Competitive Algorithms in Discrete Optimization

DSc Dissertation

Gábor Galambos

Institute of Applied Natural Sciences

University of Szeged

Szeged

2016

Contents

1 Introduction

In the thesis we consider discrete combinatorial optimization problems. The investigation of bin packing and scheduling problems has started in the late sixties. It turned out soon that they belong to the class of $\mathcal{NP}$-hard problems [56]. Although the capacity and the speed of computers has significantly increased in the last decades, and so the exact solvable problem-sizes have became larger, the research still focused on finding polynomial time complexity algorithms with near optimal solutions.

The field of data compression is also very wide including data that must be handled during text, voice, and picture processing. Text compression contrasts strikingly with the other – picture and voice – compression procedures, since here the loss of information is not permitted during the compression-decompression processes. Although there exists a shortest path graph model to solve this problem, and so it is solvable in polynomial time, the sizes of some problems require online solutions with the help of various heuristic algorithms.

The problems considered are very diversified, and many papers examine their variations. The aim of the thesis is not to review all versions of the basic problems. In each section we will exactly define the problems to be investigated in the corresponding part of the thesis. Here we refer only to the surveys [26], [27], [54], [63], and [18], where a wide variety of the problems are studied, and the recent results are also reviewed.

There are a lot of practical problems which are equivalent with the models considered in the thesis. Mostly, after defining the problems we refer to practical motivations. These examples show the practical significance of these problems. It is worth to investigate these problems – and their solutions – in different ways: it is very important to decide whether a problem can be solved in polynomial time, since this is the basis of the use of approximation algorithms. If we seek feasible solutions for a practical problem, then applying a quicker, more efficient approximation algorithm may result in various benefits. If we have quick, efficient algorithms, finding the optimal one within an algorithm-class may close the research and the given algorithm may cover more surety for the applications.

Considering a practical problem we have different possibilities for the input data. It can happen that we already know every relevant data that we need during the solution at the starting time. In this case we can apply *offline algorithms*. In some cases we get the input in snacks – sometimes the data arrive singly – and the algorithms need to be decided in the actual status without knowing anything about the subsequent part of the input. In this case we speak about *online* problems and we use *online algorithms*. If the problem allow us to look ahead in the input, or we can collect a part of the input data without any decision, or we can change – partially – our earlier decisions, then we speak about *semi-online algorithms*.

In the following we first give a short overview about the problems considered in the thesis.

1.1 Bin packing problems

Problem 1.1.1. One-dimensional bin packing problem. *Let $L = \{a_1, a_2, \ldots, a_n\}$ be a list of n items, where $a_i \in (0, 1]$, $i = 1, \ldots, n$, denotes the size of the item. The task is to assign the items to the minimal number of unit capacity bins, subject to the constraint that the total size of the items assigned to any bin is at most 1.*

Definition 1.1. *If the sizes of the items are chosen from the interval $(0, \frac{1}{r}]$ for some integer $r \geq 2$, then we speak about r-*parametric *(or simply,* parametric*) bin packing problem.*

Cutting bars into smaller demands while minimizing the waste is a typical example for practical motivation. However, to schedule advertisements into fixed length "time-windows" results in the same model. Similarly, to transfer money from a bank-account taking into account with a daily limit can be described also with this model.

Problem 1.1.2. One-dimensional dual bin packing problem. *Given a list $L = \{a_1, a_2, \ldots, a_n\}$ of n items, where $a_i \in (0, 1]$, $i = 1, \ldots, n$, denotes the size of the item. The task is to assign the items to the maximal number of unit capacity bins, subject to the constraint that the total size of the items assigned to any bin is at least 1. This problem also called as bin covering problem.*

Potential practical applications are the packing of canned goods so that each can contains at least its advertised net weight, or the stimulation of economic activity during recession by allocating tasks to a maximum number of factories, all working at or beyond the minimal feasible level.

Problem 1.1.3. Two-dimensional rectangular bin packing problem. *Given a list of $L = \{a_1, a_2, \ldots, a_n\}$ of n items, which are defined with ordered pair of sizes $(w(a_i), h(a_i))$ where $w(a_i) \in (0, 1]$ and $h(a_i) \in (0, 1]$ is the width and the height of a_i, respectively. We are also given rectangular bins with sizes $W = 1$ and $H = 1$. The task is to pack the small rectangles into minimal number of bins such that the sides of the items are parallel to the corresponding sides of the bins (i.e no rotation allowed), and no two rectangles in a bin overlap.*

Two-dimensional cutting stock problems result in the same model. Cutting furniture tympans (or panes) into smaller pieces of rectangular demands are the typical examples for practical application.

1.2 Scheduling problems

Problem 1.2.1. Parallel machine problem. *There are given n jobs $J_1, \ldots, J_n$ and m identical machines $M_1, \ldots, M_m$. Each job J_i has a fixed processing time p_i. The jobs may be processed in any order, but preemption (interruption) is not allowed. The goal is to minimize the makespan, i.e. the maximum completion time over all jobs in a schedule.*

Problem 1.2.2. Open shop problem. *An open shop problem consists of m machines $M_1, \ldots, M_m$ and n jobs $J_1, \ldots J_n$. Each job J_i consists of m operations $O_{i1}, \ldots O_{im}$, and O_{ij} has to be processed on machine M_j for p_{ij} time units without preemption. One machine can process at most one operation at a time and one job can be processed by at most one machine at a time. Operations of the same job can be processed in any order.*

Let us see the following example (see [60]). Consider a large garage with specialized shops. A car may require the following works: replace exhausted pipes and muffler, align wheels, and tune up. These three tasks may be carried out in any order. However, since the exhaust system, alignment, and tune up shops are in different buildings, it is not possible to perform two tasks simultaneously.

Problem 1.2.3. Coupled tasks problem. *We are given n jobs, each of them consisting of two distinct tasks (operations). The sequence of these tasks is fixed and also there is a fixed delay time between the two tasks. So, each job i can be described by a triple (a_i, L_i, b_i), where the values represent the processing time of the first task, the delay time between the tasks and the processing time of the second task, respectively. During the delay time the machine is idle so it can process other jobs in this interval. The aim is to schedule n coupled tasks on one machine in such a way that no two tasks overlap, and the latest completion time (also called as makespan, and denoted by $C_{\max}$) of the jobs is minimized. Preemption is not allowed, every subtask has to be processed continuously.*

For the coupled tasks problem we show two military applications. In a *pulsed radar system* for tracking an object or to survey a volume of space a predetermined length of a pulse of electromagnetic energy must be submitted and later the radar echo has to be received. The interval between the transmission and reception of the pulse depends on the distance of the target from the radar or the volume of the space. The radar can process only one task at a time. The objective is usually to minimize the idle time of the radar. *Scheduling take-offs and landings on an aircraft carrier* leads to the same model. A take-off is the first part of a coupled task, while the aircraft is far away it is an idle time for the runway and the landing is the second part of the job. In this application all lengths of first tasks (take-off times) are equal and this is true for the second tasks (landing times) as well.

Definition 1.2. *We will describe the different scheduling problems by using the standard three-fields notation $\alpha \,|\, \beta \,|\, \gamma$, introduced by Graham, Lawler, Lenstra, and Rinnooy Kan in [63]. The first field describes the machine environment, the second the job characteristics and the last one is the optimality criterion.*

1.3 Data compression problem

Problem 1.3.1. Text compression problem. *There is given a dictionary, which consists of pairs – (source-words, code-words) – of strings over two finite alphabets. The dictionary cannot be changed or extended during the encoding–decoding procedure, so it is static. A*

dictionary based compression process divides the source string into substrings – each of them corresponds to some source-word – and substitutes them by code-words from the dictionary. Our aim is to translate (encode) the source-string with the help of dictionary strings into a code–text with minimal length. After a possible data transmission, using the dictionary again, we can decode the encoded text into the original form.

Recent advances in computer technology strongly require large amounts of data to be moved between various components or to be stored in bounded capacity devices. All these operations need data transfers, either between two computers or between two parts of the same computer. Essentially, there are just two possibilities to increase the performance of the transfer: either to use better (and more expensive) hardware or to compress the data before transfer.

1.4 Analysis of approximation algorithms

Constructing approximation algorithms started in the late '60s. The analysis of bin packing and scheduling algorithms played significant role in the algorithms theory. This research was started by the pioneer works of D.S. Johnson [70], [71], and R. Graham [62].

The quality of approximation algorithms is a central question in the algorithm theory. There are several methods to measure the quality of an algorithm: experimental examination, worst case competitive analysis and probabilistic analysis.

In the case of *experimental examination* a set of problem instances are taken and they are solved by an algorithm. In most cases one can only give an upper bound for the optimal solution, so it is hard to decide how far the approximation solution of the given algorithm from the optimum is. Therefore, the experimental evaluation is convenient in cases when we can compare algorithms. Since the efficiency of any algorithm strongly depends on the set of instances chosen, the "freedom" of choice is the weakest point of this type of measurement. The larger the freedom, the more unreliable the result of the experimental estimation.

If we decide to use *worst case competitive analysis,* we are looking for performance guarantees which are valid for any – even for the extreme, so-called "pathological" – instances. Yet, it is important to investigate the extreme examples, since they can reveal the weakest points of a given algorithm. The most frequently used measurements can be defined as follows. Let A be an arbitrary approximation algorithm, and let I be an instance of the given problem. Let $A(I)$ and $\mathrm{OPT}(I)$ denote the solution of A and the optimal solution, respectively.

Definition 1.3. *The absolute competitive ratio for a minimum problem is defined as follows.*

$$R_A = \sup_I \frac{A(I)}{\mathrm{OPT}(I)}.$$

If $R_A = C$, we also say that the algorithm A is absolute C-competitive.

Definition 1.4. *The asymptotic competitive ratio for minimum problems is defined as follows.*

$$R_A^\infty = \limsup_{k\to\infty} \left\{ \max_I \left\{ \frac{A(I)}{k} \middle| \mathrm{OPT}(I) = k \right\} \right\}.$$

In case of maximum problems the definition changes to

$$R_A^\infty = \liminf_{k\to\infty} \left\{ \min_I \left\{ \frac{A(I)}{k} \middle| \mathrm{OPT}(I) = k \right\} \right\}.$$

If $R_A^\infty = C$, we say that the algorithm A is asymptotically C-competitive.

Generally, if the context is clear we will say "C-competitive" in both cases.

To execute a *probabilistic analysis* we have to know the probability distribution of the elements of our instances. Let us choose the input of an instance independently from the same distribution. Knowing the distribution the expected values of the optimal solution and the approximation algorithm can be calculated. Then having the expected values we can compare them. The wider the conditions of the distribution function, the more general the estimations of the concerning theorems.

1.5 Outline of the thesis

The thesis can be divided into three major parts. Chapters 2-5 investigate different bin packing algorithms. In Chapter 2 we begin with the classical one-dimensional bin packing problem, and we give lower bound for the online case. The given lower bound of $1.5403\ldots$ improves an almost twenty years old result $1.5401\ldots$, ([100], 1992) and recently it is the best one in its class.

The next chapter concentrates on *semi-online* problems. First, we show a new lower bound of $\frac{54}{47}$ for that class of semi-online algorithms which gets items in nonincreasing order. With this result we improved the old lower bound $\frac{8}{7}$, ([29], 1983). Thereafter, in Section 3.2 we introduce a new algorithm and we prove that for bounded space semi-online algorithms it is optimal. We continue our investigations among the semi-online algorithms by giving an asymptotically $\frac{3}{2}$-competitive algorithm for the k-repacking problem, and in the subsequent section we give a lower bound of $1.3871\ldots$ for the same class of the semi-online algorithms.

Using the technique has been introduced in Chapter 2 we give lower bounds for online two-dimensional rectangle packing problems in Chapter 4. First, we show an – almost trivial – lower bound (1.6) given in [47] to present the idea. Since the best one-dimensional algorithm is $1.5888\ldots$ due to Seiden [94] this lower bound settled the open question whether the two-dimensional (2D) online algorithms can be as good as the one-dimensional ones. Based on the paper of Galambos and van Vliet, ([50], 1994) we improve this result. More precisely, we prove that for any A online 2D algorithm $R_A^\infty \geq 1.802\ldots$.

The next chapter of the first part is dealing with the probabilistic analysis of various bin packing algorithms. Firstly, we consider the classical bin packing problem, and we analyse the Next Fit Decreasing algorithm from probabilistic point of view. For the

2D rectangle problem we investigate the probabilistic behaviour of the Hybrid Next Fit algorithm. We close the first part of the thesis by analysing three algorithms for the dual bin packing problem.

The second major block of the thesis deals with scheduling problems. We start with our early result where we presented an algorithm which has a better worst case ratio than Graham's List Scheduling algorithm. For m machines our algorithm has an asymptotic competitive ratio of $2 - \frac{1}{m} - \varepsilon_m$, where $\varepsilon \to 0$ if $m \to \infty$.

This paper encouraged researchers to investigate the problem in more details. Applying our "similarity" definition several improvements were published within a few years (see e.g. [11], [12], [22], [72], [101].)

The next chapter is a complexity result for a special open shop problem. We consider the problem with $p_{i,j} = 1$ for each job, and we suppose that all jobs have release times r_i, due dates d_i, and weights w_i.

We characterize this problem as $O2|p_{ij} = 1, r_i| \sum w_i U_i$, where $U_i = 1$ if the job is late and $U_i = 0$ otherwise. Using graph theoretical tools we give a polynomial time algorithm for this problem.

We close this part by investigating the coupled tasks problem. We consider the identical job problem, i.e. $a_i = a, L_i = L, b_i = b$ for all jobs: $1 \,|\, \text{Coup-Task}, a_i = a, L_i = L, b_i = b \,|\, C_{\max}$. We present an exact algorithm with $O(nr^{2L})$ time-complexity, where $r \leq \sqrt[a-1]{a}$. Although this algorithm is exponential in L, until now this is the best algorithm for this problem.

The data compression block starts with an upper bound for any online algorithm. First, the *Longest Matching* algorithm is analysed for prefix dictionaries. The *Differential Greedy* algorithm was investigated by Katajainen and Raita in [75]. For suffix dictionaries they gave upper bounds. Improving their results we give tight bounds for the asymptotic competitive ratios.

In the next subsection we introduce the *Fractional Greedy* algorithm. We show that for general and prefix dictionaries the worst-case behaviour of this algorithm is similar to the Longest Matching and the Differential Greedy. Giving an – almost tight – upper bound for suffix dictionaries we prove that Fractional Greedy behaves better than Differential Greedy.

In the last part of this block we analyse the *Iterated Longest Fragment First* algorithm which was defined by Schuegraf and Heaps in [93]. They had only experimental results for this algorithm. We investigate the algorithm from asymptotic worst-case point of view and we prove tight bounds for different combinations of the dictionaries.

2 One-dimensional online bin packing problem

For one-dimensional offline bin packing problem it is possible to construct an approximation algorithm with an asymptotic competitive ratio arbitrary close to 1 (see e.g. [99]). For online problem this is not possible. An online algorithm does not know anything about the subsequent items, and this lack of information results in worse behaviour. The main task of an online algorithm is to keep the balance between the good performance right now, and the good performance in the future.

