman, "Worst-Case Analysis of Memory Allocation Algorithm," in *ACM STOC*, 1972, pp. 143-150.

[49] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yaho, "Resource Constrained Scheduling as Generalized Bin Packing," *Journal Combinatorial Theory*, vol. 21, pp. 257-298, 1976.

[50] M. N. Garofalakis, Y. E. Ioannidis , and B. Özden, "Resource Scheduling for Composite Multimedia Objects," in *Proceedings of the 24rd International Conference on Very Large Data Bases*, 1998, pp. 74-85.

[51] I. P. Gent, "Heuristic Solution of Open Bin Packing Problems," *Journal of Heuristics*, vol. 3, no. 4, pp. 299-304, March 1998.

[52] D. S. Johnson, "Near-Optimal Bin Packing Algorithms," *PhD thesis, Massachusetts Institute of Technology, Department of Mathematics*, Cambridge, 1973.

[53] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, 'Worst-Case Performance bounds for Simple One-dimensional Packing Algorithms," *SIAM Journal on Computing*, vol 3, pp. 256-278, 1974.

[54] D. Karger and K. Onak, "Polynomial Approximation Schemes for Smoothed and Random Instances of Multidimensional Packing Problems," in *Proceedings*

of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 1207-1216.

[55] R. M. Karp, M. Luby, and A. Marchetti, "A Probabilistic Analysis of Multidimensional Bin Packing Problems," in *IEEE STOC*, 1984, pp. 289-298.

[56] C. Kenyon, Y. Rabani, and A. Sinclair, "Biased Random Walks, Lyapunov Functions, and Stochastic Analysis of Best Bit Bin Packing," in *ACM SODA*, 1996, pp. 351-358.

[57] R. Kleinberg and T. Leighton, "Consistent Load Balancing via Spread Minimization," in *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of computing*, 2003, pp. 565-574.

[58] V. Liberatore, "Scheduling Jobs Before Shut-Down," *Nordic Journal of Computing*, vol. 7, no. 3, pp. 204-226, 2000.

[59] K. Loh, B. Golden, and E. Wasil, "Solving the One-Dimensional Bin Packing Problem with a Weight Annealing Heuristic," *Computers and Operations Research*, vol. 35, no. 7, pp. 2283-2291, July 2008.

[60] P. Manyem, R. L. Salt, and M. S. Visser, "Approximation Lower Bounds in Online LIB Bin Packing and Covering," *Journal of Automata, Languages and Combinatorics*, vol. 8, no. 4, pp. 663-674, April 2003.

[61] F. K. Miyazawa and Y. Wakabayashi, "Cube Packing," *Theoretical Computer Science*, vol. 297, no. 1-3, pp. 355-366, March 2003.

[62] F. K. Miyazawa, Y. Wakabayashi, "Two- and Three-Dimensional Parametric Packing," *Computers and Operations Research*, vol. 34, no. 9, pp. 2589-2603, September 2007.

[63] T. Osogami and H. Okano, "Local Search Algorithms for the Bin Packing Problem and Their Relationships to Various Construction Heuristics," *Journal of Heuristics*, vol. 9, no. 1, pp. 29-49, January 2003.

[64] "Partition Function P," http://mathworld.wolfram.com/PartitionFunctionP.html, January 2007.

[65] X. Qi, "Note: A Note on Worst-Case Performance of Heuristics for Maintenance Scheduling Problems," *Discrete Applied Mathematics*, vol. 155, no. 3, pp. 416-422, February 2007.

[66] J. Remy, "Resource Constrained Scheduling on Multiple Machines," *Information Processing Letters*, vol. 91, no. 4, pp. 177-182, August 2004.

[67] T. Risse, K. Aberer, A. Wombacher, M. Surridge, and S. Taylor, "Configuration of Distributed Message Converter Systems," *Performance Evaluation*, vol. 58, no. 1, pp. 43-80, October 2004.

[68] M. Scharbrodt, T. Schickinger, A. Steger, "A New Average Case Analysis for Completion Time Scheduling," *Journal of the ACM (JACM)*, vol. 53, no. 1, pp. 121-146, January 2006.

[69] S. S. Seiden, R. V. Stee, "New Bounds for Multi-Dimensional Packing," in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 486-495.

[70] S. S. Seiden, "On the Online Bin Packing Problem," *Journal of the ACM (JACM)*, vol. 49, no. 5, pp. 640-671, September 2002.

[71] Y. Shi and D. Ye, "Online Bin Packing with Arbitrary Release Times," *Theoretical Computer Science*, vol. 390, no. 1, pp. 110-119, January 2008.

[72] P. W. Shor, "The Average-Case Analysis of Some On-line Algorithms for Bin Packing," *Combinatorica*, vol. 6, no. 2, pp. 179-200, 1986.

[73] T. Tambouratzis, "Incremental Bin Packing, Neural," *Parallel and Scientific Computations*, vol. 9, no. 2, pp. 175-186, June 2001.

[74] S. K. Williams and M. J. Magazine, "Heuristic Approaches for Batching Jobs in Printed Circuit Board Assembly," *Computers and Operations Research*, vol. 34, no. 7, pp. 1943-1962, July 2007.

[75] E. C. Xavier and F. K. Miyazawa, "The Class Constrained Bin Packing Problem with Applications to Video-on-Demand," *Theoretical Computer Science*, vol. 393, no. 1-3, pp. 240-259, March 2008.

[76] E. C. Xavier and F. K. Miyazawa, "A One-Dimensional Bin Packing Problem with Shelf Divisions," *Discrete Applied Mathematics*, vol. 156, no. 7, pp. 1083-1096, April 2008.

[77] G. Xiaodong, C. Guoliang, and X. Yinlong, "Deep Performance Analysis of Refined Harmonic Bin Packing Algorithm," *Journal of Computer Science and Technology*, vol. 17, no. 2, pp. 213-218, March 2002.

[78] D. Xu, K. Sun, and H. Li, "Parallel Machine Scheduling with Almost Periodic Maintenance and Non-Preemptive Jobs to Minimize Makespan," *Computers and Operations Research*, vol. 35, no. 4, pp. 1344-1349, April 2008.

[79] M. Yue, "A Simple Proof of the Inequality $FFD(L) \leq \frac{11}{9}OPT(L) + 1, \ \forall L$, for the FFD Bin-Packing Algorithm.," *Acta Math. App. Sinica*, vol. 7, pp. 321-331, 1991.

[80] L. Zeng, H. L. Ong, K. M. Ng, and S. B. Liu, "Two Composite Methods for Soft Drink Distribution Problem," *Advances in Engineering Software*, vol. 39, no. 5, pp. 438-443, May 2008.

[81] P. Zheng and L. M. Ni, "EMPOWER: A Cluster Architecture Supporting Network Emulation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 7, pp. 617-629, July 2004.

VITA

Xiafeng Li received his B.S. degree in mechanical engineering from Jingdezhen Ceramic Institute, China, in 1997 and M.S. degree in computer science and engineering from Shanghai Jiaotong University, China in 2001. He graduated with a Ph.D. in computer science from Texas A&M University December 2008.

His research interests include bin packing algorithms and approximation algorithms. He may be contacted at:

Xiafeng Li C/O Dongling Zhan
59 Xinjing Rd, APT. 201,
Fengjing, Jinshan District,
Shanghai, 201501, P. R. China

The typist for this dissertation was Xiafeng Li.