2.1 Lower bounds for online bin packing

Proofs of establishing a lower bound for online algorithms apply the following technique: take a series of lists $L_1, L_2, \ldots, L_k$ carefully, with identical elements each. We denote the concatenated list by $(L_1 L_2 \ldots L_k)$, where the items of L_i are followed by the items of L_{i+1}, $1 \leq i \leq k-1$. Then analyse the performance of online algorithms on the concatenated lists $(L_1 \ldots L_j)$ for every $1 \leq j \leq k$. Based on this idea, Yao was the first establish a lower bound of $\frac{3}{2}$ (see [103]) using three different lists. His result was improved by Liang [79] and Brown [19] independently to about 1.536. Later in [45], Galambos analysed the r-parametric case of the problem.

A simplified proof for the bound 1.536 was given by Galambos and Frenk in [48] using a weighting function technique. Later, van Vliet [100] proved a lower bound 1.54014... for any online algorithm in 1992. He also investigated the parametric case. To prove his result van Vliet considered the solution of a special linear program. The proof is rather complicated and assumes a fair amount of knowledge about linear programming.

All in these proofs the sizes of the items correspond with the values of a special sequence. This sequence was first introduced by Sylvester in [98] (1880) for the case $r = 1$, therefore, we refer to this sequence as *generalized Sylvester sequence.*

Definition 2.1. *For integers $k > 1$ and $r \geq 1$, the generalized Sylvester sequence $m_1^r, \ldots, m_k^r$ can be given by the following recursion.*

$$m_1^r = r + 1, \qquad m_2^r = r + 2, \qquad m_j^r = m_{j-1}^r (m_{j-1}^r - 1) + 1, \text{ for } j = 3, \ldots, k, .$$

m_j^r	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
$j = 1$	2	3	4	5	6
$j = 2$	3	4	5	6	7
$j = 3$	7	13	21	31	43
$j = 4$	43	157	421	931	1807
$j = 5$	1807	24493	176821	865831	3263443

Table 2.1. The first few elements of the generalized Sylvester sequences if $k \leq 5$.

These sequences have the following properties.

$$\sum_{i=j}^{k} \frac{1}{m_i^r} = \frac{1}{m_j^r - 1} - \frac{1}{m_{k+1}^r - 1}, \quad \text{if } j \geq 2,$$

and

$$\frac{r-1}{m_1^r} + \sum_{i=2}^{k} \frac{1}{m_i^r} = 1 - \frac{1}{m_{k+1}^r - 1} \quad \text{if } r \geq 2, .$$

In the above proofs the sizes of the lists were derived from the generalized Sylvester sequences. For example, if $r = 1$, then the sizes are $\frac{1}{2} + \varepsilon$, $\frac{1}{3} + \varepsilon$, $\frac{1}{7} + \varepsilon$, $\frac{1}{43} + \varepsilon, \ldots$. We will use these sequences throughout the thesis to analyse online lower bounds. Similarly, we will allude to the following constants.

$$h_\infty(r) = 1 + \sum_{i=2}^{\infty} \frac{1}{m_i^r - 1}.$$

The first few values of $h_\infty(r) : h_\infty(1) \approx 1.69103$, $h_\infty(2) \approx 1.42312$, $h_\infty(3) \approx 1.30238$.

Generally, to avoid the pilling of indexes we will denote m_j^r by m_j. The Sylvester sequence seemed to be irrefutable in the proofs of different lower bounds on the online field. Therefore, most of the efforts of the research were made to produce better online algorithms. The best known online algorithm is due to Seiden [94] with an asymptotic competitive ratio at most 1.58889.., Between 1992 and 2012 no improvements on the lower bounds were published.

Based on our paper [7] in this section we give an improvement on van Vliet's lower bounds using different series of lists. This section is organized as follows. First, we reformulate the packing pattern technique. Then we show that using this technique the $1.54014\ldots$ lower bound is also achievable with the right choice of the weights. Finally, giving new sequences for the sizes of elements, we consider the parametric case and we improve van Vliet's lower-bounds.

2.1.1 Reformulated packing pattern technique

In [79] a technique was introduced to analyse the lower bounds for online problems. Later we expanded the method in [47], [48] and [50]. All versions allowed only equal length of lists in the construction of the proof. A. van Vliet [101] extended the technique for those lists which have different lengths. Since we will use this basic theorem in our improvements, we discuss the proof in detail. First, we need some preliminaries and we also introduce some notations.

For an arbitrary large integer n, we consider lists $L_1^n, L_2^n, \ldots, L_k^n$ of lengths $n_j = c_j n$ for fixed integers c_j, $j = 1, 2, \ldots, k$. Sublist L_j^n contains n_j pieces of equally sized items. We assume that the size of an item does not depend on n. To simplify our presentation we write L_j instead of L_j^n.

As a further notation, let nU_j be an upper bound for the optimal packing of the concatenated list $(L_1L_2\dots L_j)$, i.e.

$$U_j \geq \frac{\mathrm{OPT}(L_1L_2\dots L_j)}{n}, \quad 1 \leq j \leq k. \tag{1}$$

Using the definition of the asymptotic competitive ratio it is clear that for any online algorithm A

$$R_A^\infty \geq \max_{1\leq j\leq k} \limsup_{n\to\infty} \frac{A(L_1L_2\dots L_j)}{\mathrm{OPT}(L_1L_2\dots L_j)} \geq \max_{1\leq j\leq k} \limsup_{n\to\infty} \frac{A(L_1L_2\dots L_j)}{n\cdot U_j}. \tag{2}$$

In order to establish the theorems we introduce the definition of packing patterns. Suppose that some algorithm A packs the elements of the concatenated list $(L_1L_2\dots L_k)$ into bins.

Definition 2.2. *A (feasible) packing pattern $p = (p_1, p_2, \dots, p_k)$ is a k-dimensional vector, and p_j denotes the number of items from the list L_j, $j = 1, 2, \dots, k$, while the algorithm places items into a bin according to that packing pattern. It is clear that $\sum_{i=1}^k a_i p_i \leq 1$, where a_i is the size of items in L_i.*

The set of all feasible packing patterns will be denoted by P. We define the subsets

$$P_i = \{p \in P \mid p_i > 0 \text{ and } p_j = 0, \text{ for } j < i\},\ i = 1, \dots k. \tag{3}$$

Clearly, $P_i \cap P_j = \emptyset$ if $i \neq j$, and $P = \cup_{i=1}^k P_i$.

While we pack the elements of the concatenated list $L = (L_1L_2\dots L_k)$, every bin must be filled according to one feasible packing pattern. Let us denote the total number of bins with the packing pattern p by $n(p)$. The number of bins used by algorithm A while successively packing the lists is

$$A(L_1\dots L_j) = \sum_{i=1}^{j} \sum_{p\in P_i} n(p),\ \text{for } j = 1, \dots, k, \tag{4}$$

and

$$n_j = \sum_{p\in P} p_j n(p),\ \text{for } j = 1, 2, \dots, k. \tag{5}$$

Van Vliet stated the following theorem.

Theorem 2.1 (van Vliet, [101])**.** *Let w_j, $1 \leq j \leq k$, be some positive weights such that for every $p \in P_i, i = 1, 2, \dots, k$*

$$\sum_{j=i}^{k} w_j p_j \leq k - i + 1 \tag{6}$$

holds. Then for every online algorithm A we have that

$$R_A^\infty \geq \frac{\sum_{j=1}^k w_j c_j}{\sum_{j=1}^k U_j}, \tag{7}$$

In this theorem van Vliet considered k positive weights without any further condition, so if we apply this theorem for a special class of algorithms the weights can be arbitrary small. To avoid this inconvenience we can rescale the weights, and so we reformulate the above theorem as follows.

Theorem 2.2. (Balogh, Békési and Galambos, [7]). *Let α_j and β_j be $2k$ positive integers such that for every $p \in P_i, i = 1, 2, \ldots, k$,*

$$\sum_{j=i}^{k} \beta_j p_j \leq \sum_{j=i}^{k} \alpha_j. \tag{8}$$

Then for every online algorithm A

$$R_A^{\infty} \geq \max_{1 \leq j \leq k} \limsup_{n \to \infty} \frac{A(L_1 L_2 \ldots L_j)}{\mathrm{OPT}(L_1 L_2 \ldots L_j)} \geq \frac{\sum_{j=1}^{k} \beta_j c_j}{\sum_{j=1}^{k} \alpha_j U_j}. \tag{9}$$

Proof. If we multiply, for $j = 1, 2, \ldots, k$, the equations (4) and (5) by α_j and β_j, respectively, and sum all weighted equations, we get

$$\sum_{j=1}^{k} \alpha_j A(L_1 \ldots L_j) = \sum_{j=1}^{k} \alpha_j \sum_{i=1}^{j} \sum_{p \in P_i} n(p) \tag{10}$$

and

$$\sum_{j=1}^{k} \beta_j n_j = \sum_{j=1}^{k} \beta_j \sum_{p \in P} p_j n(p). \tag{11}$$

Because of the property of the constants it follows that

$$\begin{aligned}
\sum_{j=1}^{k} \alpha_j \sum_{i=1}^{j} \sum_{p \in P_i} n(p) &= \sum_{p \in P_1} (\alpha_1 + \alpha_2 + \ldots + \alpha_k)\, n(p) \\
&+ \sum_{p \in P_2} (\alpha_2 + \ldots + \alpha_k)\, n(p) + \ldots + \sum_{p \in P_k} \alpha_k n(p) \\
&\geq \sum_{p \in P_1} (\beta_1 p_1 + \beta_2 p_2 + \ldots + \beta_k p_k)\, n(p) \\
&+ \sum_{p \in P_2} (\beta_2 p_2 + \ldots + \beta_k p_k)\, n(p) + \ldots + \sum_{p \in P_k} \beta_k p_k n(p) \\
&= \sum_{j=1}^{k} \beta_j \sum_{p \in P} p_j n(p).
\end{aligned}$$

So – using (10) and (11) – we get that

$$\sum_{j=1}^{k} \alpha_j A(L_1 \dots L_j) \geq \sum_{j=1}^{k} \beta_j n_j. \tag{12}$$

Therefore

$$\begin{aligned} R_A^\infty &\geq \max_{1\leq j\leq k} \limsup_{n\to\infty} \frac{\alpha_j A(L_1 L_2 \dots L_j)}{\alpha_j \mathrm{OPT}(L_1 L_2 \dots L_j)} \geq \limsup_{n\to\infty} \frac{\sum_{j=1}^{k} \alpha_j A(L_1 \dots L_j)}{\sum_{j=1}^{k} \alpha_j OPT(L_1 \dots L_j)} \\ &\geq \limsup_{n\to\infty} \frac{\sum_{j=1}^{k} \beta_j n_j}{\sum_{j=1}^{k} \alpha_j OPT(L_1 \dots L_j)} \geq \limsup_{n\to\infty} \frac{n \sum_{j=1}^{k} \beta_j c_j}{n \sum_{j=1}^{k} \alpha_j U_j} = \frac{\sum_{j=1}^{k} \beta_j c_j}{\sum_{j=1}^{k} \alpha_j U_j}. \end{aligned}$$

□

2.1.2 Right choice of the weights

In [48] Galambos and Frenk did not give an explicit discussion of the packing pattern technique, but – using the idea of the packing pattern – they were able to give a simpler proof for the 1.5363... lower bound for online bin packing algorithms given by Liang [79]. They investigated the parametric case as well. In [101] Van Vliet – using his generalization – improved the lower bound to 1.54014.... Here we will show that the right choice of the weights allows us to give the same lower bound using the packing pattern technique as van Vliet got with the help of the linear programming technique. During his proof he constructed a linear program, he solved it and defined two functions f_k and g_k, both of them depending on k. He received his result as a limit of a function in f_k and g_k for $k \to \infty$. Since van Vliet proved that with the help of the applied sequences there is no possibility to get a better lower bound, our procedure will also justify that our approach has the same power as the LP method has.

In our construction we will use the generalized Sylvester sequences, and m_j^r will be denoted by m_j, $j = 1, 2, \dots, k$. Now we define k lists as follows. Let $n = c(m_k - 1)$ for some positive integer c. Each list L_j, $j = 1, \dots, k-1$, contains n elements, while L_k contains rn pieces of elements, i.e. $c_j = 1$, if $j = 1, 2, \dots, k-1$, and $c_k = r$. The sizes of elements in L_j are $a_j = 1/m_{k-j+1} + \varepsilon$, where $0 < \varepsilon < 1/(r+k)(m_k(m_k-1))$. The following Lemma was proven in [79].

Lemma 2.1. (Liang, [79]).

(i) $\mathrm{OPT}(L_1 L_2 \dots L_j) = \frac{n}{m_{k-j+1}-1}$, *for all* $j = 1, \dots, k-1$.

(ii) $\mathrm{OPT}(L_1 L_2 \dots L_k) = n$.

So for a fixed k we set

$$U_j = \begin{cases} \frac{1}{m_{k-j+1}-1} & \text{, if } 1 \leq j \leq k-1, \\ 1 & \text{, if } j = k, \end{cases}$$

and we define the following constants.

$$\beta_j = \begin{cases} 1 & \text{, if} \quad j = 1 \\ (m_{k-j+1} - 1)\beta_{j-1} & \text{, if} \quad 2 \le j \le k-1, \\ \beta_{k-1} & \text{, if} \quad j = k. \end{cases}$$

$$\alpha_j = \begin{cases} \beta_{j+1} & \text{, if} \quad 1 \le j \le k-1, \\ r\beta_k & \text{, if} \quad j = k. \end{cases}$$

Comparing our weights to the ones given in [101] we can realize the difference between them. So, although the formula is almost the same, our result is better than the one that van Vliet has got with the help of the packing pattern technique. On the other hand, it is also easy to check that our proof is much simpler than the LP technique.

Theorem 2.3. (Balogh, Békési and Galambos, [7]). *Every one-dimensional online bin packing algorithm A has worst case ratio*

$$R_A^\infty \ge \lim_{k\to\infty} \frac{\sum_{j=1}^k c_j\beta_j}{\sum_{j=1}^k \alpha_j U_j}. \tag{13}$$

Proof. For the application of Theorem 2.2 we have to show that for every $i = 1, \dots, k$,

$$\sum_{j=i}^{k} \beta_j p_j \le \sum_{j=i}^{k} \alpha_j. \tag{14}$$

First we investigate the left hand side of (14).

Lemma 2.2.

$$\sum_{j=i}^{k} \beta_j p_j \le \beta_i (m_{k-i+1} - 1). \tag{15}$$

Proof. Let $p = (0, \dots, 0, p_i, p_{i+1}, \dots, p_k) \in P_i$ be a feasible packing pattern. It has been proven several times (see e.g. [48], [50]) that if we replace each element of L_j by $m_{k-j+1}-1$ elements of L_{j-1} for some $j = k, \dots, i+1$ then the sum of the sizes of the elements in the bin does not increase, and so the new pattern – denoted by p' – remains feasible.

If we denote the left hand side of (15) by $W(p)$ and $W(p')$, respectively, then – using the definitions of the β-s – we get

$$W(p) = \sum_{l=i}^{k} \beta_l p_l = \sum_{l=i}^{j-1} \beta_l p_l + \underbrace{(m_{k-j+1} - 1)\beta_{j-1}}_{\beta_j} p_j + \sum_{l=j+1}^{k} \beta_l p_l = W(p'),$$

i.e. the left hand side of (15) does not change while doing this substitution. Repeating this replacement iteratively on p for $j = k, \dots, i+1$ we will end up with a packing pattern

containing only elements from L_i. Clearly for this pattern $p_i \leq m_{k-i+1} - 1$ holds. From this we get that for every $p \in P_i$ pattern

$$\sum_{j=i}^{k} \beta_j p_j \leq \beta_i (m_{k-i+1} - 1). \tag{16}$$

which completes the proof of the lemma.

□

Now we concentrate on the right side of (14).

Lemma 2.3.

$$\beta_i (m_{k-i+1} - 1) = \sum_{j=i}^{k} \alpha_j. \tag{17}$$

Proof.

$$\begin{aligned}
\textstyle\sum_{j=i}^{k} \alpha_j &= (\alpha_i + \ldots + \alpha_{k-1}) + \alpha_k = (\alpha_i + \ldots + \alpha_{k-1}) + r\beta_k \\
&= (\alpha_i + \ldots + \alpha_{k-1}) + (m_1 - 1)\beta_k \\
&= (\alpha_i + \ldots + \alpha_{k-2}) + \alpha_{k-1} + (m_1 - 1)\beta_{k-1} \\
&= (\alpha_i + \ldots + \alpha_{k-2}) + m_1\beta_{k-1} \\
&= (\alpha_i + \ldots + \alpha_{k-3}) + \alpha_{k-2} + (m_2 - 1)\beta_{k-1} \\
&= (\alpha_i + \ldots + \alpha_{k-3}) + \beta_{k-1} + (m_2 - 1)\beta_{k-1} \\
&= (\alpha_i + \ldots + \alpha_{k-3}) + m_2\beta_{k-1} = (\alpha_i + \ldots + \alpha_{k-3}) + m_2(m_2 - 1)\beta_{k-2} \\
&= (\alpha_i + \ldots + \alpha_{k-3}) + (m_3 - 1)\beta_{k-2} = \ldots \\
&= \alpha_i + (m_{k-i} - 1)\beta_{i+1} = m_{k-i}\beta_{i+1} = m_{k-i}(m_{k-i} - 1)\beta_i \\
&= (m_{k-i+1} - 1)\beta_i.
\end{aligned}$$

□

Combining the results of Lemma 2.2 and Lemma 2.3 the inequality (14) follows immediately.

□

As an example we show the case $r = 1$, $k = 3$, where $m_1 = 2$, $m_2 = 3$, $m_3 = 7$, $\beta_1 = 1$, $\beta_2 = 2$, $\beta_3 = 2$, $\alpha_1 = 2$, $\alpha_2 = 2$, $\alpha_3 = 2$, $U_1 = \frac{1}{6}$, $U_2 = \frac{1}{2}$, $U_3 = 1$. So we get

$$R_A^{\infty} \geq \frac{\sum_{j=1}^{3} c_j \beta_j}{\sum_{j=1}^{3} \alpha_j U_j} = \frac{5}{\frac{1}{3} + 1 + 2} = \frac{3}{2}$$

as it was given by Yao in [103]. Table 2.2. displays the van Vliet's lower bounds for the asymptotic performance ratio of online algorithms for some values of k and r, which where calculated by our formula.

	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$
$k=3$	1,5000000	1,3793103	1,2878787	1,2283464	1,1880733
$k=4$	1,5390070	1,3895759	1,2914337	1,2298587	1,1888167
$k=5$	1,5401467	1,3896489	1,2914427	1,2298604	1,1888172
$k=6$	1,5401474	1,3896489	1,2914427	1,2298604	1,1888172
$k=7$	1,5401474	1,3896489	1,2914427	1,2298604	1,1888172
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$k=\infty$	1,5401474	1,3896489	1,2914427	1,2298604	1,1888172

Table 2.2. van Vliet's lower bounds for online bin packing algorithms.

2.1.3 New parametric online lower bound

Proving his result van Vliet used the Sylvester sequences. This is a so-called double exponential sequence whose reciprocals tend very quickly to zero (see also [1], [59], [92]). During the last two decades a lot of efforts have been made to improve this result. It was already proven by van Vliet that his result was not improvable with the Sylvester sequence. Therefore we inquired to find other sequences which do not tend so quickly. Besides other approaches we attempted to give up the greedy choice of the next elements in the sequence. Among other – unsuccessful – shots we hit the following sequence (see [7]).

Definition 2.3. *For any integer $r \geq 1$ let*

$$\begin{aligned} b_{1,r} &= r+1, & b_{2,r} &= r+2, & b_{3,r} &= b_{1,r}b_{2,r}+1, \\ b_{j,r} &= b_{3,r}^{j-2}, \quad 4 \leq j \leq k-1, & b_{k,r} &= b_{1,r}b_{2,r}b_{3,r}^{k-3}+1. \end{aligned}$$

For the simpler notation instead of $b_{j,r}$ we will use b_j.

It is easy to prove that for any fixed integer $k < \infty$

$$r\frac{1}{b_1} + \sum_{i=2}^{k} \frac{1}{b_i} < 1.$$

If we compare the contents of Table 2.2 and Table 2.3 it is conspicuous: we loose – in contrast to the greedy sequence – a bit at the fourth member, but later our patience leads to improvement.

b_j	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$
$j=1$	2	3	4	5	6
$j=2$	3	4	5	6	7
$j=3$	7	13	21	31	43
$j=4$	49	169	441	961	1849
$j=5$	343	2197	9261	29791	79507

Table 2.3. The first few parametric values of the new sequence for $k=5$.

Using this new sequence we construct our lists as follows. Let A be an online algorithm. In the first step we consider a concatenated list with sublists $L_1, L_2, \ldots, L_k$ for $k \geq 4$ as follows.

(i) L_k contains nr elements of size $a_k = \frac{1}{b_1} + \varepsilon$,

(ii) L_{k-1} contains n elements of size $a_{k-1} = \frac{1}{b_2} + \varepsilon$,

(iii) L_j contains n elements of size $a_j = \frac{1}{b_{k-j+1}} + \varepsilon$, where $2 \leq j \leq k-2$

(iv) L_1 contains n elements of size $a_1 = \frac{1}{b_k} + \varepsilon$,

where $\varepsilon \leq \frac{1}{(k+r)b_k(b_k-1)}$, and $n = c(b_k-1)$, for some integer $c \geq 1$. So, the constants we apply while we use the Theorem 2.2 are $c_j = 1$, if $j \leq k-1$, and $c_k = r$. Note that for fixed $k \leq 4$ this definition gives the same lists, which are used in the proof of the van Vliet's lower bound.

If one tries to prove that this sequence of the lists results in a better lower bound, of course the LP method established by van Vliet in [100] is adaptable. Indeed, we also constructed this LP. But – as we mentioned above - the proof of the cited paper seemed to be rather complicated, so we tried to apply the packing pattern technique. To do that, the only question was whether we could find the correct values of α-s and β-s. Before proving our main theorem we prove some lemmas.

Lemma 2.4. *For the optimum values of the concatenated lists the following relations hold*

(i) $\mathrm{OPT}(L_1 \ldots L_j) \leq \frac{n}{b_1 b_2 b_3^{k-j-2}}$, *for* $1 \leq j \leq k-2$,

(ii) $\mathrm{OPT}(L_1 \ldots L_{k-1}) \leq \frac{n}{b_1} = \frac{n}{r+1}$,

(iii) $\mathrm{OPT}(L_1 \ldots L_k) \leq n$,

Proof. We will generate a feasible packing for each concatenated list.

Case (i). It is trivial that the items of L_1 can be packed into $\frac{n}{b_1b_2b_3^{k-3}}$ bins. Consider the list $(L_1L_2\dots L_j)$ for $2 \le j \le k-2$. Let $z = b_1b_2b_3^{k-3}$, then

$$\begin{aligned} S_j = \sum_{i=1}^{j} a_i &= \tfrac{1}{b_1b_2b_3^{k-3}+1} + \tfrac{1}{b_3^{k-3}} + \ldots + \tfrac{1}{b_3^{k-j-1}} + \tfrac{j}{(k+r)b_1b_2b_3^{k-3}(b_1b_2b_3^{k-3}+1)} \\ &< \tfrac{z+b_1b_2\cdot(z+1)+b_1b_2b_3(z+1)+\ldots+b_1b_2b_3^{j-2}(z+1)+1}{z(z+1)} \\ &= \tfrac{1+b_1b_2(1+b_3+\ldots+b_3^{j-2})}{z} = \tfrac{b_3^{j-1}}{b_1b_2b_3^{k-3}} = \tfrac{1}{b_1b_2b_3^{k-j-2}}. \end{aligned}$$

This proves that the elements of the lists $(L_1 \dots L_j)$ can be packed into $\frac{n}{b_1b_2b_3^{k-j-2}}$ bins if $1 \le j \le k-2$.

Case (ii).

$$\begin{aligned} S_{k-1} = \sum_{i=1}^{k-1} a_i &= \tfrac{1}{b_1b_2b_3^{k-3}+1} + \tfrac{1}{b_3^{k-3}} + \ldots + \tfrac{1}{b_3} + \tfrac{1}{b_2} + \tfrac{k-1}{(k+r)b_1b_2b_3^{k-3}(b_1b_2b_3^{k-3}+1)} \\ &< \tfrac{z+b_1b_2(z+1)+\ldots+b_1b_2b_3^{k-4}(z+1)+b_1b_3^{k-3}(z+1)+1}{z(z+1)} \\ &= \tfrac{b_1b_2(1+b_3+\ldots+b_3^{k-4})+b_1b_3k-3}{z} = \tfrac{b_1b_2\frac{b_{k-3}-1}{b_3-1}+b_1b_3k-3}{z} \\ &= \tfrac{1+b_3^{k-3}-1+b_1b_3k-3}{z} == \tfrac{b_3^{k-3}(b_1+1)}{b_1b_2b_3^{k-3}} = \tfrac{b_1+1}{b_1b_2} = \tfrac{1}{b_1}. \end{aligned}$$

So, the elements of $(L_1 \dots L_{k-1})$ can be packed into $\frac{n}{b_1}$ bins.

Case (iii).

$$\begin{aligned} S_k &= \sum_{i=1}^{k-1} a_i + ra_k \\ &= \tfrac{1}{b_1b_2b_3^{k-3}+1} + \tfrac{1}{b_3^{k-3}} + \ldots + \tfrac{1}{b_3} + \tfrac{1}{b_2} + \tfrac{r}{b_1} + \tfrac{k+r-1}{(k+r)b_1b_2b_3^{k-3}(b_1b_2b_3^{k-3}+1)} \\ &< \tfrac{(z+1)(1+b_1b_2+b_1b_2b_3+\ldots+b_1b_2b_3^{k-4}+b_1b_3^{k-3}+rb_2b_3^{k-3})}{z(z+1)} \\ &= \tfrac{1+b_1b_2\left(1+b_3+\ldots+b_3^{k-4}\right)+b_1b_3^{k-3}+rb_2b_3^{k-3}}{z} \\ &= \tfrac{1+b_1b_2\frac{b_3^{k-3}-1}{b_3-1}+b_1b_3^{k-3}+rb_2b_3^{k-3}}{z} = \tfrac{1+b_3^{k-3}-1+b_1b_3^{k-3}+rb_2b_3^{k-3}}{z} \\ &= \tfrac{b_3^{k-3}(1+b_1+rb_2)}{b_1b_2b_3^{k-3}} = \tfrac{b_2(1+r)}{b_1b_2} = 1. \end{aligned}$$

Therefore, the elements of $(L_1 \dots L_k)$ can be packed into n bins.

□

Based on the above Lemma, we can choose the values of U_j^k as follows.

$$U_j^k = \begin{cases} \frac{1}{b_1 b_2 b_3^{k-j-2}} & \text{, if } j \leq k-2, \\ \frac{1}{b_1} = \frac{1}{r+1} & \text{, if } j = k-1, \\ 1 & \text{, if } j = k \end{cases}$$

For a given $k \geq 4$ we define two k-dimensional vectors β^k and α^k, as follows.

$$\beta_j^k = \begin{cases} 1 & \text{, if } \quad j = 1, \\ b_1 b_2 & \text{, if } \quad j = 2, \\ b_3 \beta_{j-1}^k & \text{, if } \quad 3 \leq j \leq k-2, \\ b_1 \beta_{k-2}^k & \text{, if } \quad k-1 \leq j \leq k. \end{cases}$$

$$\alpha_j^k = \begin{cases} b_1 b_2 & \text{, if } \quad j = 1, \\ (b_1 b_2)^2 & \text{, if } \quad j = 2 \text{ and } k \geq 5, \\ b_3 \alpha_{j-1}^k & \text{, if } \quad 3 \leq j \leq k-3, \\ \beta_{k-1}^k & \text{, if } \quad k-2 \leq j \leq k-1, \\ r\beta_k^k & \text{, if } \quad j = k. \end{cases}$$

Considering the above constants we need to prove that inequality (8) holds for every feasible packing. Let us consider the packing pattern $p = (0, ..., 0, p_i, ..., p_k)$, where $p_i > 0$. It is clear that p belongs to the subset P_i of the feasible packing.

Definition 2.4. *The packing pattern p is dominant in P_i if*

$$a_t + \sum_{j=1}^{k} a_j p_j > 1,$$

for any t, $1 \leq t \leq k$. We note that in our recent case $a_s < a_t$, if $s < t$, so during our proof we will use that p is dominant *in P_i if*

$$a_i + \sum_{j=1}^{k} a_j p_j > 1.$$

Let $D_i(p)$ be the set of those packing patterns for which p is dominant in P_i. So, it is enough to investigate the dominant patterns for each P_i. See for example [95].

Lemma 2.5. *Let $L = (L_1 \dots L_k)$ be the above defined concatenated list for some $k \geq 4$. Then for every feasible dominant packing pattern $p \in P_i$*

$$\sum_{j=i}^{k} \beta_j^k p_j \leq \sum_{j=i}^{k} \alpha_j^k. \tag{18}$$

Proof. We prove by induction. Since the constants for the case $k = 4$ are the same as in the new proof of van Vliet's lower bound, for this case the statement of the Lemma holds. Suppose now that the statement holds for some $k \geq 4$. We will distinguish two cases.

First, we suppose that $p \in P_i$, $i \geq 3$. Let $p' \in P_j$ for some $i \leq j \leq k+1$. We say that the packing pattern p' is the *j-suffix* of p if

$$p'_l = \begin{cases} p_l & \text{, if} \quad j \leq l \leq k+1, \\ 0 & \text{, if} \quad l < j. \end{cases}$$

Claim 2.1. *If $p \in P_i$, $i \geq 3$, and $p' \in P_j$ is its j-suffix, where $i \leq j$, then the packing pattern $p'' = (0, \ldots, 0, p_j, p_{j+1}, \ldots, p_{k+1})$ was already investigated during the case $k' = k - j + 2$ and it satisfies the condition (18).*

Proof. Since $k' < k$, so we can apply the induction hypothesis to the packing pattern p'', and the statement follows immediately from the definitions of the lists.

□

We can suppose that $p \in P_i$, $i \leq 2$. Let us transform $p = (p_1, ..., p_{k+1})$ to a new packing pattern p^T as follows

$$p_j^T = \begin{cases} p_1 + b_1 b_2 p_2 & \text{,if} \quad j = 1, \\ 0 & \text{,if} \quad j = 2, \\ p_i & \text{,if} \quad j > 2. \end{cases}$$

By the definition of β_j^{k+1} the equation

$$\sum_{j=1}^{k+1} \beta_j^{k+1} p_j^T = \sum_{j=1}^{k+1} \beta_j^{k+1} p_j$$

holds. Clearly, if p is dominant with respect to the set $D_i(p)$ then p^T is also feasible and dominant with respect to those packing patterns which we get with the same transformation from the elements of $D_i(p)$.

Claim 2.2. *For every dominant pattern p of the form $(p_1, 0, p_3, ..., p_{k+1})$, p_1 can be divided by b_3.*

Proof. Consider an arbitrary index j, $3 \leq j \leq k+1$. The packing pattern p contains exactly p_j items from L_j. By the definition of the items $q_j = \frac{b_{k+1}-1}{b_{k-j+2}}$ is an integer and can be divided by b_3. So we can substitute each element of L_j by q_j elements of L_1. We must prove that the pattern remains feasible after the substitutions. For this we show that $p_1 + \sum_{j=3}^{k+1} p_j q_j \leq b_{k+1} - 1$.

The pattern is feasible before the substitutions, so

$$p_1 a_1 + \sum_{j=3}^{k+1} p_j a_j \leq 1.$$

Since there is at least one positive p_j $(j = 1, 3, \ldots, k+1)$ and the previous sum contains at least one ε it follows that

$$p_1 \frac{1}{b_{k+1}} + \sum_{j=3}^{k+1} p_j \frac{1}{b_{k-j+2}} < 1,$$

so

$$1 \;>\; p_1 \frac{1}{b_{k+1}} + \sum_{j=3}^{k+1} p_j \frac{b_{k+1}-1}{(b_{k+1}-1)b_{k-j+2}} \geq p_1 \frac{1}{b_{k+1}} + \sum_{j=3}^{k+1} p_j q_j \frac{1}{b_{k+1}}, \tag{19}$$

i.e.

$$p_1 + \sum_{j=3}^{k+1} p_j q_j < b_{k+1}, \tag{20}$$

and since p_1, p_j-s and q_j-s $(j = 3, ..., k+1)$ are integers,

$$p_1 + \sum_{j=3}^{k+1} p_j q_j \leq b_{k+1} - 1. \tag{21}$$

Considering (21) and the definition of ε, we get that

$$\big(p_1 + \sum_{j=3}^{k+1} p_j q_j\big) a_1 \leq \big(b_{k+1} - 1\big)\big(\frac{1}{b_{k+1}} + \varepsilon\big) \leq \frac{b_{k+1}-1}{b_{k+1}} + \frac{1}{b_{k+1}} = 1,$$

which means that the pattern is feasible.

Having done this substitution for every j, we can calculate the maximal value of p_1 as

$$p_1 = b_1 b_2 b_3^{k-2} - \sum_{j=3}^{k+1} p_j q_j, \tag{22}$$

i.e. p_1 is the difference between the maximal possible number of a_1 items in a bin, and the sum of the weighted q_j-s. Since each q_j can be divided by b_3, so p_1 can also be divided by b_3.

□

This fact proves, that if p^T is a dominant pattern, then $p_1 + b_1 b_2 p_2$ can be divided by b_3 in p^T. Now we are ready to define a new packing pattern of $(L_1 L_2 \ldots L_k)$. This will be

$$p^k = \big(\frac{p_1 + b_1 b_2 p_2}{b_3}, p_3, ..., p_{k+1}\big).$$

Claim 2.3. *If $p = (p_1, p_2, \ldots, p_{k+1})$ is a feasible dominant packing pattern for the list $L = (L_1 L_2 \ldots L_{k+1})$ then $p^k = (\frac{p_1 + b_1 b_2 p_2}{b_3}, p_3, ..., p_{k+1})$ is also feasible for $L' = (L_1 L_3 \ldots L_{k+1})$.*

Proof. Let we remind the reader that the sizes of the elements are different if one consider L or L'. We need to prove that by substituting the p_1 and p_2 pieces of elements from L with $\frac{p_1+b_1b_2p_2}{b_3}$ pieces of items from L' the occupied place will not increase.

$$\begin{aligned}\tfrac{p_1+b_1b_2p_2}{b_3}\left(\tfrac{1}{b_1b_2b_3^{k-3}+1}+\varepsilon\right) &< \tfrac{p_1}{b_1b_2b_3^{k-2}+b_3}+\tfrac{p_1}{b_3^{k-2}}\varepsilon+\tfrac{p_2}{b_3^{k-2}}+\tfrac{b_1b_2p_2}{b_3^{k-2}}\varepsilon\\ &< p_1\left(\tfrac{1}{b_1b_2b_3^{k-2}+1}+\tfrac{1}{b_3^{k-2}}\varepsilon\right)+p_2\left(\tfrac{1}{b_3^{k-2}}+\tfrac{b_1b_2b_3^{k-2}}{b_3^{k-2}}\varepsilon\right)\\ &< p_1\left(\tfrac{1}{b_1b_2b_3^{k-2}+1}+\varepsilon\right)+p_2\left(\tfrac{1}{b_3^{k-2}}+\varepsilon\right).\end{aligned}$$

□

By the induction hypothesis for each feasible packing

$$\sum_{j=1}^{k}\beta_j^k p_j^k \le \sum_{j=1}^{k}\alpha_j^k. \tag{23}$$

Using the definition of α-s and β-s we can calculate both sides of (23) for a given packing pattern:

$$\sum_{j=1}^{k}\beta_j^k p_j^k = \frac{p_1+b_1b_2p_2}{b_3}+\sum_{j=2}^{k}\beta_j^k p_{j+1} = \frac{p_1+b_1b_2p_2}{b_3}+\sum_{j=3}^{k+1}\frac{\beta_j^{k+1}}{b_3}p_j = \frac{1}{b_3}\sum_{j=1}^{k+1}\beta_j^{k+1}p_j,$$

and

$$\sum_{j=1}^{k}\alpha_j^k = b_1b_2+\sum_{j=2}^{k}\alpha_j^k = b_1b_2+\sum_{j=3}^{k+1}\frac{\alpha_j^{k+1}}{b_3} = b_1b_2+\frac{1}{b_3}\sum_{j=3}^{k+1}\alpha_j^{k+1}.$$

So

$$\frac{1}{b_3}\sum_{j=1}^{k+1}\beta_j^{k+1}p_j \le b_1b_2+\frac{1}{b_3}\sum_{j=3}^{k+1}\alpha_j^{k+1},$$

therefore

$$\sum_{j=1}^{k+1}\beta_j^{k+1}p_j \le b_1b_2+(b_1b_2)^2+\sum_{j=3}^{k+1}\alpha_j^{k+1} = \sum_{j=1}^{k+1}\alpha_j^{k+1}. \tag{24}$$

So we completed the proof of the Claim 2.3.

□

Now we are ready to prove the new lower bound.

Theorem 2.4. (Balogh, Békési and Galambos, [7]). *Let r be a positive integer, and we consider the r-parametric bin packing problem. Then there is no one-dimensional online bin packing algorithm A with an asymptotic performance ratio*

$$R_A^\infty < \frac{r^6+8r^5+29r^4+60r^3+75r^2+55r+20}{r^6+7r^5+22r^4+40r^3+45r^2+33r+13}. \tag{25}$$

Proof. Using Lemmas (2.4) and (2.5) and Theorem (2.2) the following inequality is true

$$R_A^\infty \geq \frac{\sum_{j=1}^{k} c_j\beta_j}{\sum_{j=1}^{k} \alpha_j U_j}$$

Because

$$\begin{aligned}\sum_{j=1}^{k} c_j\beta_j &= 1+b_1b_2+(b_3-1)\sum_{j=3}^{k-2} b_3^{j-2}+b_3^{k-4}b_1^2b_2(1+r)\\ &= b_3+(b_3-1)b_3\tfrac{b_3^{k-4}-1}{b_3-1}+b_3^{k-4}b_1^3b_2 = b_3^{k-4}(b_3+b_1^3b_2)\end{aligned}$$

and

$$\begin{aligned}\sum_{j=1}^{k} \alpha_j U_j &= \tfrac{1}{b_3^{k-3}}+\tfrac{(b_1b_2)^2}{b_1b_2b_3^{k-4}}+(b_1b_2)^2\textstyle\sum_{j=3}^{k-3}\tfrac{b_3^{j-2}}{b_1b_2b_3^{k-j-2}}\\ &+\tfrac{1}{b_1b_2}b_1^2b_2b_3^{k-4}+\tfrac{1}{b_1}b_1^2b_2b_3^{k-4}+rb_1^2b_2b_3^{k-4}\\ &= \tfrac{1}{b_3^{k-3}}+\tfrac{b_1b_2}{b_3^{k-4}}+\tfrac{b_1b_2}{b_3^{k}}\textstyle\sum_{j=3}^{k-3} b_3^{2j}+b_3^{k-4}b_1(1+b_2+rb_1b_2)\\ &= \tfrac{1}{b_3^{k-3}}+\tfrac{b_1b_2}{b_3^{k-4}}+\tfrac{b_1b_2b_3^6}{b_3^{k}}\tfrac{b_3^{2(k-5)}-1}{b_3^2-1}+b_3^{k-4}b_1(1+b_2+rb_1b_2)\\ &= \tfrac{\frac{b_3+1}{b_3}+b_1b_2(b_3+1)+b_3^{2(k-4)}-b_3^2}{b_3^{k-4}(b_3+1)}+\tfrac{b_3^{2(k-4)}b_1(1+b_2+rb_1b_2)(b_3+1)}{b_3^{k-4}(b_3+1)}\\ &= \tfrac{\frac{1}{b_3}+b_3^{2(k-4)}\Big(1+b_1(1+b_2+rb_1b_2)(b_3+1)\Big)}{b_3^{k-4}(b_3+1)}.\\ &= \tfrac{\frac{1}{r^2+r+2}+(r^2+r+2)^{2(k-4)}(1+r(r^3+2)(r^2+r+2))}{(r^2+r+1)^{k-4}(r^2+r+2)}.\end{aligned}$$

So, we get that

$$R_A^\infty \geq \lim_{k\to\infty}\frac{b_3^{2(k-4)}(b_3+b_1^3b_2)(b_3+1)}{\frac{1}{b_3}+b_3^{2(k-4)}\Big(1+b_1(1+b_2+rb_1b_2)(b_3+1)\Big)} = \frac{(b_3+b_1^3b_2)(b_3+1)}{1+b_1(1+b_2+rb_1b_2)(b_3+1)}.$$

We know that $b_1=r+1$, $b_2=r+2$ and $b_3=r^2+3r+3$, so we get

$$R_A^\infty \geq \frac{r^6+8r^5+29r^4+60r^3+75r^2+55r+20}{r^6+7r^5+22r^4+40r^3+45r^2+33r+13}. \tag{26}$$

□

At the end of the section we present a table which displays the new lower bounds for the asymptotic competitve ratio of online algorithms for some values of r.

	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
$k = 3$	1,5000000	1,3793103	1,2878787	1,2283464	1,1880733
$k = 4$	1,5390070	1,3895759	1,2914337	1,2298587	1,1888167
$k = 5$	1,5403448	1,3896631	1,2914442	1,2298607	1,1888172
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$k = \infty$	1,5403726	1,3896636	1,2914443	1,2298607	1,1888172
$k = \infty$	$\frac{248}{161}$	$\frac{1694}{1219}$	$\frac{7502}{5809}$	$\frac{24992}{20321}$	$\frac{68420}{57553}$

Table 2.4. The new lower bounds for online bin packing algorithms

2.2 Concluding remarks

In this chapter we presented lower bounds for the asymptotic competitive ratios of online one-dimensional bin packing algorithms. Our improvements are very small in absolute values. However, we did an exhaustive search for possible lists and we have not found a worse. We strongly believe that the gap might be decreased only by defining better algorithms or one needs to find a new method for proving lower bounds.

The packing pattern technique was used for the two- and three-dimensional bin packing problems [50] and the best known lower bound for the online vector packing algorithms operates also with this technique (see [49]). Since in these cases the Sylvester sequences were used during the proof it is plausible that the application of the new series will also improve these lower bounds. We are convinced that this technique is usable for other classes of algorithms.

3 One-dimensional semi-online bin packing problems

In the previous section we revealed that the online algorithms do not have enough information to produce good packing. It is a question if by getting more – but not offline – sweep during the packing can we improve the efficiency of online algorithms? If for an online algorithm it is allowed to apply at least one of the following operations: repacking some items, look ahead into several next elements, or some kind of preordering, then we speak about *semi-online algorithms.* If the items arrive in preordered form then we have semi-online algorithm with *preordered lists.* In case of *k-bounded space* semi-online algorithms, for each item a_i, the choice of bins to pack them into is restricted to a set of maximal k *active* bins. Those semi-online algorithms which in each step allow only a restricted number of items to be repacked are called *k-repacking* semi-online algorithms.

3.1 Improved lower bound for decreasing lists

It was observed very early that the asymptotic competitive ratio of online algorithms becomes significantly better if one can suppose that the elements arrive in decreasing order. For this case the best known online algorithm is *Modified First Fit Decreasing* given in [70] with $R^{\infty}_{\mathrm{MFFD}} = \frac{71}{60}$. For preordered lists the best known lower bound was $\frac{8}{7}$, which was given by Csirik, Galambos and Turán [29]. So we had a very narrow gap $[1.142857..., 1.1833...]$ between the lower- and upper-bounds. In the proof of the lower bound of $\frac{8}{7}$ in [29] we used 2 lists which contain items with sizes $\frac{1}{3} + 2\varepsilon$ and $\frac{1}{3} - \varepsilon$, respectively. In the last 3 decades there was no success to give a better lower bound.

The difficulty originates from the fact that the sizes of the last list in the concatenated list may not be too small, since they may fill up the opened bins, resulting a better packing than in the earlier steps. So, there is no point in investigating concatenated lists with k different sublists with $k \to \infty$, while the sizes of elements getting progressively smaller and smaller. As a further application of the Theorem 2.2 in [7] we gave a construction with 3 different lists.

Let A be an online algorithm. We consider a concatenated list with three sublists L_1, L_2 and L_3.

- L_1 contains n_1 elements of size $\frac{7}{24} - 4\varepsilon$,
- L_2 contains n_2 elements of size $\frac{5}{24} + \varepsilon$,
- L_3 contains n_3 elements of size $\frac{4}{24} + \varepsilon$,

where $\varepsilon < \frac{1}{96}$, $n_1 = n_2 = 6n$ and $n_3 = 18n$. It means that $c_1 = c_2 = 6$ and $c_3 = 18$. It is easy to see that the following inequalities are true.

$$\mathrm{OPT}(L_1) \leq 2n, \quad \mathrm{OPT}(L_1L_2) \leq 3n, \quad \mathrm{OPT}(L_1L_2L_3) \leq 6n.$$

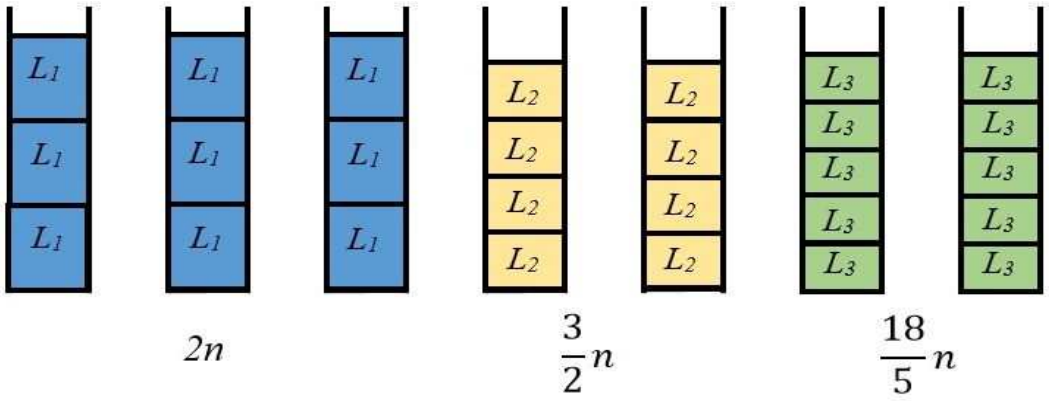

Figure 3.1. An example how to pack FFD the above, concatenated lists. $FFD(L_1) = 2n$, $FFD(L_1L_2) = \frac{7n}{2}$, $FFD(L_1L_2L_3) = \frac{71n}{10}$

So, we can set the upper bounds to

$$U_1 = 2, \qquad U_2 = 3, \qquad U_3 = 6.$$

In fact, these upper bounds are sharp. Let us now consider the following constants.

$$\alpha_1 = 4, \qquad \alpha_2 = 3, \qquad \alpha_3 = 5$$

and

$$\beta_1 = 4, \qquad \beta_2 = 2, \qquad \beta_3 = 1.$$

Considering the above constants we need to prove that inequality (8) holds for every dominant packing pattern. Since the number of dominant patterns is small we can investigate all of them. Three cases have to be distinguished.

(i) For $i = 1$, we consider the dominant patterns in P_1. We need to prove that any feasible packing pattern $p \in P_1$ satisfies

$$12 \geq 4p_1 + 2p_2 + p_3.$$

The dominant patterns are (3,0,0), (2,2,0), (2,1,1), (2,0,2), (1,3,0), (1,2,1), (1,1,3) and (1,0,4). It is easy to check that the inequality holds for all of them.

(ii) For $i = 2$, the dominant patterns of bins in P_2 have to be considered. These are (0,4,0), (0,3,2), (0,2,3) and (0,1,4). All of them satisfy

$$8 \geq 2p_2 + p_3.$$

(iii) Finally, we have to address the packing patterns in P_3. The only dominant pattern is (0,0,5) and the inequality

$$5 \geq p_3$$

trivially holds.

So, the conditions of Theorem 2.2 hold and therefore

$$R_A^\infty \geq \frac{\sum_{j=1}^{3} c_j\beta_j}{\sum_{j=1}^{k} \alpha_j U_j} = \frac{24+12+18}{8+9+30} = \frac{54}{47}.$$

We can summarize our calculation in the following theorem.

Theorem 3.1. (Balogh, Békési and Galambos, [7]). *There is no online algorithm for the one-dimensional bin packing problem which packs the elements in decreasing order and can have better asymptotic competitive ratio than* $\frac{54}{47} = 1.1489361\ldots$.

3.2 Bounded space semi-online algorithms with repacking

We will consider k-bounded space algorithms where repacking the items is allowed in each step. We note that the standard actions are also possible, i.e. we are allowed to (i) if the number of active bins is less or equal to $k-1$, open a new bin, (ii) close some active bins, (iii) pack a new item into one of the active bins.

Let $\mathrm{BIN}_1, \ldots, \mathrm{BIN}_k$ be the active bins, and we denote the contents of a bin with the bin. Then, contrary to the standard problem, we also allowed to

(iv) repack the set of items in the active bins, i.e. to form a new portion $\mathrm{BIN}'_1, \ldots, \mathrm{BIN}'_k$ of the items inside the active bins such that $\bigcup \mathrm{BIN}_i = \bigcup \mathrm{BIN}'_i$ holds, and the sum of the items in each bin of the new partition is at most one.

These restrictions (online and bounded-space) arise in many applications, as in packing trucks at a loading dock or in communicating via channels with bounded buffer size.

To allow action (iv) is a natural assumption. As long as an item is in some active bin, the item is available for the algorithm and it may change the position of this item. In the loading dock example mentioned above, trucks will partially be repacked and items will be moved from one truck at the loading dock to another in order to increase the number of pieces packed.

Chronologically, the first semi-online bounded-space algorithm with repacking was given by Galambos in [44]. This algorithm uses three active bins: BIN_1, BIN_2, and BIN_3. The algorithm is called *Buffered Next Fit Decreasing* (BNFD) and its steps can be shown in the next frame. (Indeed, BIN_2, and BIN_3 form a buffer with capacity 2.) The time complexity of this algorithm is $O(n)$ and its space complexity is $O(1)$. The algorithm is a 3-bounded space algorithm and $R^\infty_{\mathrm{BNFD}} \leq 18/10$. On the other hand, it is a 6-repacking algorithm in the sense that we are allowed to repack maximum 6 items in each step. (We will analyse such types of algorithms in the next subsection.)

(1) Get a new item x. If $x \leq 1/6$ then put x into BIN_1 by the Next-Fit rule (i.e we put the item into BIN_1 if it fits, otherwise we close the BIN_1, open a new empty bin, and put x into this newly opened active bin).
(2) If $x > 1/6$ then we put it into BIN_2 if it does fit, and go to Step (1).
(3) If $x > 1/6$ and it does not fit in BIN_2 then we put x into BIN_3. (BIN_3 may contain only one item.) Then order the items in bins BIN_2 and BIN_3 into decreasing order.
(4) BIN_2 will contain the larger items. Close BIN_2, open a new active BIN_2 and repack the contents of BIN_3 into BIN_2. Goto Step (1)

Table 3.1. The steps of the BNFD algorithm

In [35] J. Csirik and D. Johnson analysed a 2-bounded algorithm (without repacking) with an asymptotic competitive ratio of 17/10. Since we had a lower bound $h_\infty(1)$ (see [77], [51]), it was an open question that how many bins are needed to reach this value.

Based on our paper [51] we prove that the lower bound can be reached by 3 active bins.

3.2.1 Weighting function

In our proof we use the weighting function technique which was introduced by D. Johnson in [70]. The idea is the following. Let $W(x) : (0, 1] \to R^+$ be an appropriate weighting function. The weight of a set of items is defined to be the sum of weights over all items in it. Let A be an algorithm. Then, if for any list L there exists a finite constant C that the inequalities

(i) $W(L) \leq C \cdot \text{OPT}(L)$,

(ii) $A(L) \leq W(L) + K$

are true for any list L and $K < \infty$, then $A(L)$ is a C-competitive algorithm. We define our weighting function as follows.

$$W(x) = \begin{cases} x + \frac{1}{m_{i+1}-1}, & \text{for } \frac{1}{m_i} < x \leq \frac{1}{m_i-1}, \text{ and } 1 \leq i \\ \frac{m_i+1}{m_i} \cdot x, & \text{for } \frac{1}{m_{i+1}-1} < x \leq \frac{1}{m_i}, \text{ and } 1 \leq i. \end{cases}$$

where $m_i = m_i^1$, $i = 1, 2, \ldots$, denote the Sylvester sequence for $r = 1$.

Fact 3.1. *The proofs of the following observations are straightforward.*

(i) $W(x)$ *is nondecreasing in* $(0, 1]$.

(ii) For $i \geq 1$ *and* $x \leq 1/m_i$, $W(x)/x \leq (m_i + 1)/m_i$ *holds.*

(iii) For $i \geq 1$ and $x \geq (1/(m_{i+1}-1)$, $W(x)/x \geq (m_i+1)/m_i$ holds.

Similarly to the above definitions, we define the weight of a bin as the sum of the weights of all items in it, and the size of a set of items is the sum of sizes over all items in the set. For the defined weighting function the following lemma is true.

Lemma 3.1. *In any packing of a list L the weight of any bin is at most $h_\infty(1)$. Hence $W(L) \leq h_\infty(1)\mathrm{OPT}(L)$ holds.*

Proof. Let us consider some fixed bin $\mathcal{B}$ that contains items $q_1 \geq q_2 \geq \ldots \geq q_k$.

We assume that $r \leq k$ is the least i such that $q_r \leq 1/m_r$. Let us consider the following sum

$$Q_{rk} = \sum_{i=r}^{k} q_i < \frac{1}{m_r - 1}.$$

Then by Fact 3.1 (ii),

$$W\Big(\sum_{i=r}^{k} q_i\Big) \leq Q_{rk}\frac{m_r+1}{m_r}.$$

This yields

$$W(\mathcal{B}) \leq 1 - Q_{rk} + \sum_{i=1}^{r-1}\frac{1}{m_{i+1}-1} + \frac{m_r+1}{m_r}Q_{rk} < \sum_{i=1}^{r}\frac{1}{m_i-1} + \frac{1}{m_r(m_r-1)} < h_\infty(1).$$

□

3.2.2 Repacking algorithm REP_3

Now we define and analyse the Repacking Algorithm REP_3. This algorithm always keeps three active bins, denoted by BIN_1, BIN_2, and BIN_3, proceeds as follows.

(1) Get a new item x and put x into an empty active bin.
(2) Repack the three active bins so that at least one bin is empty or so that at least one bin has weight greater or equal to 1.
(3) Close all active bins with weight at least one and open new bins instead of them. Goto (1).

Table 3.3. The steps of the REP_3 algorithm

The running time of of REP_3 is $O(n^2)$, and its space complexity is $O(n)$.

The crucial step of the algorithm REP_3 is Step (2). The main part of this section is devoted to establishing the fact that Step (2) is always possible and how to perform it. To do this, we first prove the following theorem.

Type	Interval	$W(x)$	Type	Interval	$W(x)$
A	$(1/2, 1]$	$x + 1/2$	B_3	$(1/7, 1/6]$	$x + 1/42$
B_2	$(1/3, 1/2]$	$x + 1/6$	C_3	$(1/8, 1/7]$	$8x/7$
C_2	$(1/4, 1/3]$	$4x/3$	D_3	$(1/42, 1/8]$	$8x/7$
D_2	$(1/6, 1/4]$	$4x/3$	B_4	$(1/43, 1/42]$	$x + 1/1806$

Table 3.4. Illustration for the weighting function W.

Theorem 3.2. (Galambos and Woeginger, [51]). *Let* BIN_1, BIN_2, BIN_3 *be three bins. Then we can either repack the bins to produce a packing with an empty bin or we can find a subset of items with weight at least one and size at most one.*

Proof. We will assume that it is not possible to produce a *good packing* (a packing with empty empty bin) or to find a *good subset* (a subset of weight greater or equal one and size at most one), and we will derive a contradiction from this. It is convenient to classify the items according to the following partition of the unit-interval.

$$\begin{array}{lll} A & = (1/2, 1] & \\ B_i & = (1/m_i, 1/(m_i - 1)] & \text{for } i \geq 2. \\ C_i & = (1/(m_i + 1), 1/m_i] & \text{for } i \geq 2. \\ D_i & = (1/(m_{i+1} - 1), 1/(m_i + 1)] & \text{for } i \geq 2. \end{array}$$

An illustration for the weight of items with size close to one is given in Table 3.4. First, we will produce a *First-Fit Decreasing* (FFD) packing of the items in the three active bins. That means that we first order the items by decreasing size; then we go through the sorted list and place each successive piece into the first (leftmost) active bin into which it will fit. Because of our assumption, in the produced FFD-packing neither BIN_3 will be empty nor will there be a bin with weight greater or equal one. We will prove a number of combinatorial properties of the FFD-packing.

Claim 3.1. *In the FFD-packing, no bin contains an A-item.*

Proof. Trivial, as every A-item has weight at least one and would form a good subset. □

Claim 3.2. *In the FFD-packing, neither* BIN_2 *nor* BIN_3 *contains any* D_i*-item* x, $i \geq 2$.

Proof. Suppose the opposite. Since x was not put into BIN_1 therefore BIN_1 is at least $m_i/(m_i + 1)$ full. By Fact 3.1 (iii) for $x \geq 1/(m_{i+1} - 1)$, $W(x)/x \geq (m_i + 1)/m_i$ must hold. Therefore the total weight of BIN_1 is at least

$$\frac{m_i}{m_i + 1} \cdot \frac{m_i + 1}{m_i} = 1.$$

So the contents of BIN_1 would be a good subset.

□

Claim 3.3. *In the FFD-packing, neither* BIN_2 *nor* BIN_3 *contains any* B_i*-item* x, $i \neq 3$. *Moreover,* BIN_2 *and* BIN_3 *together contain at most one* B_3*-item.*

Proof. First we show that BIN_2 and BIN_3 cannnot contain a B_2-item. By Claim 3.1, BIN_1 does not contain A-item. If BIN_2 or BIN_3 contains a B_2-item, BIN_1 must contain two B_2-items, and so, the items in BIN_1 would form a good subset.

Next, we examine the case $i \geq 3$. We denote the overall size of all items in BIN_1 that are larger than the B_i items by X. By Fact 3.1 (iii), the total weight of these items is at least $(m_{i-1}+1)X/m_{i-1}$. Let the number of B_i-items in BIN_1 be b, and let their overall size be denoted by B. We use the fact that $W(\mathrm{BIN}_1) < 1$ must hold and get

$$\frac{m_{i-1}+1}{m_{i-1}}X + B + \frac{b}{m_{i+1}-1} < 1. \tag{27}$$

As the item $x \leq 1/(m_i - 1)$ did not fit into BIN_1, we know that

$$X + B > \frac{m_i - 2}{m_i - 1} \tag{28}$$

must hold. We subtract (28) multiplied by $(m_{i-1}+1)$ from (27) multiplied by m_{i-1} and derive

$$\frac{m_{i-1}}{m_{i+1}-1}b < m_{i-1} - \frac{m_i - 2}{m_i - 1}(m_{i-1}+1) + B. \tag{29}$$

On the other hand, we know that every B_i-item in BIN_1 has size at most $1/(m_i - 1)$ and that there are exactly b such items. This implies $B \leq b/(m_i - 1)$. Plugging this into (29) and simplifying the resulting inequality yields

$$b > \frac{m_i - m_{i-1} - 2}{m_i - m_{i-1}}m_i = m_i - \frac{2m_i}{m_i - m_{i-1}}. \tag{30}$$

Now for $i \geq 4$, the righthand side of (30) is at least $(m_i - 3)$ and therefore, b is at least $(m_i - 2)$. Together with the item x in BIN_2, there are at least $(m_i - 1)$ B_i-items, each of size greater than $1/m_i$. Hence, the total weight of these items is at least

$$(m_i - 1)W\Big(\frac{1}{m_i} + \varepsilon\Big) > (m_i - 1)\Big(\frac{1}{m_i} + \frac{1}{m_{i+1}-1}\Big) = 1,$$

and it is easy to check that they fit into a single bin. Thus, we constructed a good subset and obtained a contradiction.

For $i = 3$, the righthand side of (30) is $7/2$, and this yields $b \geq 4$. If there are two or more B_3 items in BIN_2 and BIN_3, we have at least six B_3-items and we can argue analogously to above.

□

Claim 3.4. *In the FFD-packing, neither* BIN_2 *nor* BIN_3 *contains any* C_2*-item* x. *The bin* BIN_3 *does not contain any* B_3*-item* y.

Proof. Assume the contrapositive. As x did not fit into BIN_1, BIN_1 must contain three C_2-items, each of size at least $1/4$. This gives for BIN_1 an overall weight of at least one, and the first part of the claim is proven.

To see the correctness of the second part, we observe that BIN_2 and BIN_3 do not contain A-, B_2-, C_2- or D_2-items. Consequently, the B_3-item y is the largest item in these two bins, and FFD puts it into BIN_2.

□

Claim 3.5. *In the FFD-packing* BIN_3 *can not contain any* C_i*-item* x *with* $i \geq 4$.

Proof. We denote the overall size of all items in BIN_2 that are larger than the C_i items by Y. All of these items are at least C_{i-1}-items, and by Fact 3.1 (iii), the total weight of them is at least $(m_{i-1}+1)Y/m_{i-1}$. Let the number of C_i-items in BIN_2 be c, let their overall size be denoted by C. Since the contents of BIN_2 is not a good subset, $W(\mathrm{BIN}_2) < 1$ holds and this yields

$$\frac{m_{i-1}+1}{m_{i-1}}Y + \frac{m_i+1}{m_i}C < 1. \tag{31}$$

Since the item $x \leq 1/m_i$ did not fit into BIN_2, we have

$$Y + C > \frac{m_i - 1}{m_i}. \tag{32}$$

We multiply the inequality (32) by $(m_{i-1}+1)$ and the inequality (31) by the factor m_{i-1}. Subtracting one inequality from the other and simplifying gives

$$C > \frac{m_i - m_{i-1} - 1}{m_i - m_{i-1}} = 1 - \frac{1}{m_i - m_{i-1}}. \tag{33}$$

Moreover, $C \leq c/m_i$ holds. Combining this with inequality (33), we drive

$$c > m_i - \frac{m_i}{m_i - m_{i-1}} > m_i - 2, \tag{34}$$

where the righthand inequality in (34) follows from $i \geq 4$. Summarizing, we have at least $(m_i - 1)$ C_i-items in BIN_2 and at least one C_i-item in BIN_3. This gives m_i item with overall weight at least one and overall size at most one, and we can construct a good subset again.

□

Claim 3.6. *In the FFD-packing* BIN_3 *can not contain* C_3*-item* x.

Proof. If BIN_2 does not contain a B_3-item, it must contain seven C_3-items with overall weight at least one (otherwise FFD puts x into BIN_2, too). Hence, we may assume that BIN_2 does not contain any B_3-item y.

Similarly, as in the proof of Claim 3.3, we denote the overall size of all items in BIN_1 that are larger than the B_3-items (i.e. larger than $1/6$) by X. By Fact 3.1 (iii), the total weight of these items is at least $4X/3$. Let the number of B_3-items in BIN_1 be b, and let

their overall size be denoted by B. Then $W(\mathrm{BIN}_1) < 1$ implies $4X/3+B+b/42 < 1$. As the C_3-item $x \leq 1/7$ did not fit into bin BIN_1, we have $X + B > 6/7$. These two inequalities yield $14B > b + 6$. Finally, we plug in $B \leq b/6$ and derive $b > 9/2$. Altogether, there are at least six B_3-items, and once more we detected a good subset of items with size at most one and weight at least one.

□

Summarizing, in Claims 3.1 through 3.6 we have shown that in the FFD-packing BIN_3 can neither contain an A-, nor a B_i-, a C_i-, or a D_i-item. Consequently, BIN_3 is empty and the FFD-packing is a good packing. This is the final contradiction and the proof the Theorem 3.2 is complete.

□

Theorem 3.3. (Galambos and Woeginger, [51]). *The algorithm* REP_3 *can be implemented in such a way that it never gets stuck in Step(2). It has the best possible worst case ratio* $h_\infty(1)$*, and a running time of* $O(n^2)$*, where* n *denotes the number of items.*

Proof. The proof that REP_3 can be implemented so that it never gets stuck is done by induction over the number of packed items. We will keep the following three invariants. As REP_3 receives a new item to pack, (i) BIN_3 is empty, (ii) $\mathrm{BIN}_1 \cup \mathrm{BIN}_2$ does not contain any A-item and (iii) $\mathrm{BIN}_1 \cup \mathrm{BIN}_2$ contains at most one B_2-item.

Obviously, the three invariants hold as REP_3 receives the first item. Assume they hold after the packing of item a_k, and consider the moment as REP_3 receives a_{k+1}. In Step (1), item a_{k+1} is put into BIN_3. If a_{k+1} is an A-item, we just move on to Step (3) where BIN_3 is closed and replaced by an empty bin. Obviously, all three invariants hold again. Otherwise, a_{k+1} has size at most $1/2$ and we distinguish three subcases.

(a) If a_{k+1} is a B_2-item and $\mathrm{BIN}_1 \cup \mathrm{BIN}_2$ contains some B_2-item z, we simply move z into BIN_3 and go to Step (3).

(b) $\mathrm{BIN}_1 \cup \mathrm{BIN}_2$ does not contain any B_2-item. In this case, we apply Theorem 3.2. Either we repack all items into two bins so that BIN_3 is empty (in this case we go to Step (3) and all three invariants are fulfilled), or we detect a good subset $\mathcal{S}$. All items in $\mathcal{S}$ are of size at most $1/3$, and from Fact 3.1 (iii) we derive that the total size of $\mathcal{S}$ is at least $3/4$.

We pack all items of $\mathcal{S}$ into BIN_3, and a_{k+1} moves into BIN_1. The remaining pieces are packed by FFD into BIN_1 and BIN_2. We claim that all pieces can be packed. Assume that some piece z does not fit in any bin. If $z > 1/4$, then at least one larger piece has been packed into BIN_1, and the contents of BIN_1 is at least $a_{k+1} + 1/4$. If $z \leq 1/4$, then the contents of BIN_1 is at least $3/4 \geq 1/4 + a_{k+1}$. Thus, in any case the contents of BIN_1 is at least $a_{k+1}+1/4$. But now we have derived a contradiction: The overall size of all pieces is at most $2 + a_{k+1}$. BIN_3 contains at least $3/4$, BIN_1 contains at least $a_{k+1}+1/4$, and all the remaining pieces fit into BIN_2. Consequently, we may go on to Step (3).

(c) If $a_{k+1} \leq 1/3$ and $\mathrm{BIN}_1 \cup \mathrm{BIN}_2$ contains some B_2-item of size greater than $1/3$, we proceed similarly to subcase (b). We apply Theorem 3.2. If we can repack all items into two bins so that BIN_3 is empty, everything is fine. Hence assume we detect a good subset $\mathcal{S}$. All items in $\mathcal{S}$ are of size at most $1/2$, and from Fact 3.1 (iii) we derive that the total size of $\mathcal{S}$ is at least $2/3$.

Again we pack all items of $\mathcal{S}$ into BIN_3, a_{k+1} into BIN_1 and apply FFD-packing to the remaining pieces. If some piece z could not be packed, it is easy to see that $z \leq 1/3$ and the contents of BIN_1 must be at least $2/3 \geq 1/3 + a_{k+1}$. We end with a contradiction as in subcase (b).

This completes the inductional proof.

To prove that REP_3 has the best possible worst case ratio $h_\infty(1)$, we simply observe that for any list L of items, $\mathrm{REP}_3\, 3(L) \leq W(L) + 3$ holds. (The algorithm closes only bins of weight at least one, the last three active bins are added to this number.) Combining this with the inequality in Lemma 3.1, we get

$$\mathrm{REP}_3(L) - 3 \leq W(L) \leq h_\infty(1)\mathrm{OPT}(L).$$

Together with the lower bound given in [77] this implies $R^\infty_{\mathrm{REP}_3\, 3} = h_\infty(1)$.

Finally, we consider the time complexity of $\mathrm{REP}_3\, 3$. If we store all items in the active bins in a sorted binary tree, inserting and removing some item can be done in $O(\log n)$ time. Hence, all the sorting steps can be performed in overall time $O(n \log n)$. Every time a new item arrives, we have to repack at most n items. This gives an overall time complexity of $O(n^2)$.

□

3.3 Upper bound for semi-online algorithms with restricted repacking

In this section we deal with semi-online algorithms which also allow repacking, but we will consider that type of semi-online algorithms which allow to repack a *restricted number of items*. We call these algorithms *k-repacking* semi-online algorithms. Gambosi, Postiglione, and Talamo [55] analysed such semi-online algorithms. In their algorithms repacking was allowed in a special way: elements which were "large enough" were repacked one by one, while the "small" items were moved together in a bundle among the bins. In this sense these algorithms can repack even $O(n)$ elements in one step. They analysed two algorithms. The faster one has linear time and is $\frac{3}{2}$-competitive, the other one runs in $O(n \log n)$ time and is $\frac{4}{3}$-competitive.

Based on our paper [9] in this part we will concentrate on *k-repacking semi-online algorithms*. No other semi-online activity is allowed. Along the next sections we assume that repacking one item has a unit cost independently from the size of the item.

3.3.1 The algorithm and its time complexity

We start by making some formal definitions and defining our notations used here. First, let x denote an arbitrary item of the input list. Then we will refer to its size just by x, if the context is clear.

Definition 3.1. *For an open bin B, we denote the sum of the sizes of items presently packed into B by $level(B)$. Clearly, $0 < level(B) \leq 1$.*

Definition 3.2. *We call an item small if its size is in the interval $(0, \frac{1}{2}]$ otherwise the item is large. A bin B is a large bin if it contains large item.*

For each $k \in N^+$ we denote the unique such solution of the equation $2kx^2 - (6k + 3)x + 1 = 0$, which is in the interval $(0, \frac{1}{6k})$ by b_k. (The precise way of getting the b_k values will be shown later.)

Let k be an arbitrary nonnegative integer. The interval $(0, 1]$ is divided into $2k + 3$ disjoint subintervals $I_j, j = 1, \ldots, 2k + 3$, in the following way

$$\begin{aligned}
(0, b_k] &=: I_1, \\
\left(b_k, \frac{1}{2} - kb_k\right] &=: I_2, \\
\left(\frac{1}{2} - jb_k, \frac{1}{2} - (j-1)b_k\right] &=: I_{k+3-j}, \quad j = 1, \ldots, k, \\
\left(\frac{1}{2} + (j-1)b_k, \frac{1}{2} + jb_k\right] &=: I_{k+2+j}, \quad j = 1, \ldots, k, \\
\left(\frac{1}{2} + kb_k, 1\right] &=: I_{2k+3}.
\end{aligned}$$

It is easy to see that

$$\begin{aligned}
\bigcup_{j=1,\ldots,k} I_{k+3-j} &= \left(\frac{1}{2} - kb_k, \frac{1}{2}\right], \\
\bigcup_{j=1,\ldots,k} I_{k+2+j} &= \left(\frac{1}{2}, \frac{1}{2} + kb_k\right],
\end{aligned}$$

With each interval I_j we associate a class of bins denoted by $\mathcal{B}_j, j = 1, \ldots, 2k + 3$. A bin will be assigned to class $\mathcal{B}_j$ if the size of the smallest item put into it is in the interval I_j. Note that the classes of the bins may change during the packing, the exact rules will be given later.

Definition 3.3. *We call two classes $\mathcal{B}_j$, $1 \leq j \leq k + 2$, and $\mathcal{B}_l$, $k + 3 \leq l \leq 2k + 3$, complementary classes if $x_j + x_l \leq 1$ for all $x_j \in I_j$ and $x_l \in I_l$. The set of the complementary classes w.r.t. a class $\mathcal{B}_i$ is denoted by $C(\mathcal{B}_i)$, $1 \leq i \leq 2k + 3$.*

Note that $C(\mathcal{B}_{k+2})$ and $C(\mathcal{B}_{2k+3})$ are empty. Clearly, if l^* denotes the largest index of the classes in $C(\mathcal{B}_j)$, $1 \leq j \leq k+1$, then $l^* = \min\{2k+2, 2k+4-j\}$. Similarly, if $\mathcal{B}_j \in C(\mathcal{B}_l)$ for $k+3 \leq l \leq 2k+2$, then $1 \leq j \leq 2k+4-l$.

Basically, our algorithm will pack the items using the so-called *Harmonic Fit* rule (HF) [77], and therefore we call it *Harmonic Repacking*, and is denoted by HR-k. Let x be the current item to be packed and suppose that $x \in I_j$. If B is the last opened bin in class $\mathcal{B}_j$ and $level(B) + x \leq 1$, then we put x into B. Otherwise, we open a new bin of class $\mathcal{B}_j$ and put x into this bin. When applying the rule HF, it is easy to see that in each class $\mathcal{B}_i$, $i \leq k+2$ or $i = 2k+3$, all bins B – except for possibly the last one – being $level(B) \geq \frac{1}{2} + kb_k$. (In the case of $i = 2k+3$ it is trivial. In the remaining cases if $i > 2$, each bin of a class – except the last opened one for possibly – contains at least two items, and their total size is at least $1 - 2kb_k$, which with the choice of b_k is at least $\frac{1}{2} + kb_k$. In the case of $i \leq 2$ it can be readily seen that any bin B of these classes – except for possibly one – $level(B) \geq \frac{1}{2} + kb_k$.) So, if the list does not contain items belonging to other intervals, one can see with little effort (observing that that kb_k never exceeds $1/6$, and approaches this value for large k) that an *asymptotic competitive ratio* of value $\frac{3}{2}$ can be attained if k goes to infinity. In the rest part of the section we mostly focus on the other case (when the list contains item(s) from the interval $I_{k+3} \cup \ldots \cup I_{2k+2}$), enhancing the HF rule in order to handle this case.

Evidently, the HF rule puts only one single item into bins of classes $\mathcal{B}_i$, $k+3 \leq i \leq 2k+2$. Our algorithm will improve this by filling up these bins with other items, if their sizes allow it. Furthermore, repacking is possible, i.e., in each step the bin assignment can change for a limited number of items. Consequently, in our algorithm the number of currently used bins can change dynamically since we may open new bins and bins may become empty. To compute $A(L)$, we consider only those bins which are nonempty after the algorithm has packed all of the items of the list.

Our algorithm also differs from the "classical" Harmonic Fit algorithm in the treatment of bins within the classes: we *never close them*, so their contents will be accessible during the whole execution of the algorithm (additional elements can be packed into the bin or repacked from the bin into another one). Thus, it is *not a bounded-space algorithm*. We enhance the standard HF rule based on the following two refinements.

- **Rule 1 (Refill)**: If x is the current item to be packed and $x \in I_j$ is a small item with $1 \leq j \leq k+1$, then we examine the complementary classes of $\mathcal{B}_j$ in increasing order of their indices. If we find a nonempty class $\mathcal{B}_l$, we put the item into the last opened bin within the class. Let this bin be B. If $level(B) + x \notin I_l$ then we reassign B to the class $\mathcal{B}_t$ where $level(B) + x \in I_t$.

  Clearly, if $x \notin I_1$, then after having packed it into a bin B this bin will always change its class.

  If all complementary classes are empty, we use the HF rule to pack this small item, but we do not close the bins opened previously in its class.

- **Rule 2 (Repack)**: If x is the current item to be packed and $x \in I_l$ is a large item

with $k+3 \leq l \leq 2k+2$, then we open a new bin in class $\mathcal{B}_l$ and put x into this bin. Furthermore, we look for nonempty bins in the complementary class $\mathcal{B}_j \in C(\mathcal{B}_l)$ in decreasing order of their indices. If there exists such a bin, we repack one item from this bin into the last opened bin in the class $\mathcal{B}_l$. If the level of the large bin changes its interval (which will always happen if $j \neq 1$), we reassign it to the respective class.

(We need to take into account the fact that bins may become empty if their last small item is repacked.)

It is worth noting that as a final step while packing an item (at the end of both rules), we try to carry out a further possible repacking if the bin in class $\mathcal{B}_l$, $k+3 \leq l \leq 2k+2$, has changed its class, and for the new $\mathcal{B}_t$, $t \neq 2k+3$. This results in the call of Rule 2 (Repack) from both rules and it means that Repack is a recursive procedure (because after the first call of Repack, it can be called recursively by itself). In both cases it must be ensured that at most k number of items are repacked for each step.

It should be mentioned that the above description of the two rules (and the third rule which is the recursive call of Repack) is only a brief description of the algorithm. The precise description of the algorithm (parameter values passed to procedures, etc.) is given below in the pseudocode listing. In the description we use the notations $x_{HF} \to \mathcal{B}_t$ and $x \to \mathcal{B}_t$ if we pack the actual item x using the HF rule, or if we simply pack x into the last opened bin of the class $\mathcal{B}_t$, respectively. Similarly, we will use the notation $B \to \mathcal{B}_t$ to mean the rearrangement of a bin from its previously class to the class $\mathcal{B}_t$.

Algorithm 1 : Algorithm HR-k

```
1.  while there exist unpacked items do
2.      input next x;
3.      if x ∈ I_l, k+2 ≤ l ≤ 2k+3, then
4.          x_HF → B_l;
5.          if x ∉ I_{k+2} and x ∉ I_{2k+3} then
6.              call Repack(l, 2k+3−l);
7.          endif
8.      endif
9.      else if x ∈ I_j, 1 ≤ j ≤ k+1, then
10.         call Refill(x);
11.     endelse
12. endwhile
```

The main program and the procedure Refill are self-explanatory. The parameter of Refill is the (size of the) item which we attempt to fit into a large bin. For Repack, the most important note is that this procedure can call itself recursively. Its first parameter l is the index of the class of the large item which we try to match from a complementary

class. In the first call, the second parameter is always $2k+4-l$, but in a series of recursive calls it is monotonically decreasing.

Algorithm 2 : Procedure Refill(x)

1. $j :=$ the index of the class for which $x \in I_j$;
2. $l^* := \min\{2k+4-j, 2k+2\}$; (the largest index of the classes of complements of $\mathcal{B}_j$)
3. **do** $l = k+3$ **to** l^*
4. **if** $\mathcal{B}_l \neq \emptyset$ **then**
5. $x \to \mathcal{B}_l$; Let B the bin into which x is packed.
6. **if** $level(B) \in I_p$, $p \neq l$, **then**
7. $B \to \mathcal{B}_p$;
8. **if** $p < 2k+3$ **then**
9. **call** Repack(p, $2k+4-p$);
10. **endif**
11. **endif**
12. **return** ; (either no change between classes or we have returned from Repack)
13. **endif**
14. **enddo**
15. $x_{HF} \to$ to the last opened bin of the class $\mathcal{B}_j$; (there was no complementary class in the do-loop)

The next three statements are useful for analysing the time complexity of the algorithm.

Lemma 3.2. *Algorithm* HR-k *terminates in finite steps.*

Proof. The main loop will be executed exactly n times, namely once for every item in L. The Refill procedure will be called at most once for each item of the list.

Although Repack may call itself recursively, this can only happen if the currently used bin changes its class and its level is at most $\frac{1}{2} + kb_k$. It follows that in the sequence of possible recursive calls the first parameters of Repack form a strictly monotonically increasing sequence: in the first call we have $l \geq k+3$ and for the last one $l \leq 2k+2$. Therefore the number of recursive calls is at most $k-1$. From this we can conclude that the algorithm calls Repack at most k times for each item of the list. Since Repack is called either from the main loop or at most once for every call of Refill, the statement follows. □

Having demonstrated the truth of this lemma, the next statement follows immediately.

Corollary 3.1. HR-k *repacks at most k items in each step.*

Lemma 3.3. *The number of inspections of the contents of nonempty classes $\mathcal{B}_i$ is at most $(2k+2)n$.*

Algorithm 3 : Procedure Repack(l, j)

(We try to repack a small item from a bin of the class $C(\mathcal{B}_l)$ into the last bin of $\mathcal{B}_l$, $l \geq k+3$. Here j is the first index of the classes in $C(\mathcal{B}_l)$ where we need to start the search.)

```
1.  do t = j to 1
2.      if B_t ≠ ∅ then
3.          x ∈ B ∈ B_t → B_l; (B is the last opened bin of B_t and x is its topmost
4.          item);
5.          if level(B) ∈ I_s, l + 1 ≤ s ≤ 2k + 3, then
6.              B → B_s;
7.              if s < 2k + 3 then
8.                  call Repack(p, s);
9.              endif
10.         endif
11.     endif
12. enddo
```

Proof. We will show that for each step the algorithm cannot test the emptiness of more than $2k+2$ bin classes.

In the main program there is no such test, and in Refill there are at most k comparisons for each item.

The procedure Repack has two parameters. The first one is l, where $l = k+2+i$, for some $1 \leq i \leq k$, and the second one is at most $2k+4-l = k+2-i$. The values of the second parameters of the recursive calls are monotonically, but not strictly decreasing. (The second parameter j is an index of a bin-class of small items from which we attempt to repack a small item in each call of Repack. If this bin-class is empty, we continue the examination of the bin-classes of small items in decreasing index-order.) Since the procedure Repack will be called at most $k+1-i$ times for each item of a given list, the algorithm performs at most $(k+2-i)+(k+1-i) = 2k+3-2i$ comparisons.
While packing the current item, if Repack is called for the first time from procedure Refill, it makes at most $i-1$ additional comparisons in that procedure. Otherwise, if the first call of Repack is executed directly from the main loop, there is no additional comparison. □

Combining these three statements yields

Theorem 3.4. (Balogh, Békési, Galambos and Reinelt, [9]). *The time complexity of* HR-k *is* $O(kn)$.

3.3.2 Asymptotic competitive ratio of HR-k

After having packed all of the items of a given list L we will distinguish two cases:

Case A: All of the classes $\mathcal{B}_{k+3}, \ldots, \mathcal{B}_{2k+2}$ are empty.
Case B: There is at least one nonempty class among $\mathcal{B}_{k+3}, \ldots, \mathcal{B}_{2k+2}$.

Lemma 3.4. *If all items of a given list L have been packed by* HR-k *and Case A holds, then*

$$\text{HR-k}\,(L) \leq \frac{2}{1+2kb_k}\,\text{OPT}(L) + (k+2).$$

Proof. In this case all opened bins have been packed at least to level $\frac{1}{2} + kb_k$, except for possibly the last opened bins of the classes $\mathcal{B}_1, \ldots, \mathcal{B}_{k+2}$. The number of such bins is at most $k+2$. Therefore,

$$\text{OPT}(L) \geq \left(\frac{1}{2} + kb_k\right)\text{HR-k}\,(L) - (k+2)$$

and the statement follows. □

For Case B we will apply the standard weight function technique. (In Lemma 3.4 we analysed Case A without dealing explicitly with a weighting function. However, a weight function would obviously be defined in Case A as well, assigning a weight $w(x) = \frac{1}{\frac{1}{2}+kb_k}x$ to each item $x \in (0,1]$. Then the weight of an item would depend on whether Case A or Case B holds after the algorithm has finished the packing of the input list.) Before we define the weight function we need a lemma.

Lemma 3.5. *Suppose that all items of a given list L have been packed by* HR-k *and Case B holds. Let $k+2+i^*$ be the smallest index of such a nonempty class. Then all complementary classes of $\mathcal{B}_{k+2+i^*}, 1 \leq i^* \leq k$, are empty except possibly $\mathcal{B}_1$.*

Proof. Since $\mathcal{B}_{k+2+i^*}$ is not empty, it contains at least one nonempty bin B_1, and B_1 is a large bin.

Suppose, the contrary, that there exists a nonempty complementary class of $\mathcal{B}_{k+2+i^*}$ and it is not $\mathcal{B}_1$. Let j_0 be the index of the largest nonempty class and let $B_2 \in \mathcal{B}_{j_0}$ be the last opened bin in this class. Then B_2 contains at least one small item. Let x be the top item in B_2. There may be two different scenarios.

- In the first case the arrival of $x \in L$ precedes the packing (or repacking) of y, where item y has been packed as the last item into B_1. Since x preceded the packing (or repacking) of y HR-k will pack x (or another small item) into B_1.

- Otherwise, if the arrival of x was later than the packing (or repacking) of item y, then the algorithm has packed x into B_1 (or into another large bin) after it had placed y into B_1.

Both cases lead to a contradiction.

□

Now we define our weight function $w(x)$. Let i^* be defined as in Lemma 3.5. In the following we will just write b instead of b_k if k is fixed (and clear from the context). Then

$$w(x) := \begin{cases} \frac{x}{1-b}x, & x \in \{I_1 \cup I_2 \cup \ldots \cup I_{k+2-i^*}\}, \\ \frac{1}{2}, & x \in \{I_{k+3-i^*} \cup \ldots \cup I_{k+2}\}, \\ 1 - \frac{1}{1-b}\big(\frac{1}{2} + (i^*-1)b - x\big), & x \in \{I_{k+3} \cup \ldots \cup I_{k+1+i^*}\} (= \emptyset, \text{ if } i^* = 1), \\ 1, & x \in \{I_{k+2+i^*} \cup \ldots \cup I_{2k+3}\}. \end{cases}$$

Note that in the case of $i^* = 1$ the weight of any large item is 1. If some items have arrived from the intervals $I_2, I_3, \ldots, I_{k+2-i^*}$ they must have been packed in a large bin, because the $\mathcal{B}_2, \mathcal{B}_3, \ldots, \mathcal{B}_{k+2-i^*}$ bin classes are empty (see Lemma 3.5).

In the case of $i^* > 1$ the classes $\mathcal{B}_{k+2+j}$, $j = 1, \ldots, i^* - 1$ are empty at the end. That is, if some items have arrived from these intervals after finishing the packing they must be in bins of classes for which the index of the class is at least $k + 2 + i^*$. For such a large item we assign a weight w, which is less than 1. To guarantee that the weight of such a bin is at least 1, for the small items of this bin we assign a weight so that the total weight of the small items in the bin is at least $1 - w$.

In the proof of the next lemma we will show this, and also that in Case B the weight of each bin is at least 1, except for at most one bin of the class $\mathcal{B}_1$ and one bin of each of the classes $\mathcal{B}_{k+3-i^*}, \ldots, \mathcal{B}_{k+2}$.

Lemma 3.6. *If Case B occurs and $k + 2 + i^*$ is the smallest index for which $\mathcal{B}_{k+2+i^*}$ contains at least one nonempty bin, then for any list L*

$$w(L) = \sum_{x \in L} w(x) \geq \text{HR-k}(L) - (k+1).$$

Proof. We will show that in each class, except for at most one bin in each class, the sum of the weights of the items in a bin is at least 1.

- If $B \in \mathcal{B}_1$ and it is not the last opened nonempty bin, then $level(B) > 1 - b$. If $x \in B$ then $w(x) = \frac{1}{1-b}x$. Therefore $w(B) = \sum_{x \in B} w(x) > 1$.
- The class $\mathcal{B}_i$ is empty, where $2 \leq i \leq k + 2 - i^*$ (see Lemma 3.5).
- If $B \in \mathcal{B}_j$, $k + 3 - i^* \leq j \leq k + 2$, and it is not the last opened nonempty bin in its class, then it contains exactly two items x and y with $w(x) = w(y) = \frac{1}{2}$.
- If $B \in \{\mathcal{B}_{k+2+i^*}, \ldots, \mathcal{B}_{2k+3}\}$, then $level(B) > \frac{1}{2} + (i^* - 1)b$ and the bin contains one large item x. Here, there are two distinct cases.
  - If $x > \frac{1}{2} + (i^* - 1)b$, then $w(x) = 1$, so in this case $w(B) \geq 1$.

– If $x \in \left(\frac{1}{2}, \frac{1}{2} + (i^* - 1)b\right]$, then $w(x) = 1 - \frac{1}{1-b}(\frac{1}{2} + (i^* - 1)b - x)$ and the bin contains small items with cumulative size $level(B) - x \geq \frac{1}{2} + (i^* - 1)b - x$. The weight of these small items (from the definition of the weight function) is at least $\frac{1}{1-b}\left(\frac{1}{2} + (i^* - 1)b - x\right)$. Thus $w(B) \geq 1$ in the second case as well.

We note that the number of "exceptional" bins is at most 1 in the nonempty bin class and a bin containing a large item cannot be an exception, so the total number of exception bins is at most $k + 1$, and the claim of the lemma is true.

□

Lemma 3.7. *If Case B holds and S is a subset of the items from L with $\sum_{x \in S} x \leq 1$, then*

$$w(S) = \sum_{x \in S} w(x) \leq \frac{3}{2} + \frac{b}{1-b}.$$

Proof. If S contains only small items, then $w(x) \leq \frac{3}{2}x$ for $x \in S$ by definition of w. Therefore

$$w(S) = \sum_{x \in S} w(x) < \sum_{x \in S} \frac{3}{2}x \leq \frac{3}{2}.$$

If S contains a large item x_1, for which $x_1 > \frac{1}{2} + kb$ holds, then $w(x_1) = 1$ and $\sum_{x \in S, x \neq x_1} x \leq 1 - x_1 = \frac{1}{2} - kb$. But if $x < \frac{1}{2} - kb$, then $w(x) \leq \frac{1}{1-b}x$, so

$$\begin{aligned} w(S) &= \sum_{x \in S} w(x) \leq 1 + \sum_{x \in S, x \neq x_1} \frac{1}{1-b}x \\ &\leq 1 + \frac{1}{1-b}\left(\frac{1}{2} - kb\right) = 1 + \frac{1}{1-b}\left(\frac{1}{2} - \frac{1}{2}b + \frac{1}{2}b - kb\right) \\ &= \frac{3}{2} - \frac{(k - \frac{1}{2})b}{1-b} \leq \frac{3}{2} - \frac{b}{2(1-b)} < \frac{3}{2}. \end{aligned}$$

If $x_1 \in S$ and $\frac{1}{2} < x_1 \leq \frac{1}{2} + kb$ then x_1 is the largest item in S. By the assumption that Case B holds, there exists a positive integer i, $1 \leq i \leq k$, such that $x_1 \in I_{k+2+i}$.

There are two cases that need to be examined:

a) If $x_1 > \frac{1}{2} + (i^* - 1)b$, then $i \geq i^*$. If x_2 is the second largest item in S, then we have two subcases:

– If $x_2 \in \{I_1 \cup \ldots \cup I_{k+2-i}\}$ then for each item x ($x \neq x_1$), $x \leq x_2$, so $x \in \{I_1 \cup \ldots \cup I_{k+2-i}\}$ holds as well. As regards the weight of a small item like x, $w(x) \leq \frac{1}{1-b}x$ holds, so

$$\begin{aligned} w(S) &= \sum_{x \in S} w(x) = w(x_1) + \sum_{x \in S, x \neq x_1} w(x) \\ &\leq 1 + \frac{1}{1-b} \cdot \frac{1}{2} = 1 + \frac{1}{2-2b} = \frac{3-2b}{2-2b} = \frac{3}{2} + \frac{b}{2(1-b)}. \end{aligned}$$

– if $x_2 \in \{I_{k+3-i} \cup \ldots \cup I_{k+2}\}$, then $x_2 \in I_{k+3-i}$, because otherwise $x_1 + x_2 > 1$. But in this case $\sum_{x \in S, x \neq x_1, x \neq x_2} x \leq 1 - x_1 - x_2 \leq 1 - (\frac{1}{2} + (i-1)b) - (\frac{1}{2} - ib) = b$. It means that $x \in I_1$. Hence

$$\begin{aligned} w(S) &= \sum_{x \in S} w(x) = w(x_1) + w(x_2) + \sum_{x \in S, x \neq x_1, x \neq x_2} w(x) \\ &\leq 1 + \frac{1}{2} + \frac{b}{1-b} = \frac{3}{2} + \frac{b}{1-b}. \end{aligned}$$

b) if $x_1 \leq \frac{1}{2} + (i^* - 1)b$, then $x_1 \in I_{k+2+i}$, when $1 \leq i < i^*$. If x_2 is the second largest item in S, then we also have two subcases:

– If $x_2 > \frac{1}{2} - i^*b$, then $w(x_2) = \frac{1}{2}$, and $\sum_{x \in S, x \neq x_1, x \neq x_2} x \leq 1 - x_1 - x_2 < 1 - x_1 - (\frac{1}{2} - i^*b) = \frac{1}{2} + i^*b - x_1$.

From this

$$\begin{aligned} w(S) &= \sum_{x \in S} w(x) = w(x_1) + w(x_2) + \sum_{x \in S, x \neq x_1, x \neq x_2} w(x) \\ &\leq 1 - \frac{1}{1-b} \cdot \left(\frac{1}{2} + (i^* - 1)b - x_1\right) + \frac{1}{2} + \frac{1}{1-b}\left(\frac{1}{2} + i^*b - x_1\right) \\ &= \frac{3}{2} + \frac{b}{1-b}. \end{aligned}$$

– If $x_1 \leq \frac{1}{2} + (i^* - 1)b$ and $x_2 \leq \frac{1}{2} - i^*b$, then for all $x \neq x_1$ items of S $w(x) \leq \frac{1}{1-b}x$ holds. As before, (case a), first subcase) it can be shown that $w(S) \leq \frac{3}{2} + \frac{b}{2(1-b)}$ also applies in this case.

This completes the proof of the lemma.

□

If we compare the results of Lemmas 3.4 and 3.7, we see that in both cases for given $k \in N^+$ we can fix a real number b which satisfies the desired condition $kb < \frac{1}{6}$. But in this case we will get different upper bounds for HR-k. The two multiplicative constants will be equal if the equation

$$\frac{2}{1 + 2kb_k} = \frac{3}{2} + \frac{b_k}{1 - b_k}$$

has a solution for every fixed k satisfying the condition $kb_k < \frac{1}{6}$. One can easily check that the equation $2kb_k^2 - (6k+3)b_k + 1 = 0$ has such a solution, and only one solution of this type. That is

$$b_k = \frac{6k + 3 - \sqrt{\left(6k + \frac{7}{3}\right)^2 + \frac{32}{9}}}{4k}.$$

We list some values for the respective pairs k and b_k in Table 3.5.

Let $M_1 = k + 2$, from Lemma 3.4 and $M_2 = k + 1$ from Lemma 3.6, and let $M := \max\{M_1, M_2\} = k + 2$. Combining the results of Lemmas 3.4, 3.6, and 3.7 we get

$$\text{HR-k}(L) \leq \max\left\{\frac{2}{1+2kb_k}, \frac{3}{2} + \frac{b_k}{1-b_k}\right\}\text{OPT}(L) + M.$$

Now, we can state an upper bound on the asymptotic competitive ratio of our algorithm for all k values and state the limes of these ratios, taking into consideration for the latter one that $b_k \in \left(0, \frac{1}{6k}\right)$, for any k values.

Theorem 3.5. (Balogh, Békési, Galambos and Reinelt, [9]). *For any fixed $k \in N^+$*

$$R^{\infty}_{\text{HR-k}} \leq \frac{3}{2} + \frac{b_k}{1-b_k},$$

and $R_{\text{HR-k}} \to \frac{3}{2}$ for $k \to \infty$.

The next theorem provides a lower bound. Though this bound does not quite match the upper bound, the results are fairly close.

Theorem 3.6. (Balogh, Békési, Galambos and Reinelt, [9]). *For any fixed $k \in N^+$*

$$R^{\infty}_{\text{HR-k}} \geq \left(1 - \frac{b_k^2}{(1-b_k)^2}\right)\left(\frac{3}{2} + \frac{b_k}{1-b_k}\right).$$

Proof. For each t, $t \in N^+$, let $n = 2t(\lfloor \frac{1}{b_k} \rfloor - 1)$. For each given n we construct a list L with $N = 4n + 8t - 3$ items as a concatenation of four sublists, $L = L_1L_2L_3L_4$ as follows.

(i) $L_1 = (L_{11}L_{12}L_{13})^{2t}$ (the exponent gives the number of repetitions of the concatenated sublists) where

- L_{11} contains $\lfloor \frac{1}{b_k} \rfloor - 1$ items, each with size $b_k - 2\varepsilon$,
- L_{12} contains one item with size $1 - b_k\left(\frac{n}{2t} + 1\right) + \frac{n}{t}\varepsilon$,
- L_{13} contains three items with sizes ε, where

$$\varepsilon < min\left\{\frac{b_k}{n}, \frac{1}{6t+n-3}, \frac{2b_kt + \frac{nb_k}{2} - t}{n}\right\}.$$

(ii) L_2 contains $n - 3$ items of size ε.

(iii) L_3 contains n items of size $\frac{1}{2} - b_k + \varepsilon$.

(iv) L_4 contains of n items of size $\frac{1}{2} + \varepsilon$.

HR-k packs the items of this list in the following way. First, it packs the items of the first list $L_{11}L_{12}L_{13}$ into bins of class $\mathcal{B}_1$ using the pure HF rule. The level of each of these bins will be exactly $1 - b_k + 3\varepsilon$, so the first item of the next $L_{11}L_{12}L_{13}$ must be put into a newly opened bin. Therefore the algorithm uses

$$2t = \frac{n}{\lfloor \frac{1-b_k}{b_k} \rfloor}$$

bins to pack the items of L_1.

The items from L_2 belong to I_1, so they will also use bins from $\mathcal{B}_1$. Since the last opened bin in class $\mathcal{B}_1$ is filled to level $1 - b_k + 3\varepsilon$, the elements of L_2 can be put into this bin. The items of L_3 are packed pairwise into bins belonging to the class $\mathcal{B}_{k+2}$. Lastly, when processing the items from L_4, the algorithm opens a new bin of class $\mathcal{B}_{k+3}$ for each item of L_4, and in each step it repacks one item with size ε from the last opened bin of $\mathcal{B}_1$. Summarizing, HR-k uses

$$\text{HR-k}\,(L) = n + \frac{n}{2} + \frac{n}{\lfloor \frac{1-b_k}{b_k} \rfloor} \leq n\left(\frac{3}{2} + \frac{b_k}{1-b_k}\right)$$

bins. But, since the sum of the sizes of the items is at most

$$n + 2t\left(1 - b_k - \frac{n}{2t}(b_k - 2\varepsilon)\right) + 1 \leq n + 2tb_k + 1,$$

we get

$$\text{OPT}(L) \leq n + \frac{nb_k}{\lfloor \frac{1}{b_k} \rfloor - 1} + 1 \leq n + \frac{nb_k}{\frac{1}{b_k} - 2} + 1 = n\frac{(1-b_k)^2}{1-2b_k} + 1.$$

Therefore

$$\begin{aligned} R^{\infty}_{\text{HR-k}} &\geq \lim_{n\to\infty} \frac{n\left(\frac{3}{2} + \frac{b_k}{1-b_k}\right)}{n\frac{(1-b_k)^2}{1-2b_k} + 1} = \lim_{n\to\infty} \frac{\frac{3}{2} + \frac{b_k}{1-b_k}}{\frac{(1-b_k)^2}{1-2b_k} + \frac{1}{n}} \\ &= \left(\frac{3}{2} + \frac{b_k}{1-b_k}\right)\left(\frac{1-2b_k}{(1-b_k)^2}\right), \end{aligned}$$

which yields the statement of the theorem.

□

One consequence of the previous two theorems is

Corollary 3.2. *For any fixed $k \in N^+$*

$$\left(\frac{3}{2} + \frac{b_k}{1-b_k}\right)\left(\frac{1-2b_k}{(1-b_k)^2}\right) \leq R^{\infty}_{\text{HR-k}} \leq \frac{3}{2} + \frac{b_k}{1-b_k}.$$

Table 3.5 shows lower and upper bounds for the most interesting values of k.

k	b_k	**LB**	**UB**	Δ
1	0.113999	1.601704	1.628666	0.026962
2	0.067895	1.564496	1.572841	0.008345
3	0.048285	1.546743	1.550734	0.003991
4	0.037452	1.536580	1.538909	0.002329
5	0.030586	1.530026	1.531551	0.001524
10	0.015953	1.515813	1.516212	0.000398
56	0.002952	1.502961	1.502947	0.000013
167	0.000995	1.500994	1.500996	0.000001

Table 3.5. Values of b_k and the bounds for different values of k.

3.4 Lower bound for semi-online algorithms with restricted repacking

In the previous section we showed a class of k-repacking algorithms with asymptotic competitive ratio at most of of $\frac{3}{2} + \frac{b_k}{1-b_k}$, where b_k is a constant that depends on the number of items that can be repacked in each step. We proved that $\lim_{k\to\infty} R^{\infty}_{\text{HR-k}} = \frac{3}{2}$.

We mentioned some ideas concerning the lower bounds of these types of algorithms. It is natural to ask about the best k-repacking algorithm. The best lower bound for k-repacking semi-online algorithms was given by Ivkovič and Lloyd in [68]. They proved that there is no k-repacking semi-online algorithms with better competitive ratio than $\frac{4}{3}$. (In fact, they proved the lower bound for the class of *fully-dynamic* (FDBP) algorithms. This class can be derived from the *dynamic* bin packing [25] where the items can also depart (*deleted*) and the goal is to minimize the maximum number of bins. During an FDBP algorithm items may be moved (*repacked*) among the bins as the packing is adjusted to accommodate arriving and departing items.)

Using our results published in [5] in this section we improve the lower bound to 1.3871 for the k-repacking semi-online algorithms. It follows from the construction that this lower bound is also valid for k-repacking FDBP algorithms. The last fact is coming from the following observation. While we try to construct lower bounds either for the k-repacking semi-online problem or the k-repacking FDBP we can realize that the two constructions differ slightly from each other: to allow some deletions will not spoil the k-repacking semi-online lower-bound construction for the k-repacking FDBP case.

During our analysis we will use different instruments: LP-techniques will be combined with results from linear algebra, and finally we will solve a nonlinear optimization problem.

3.4.1 Construction of the linear program

For $m \geq 1$, $n \geq 1$, let $x_1, x_2, \ldots, x_m$ $\left(\frac{1}{2} \leq x_1 < x_2 < \ldots < x_m < 1\right)$ be fixed real numbers and $k \geq 1$ be an arbitrary integer. Define $y_i = 1 - x_i\,(i = 1, \ldots, m)$ and $y_{m+1} = 0$. Furthermore let $\varepsilon < \min_{j=1,\ldots,m}\{y_j - y_{j+1}\}$ be an arbitrary small positive number and

$\varepsilon_j := \frac{\varepsilon}{\lceil \frac{n}{2y_j} \rceil}$.

Denote the first list by L_0. Define L_0 as a list of N items of size a, where $a < \frac{\varepsilon_m}{\lceil \frac{n}{y_m} \rceil k}$ and $N := \lfloor \frac{\frac{n}{2}-\varepsilon}{a} \rfloor$. We can see that L_0 contains very small elements with equal, suitably chosen size. The cumulative size of the small items converges to $\frac{n}{2}$ from below, if $n \to \infty$. Denote by L_i $(1 \leq i \leq m, m \geq 2)$ the lists containing big elements. Define L_i as a list of $\lceil \frac{n}{2y_i} \rceil$ items of size $x_i + \varepsilon_i$. This means that the size of the elements are equal in list L_i. We analyse the behavior of an arbitrary k-repacking semi-online algorithm for the lists L_0 and $L_0 L_i$ $(1 \leq i \leq m)$.

We present the lower bound as a solution of a linear programming problem. Consider an arbitrary k-repacking algorithm A, that has to pack list L_0 first. It is clear that the above defined a is a very small number, and the total size of the repackable items in $\lceil \frac{n}{2y_j} \rceil$ $(j = 1, \ldots, m)$ steps is less than ε_m, which is the smallest among $\varepsilon_1, \ldots, \varepsilon_m$. We want to give a lower bound on $A(L_0)$. Bin B is called as y_i-type bin if $level(B) \in (y_{i+1}, y_i]$. Denote by $z_i n$ the cumulative size of the items that have been packed in y_i-type $(i = 1, \ldots, m)$. Then the total number of y_i-type bins is at least $\frac{z_i n}{y_i}$.

Lemma 3.8.

$$\limsup_{n \to \infty} \frac{A(L_0)}{\mathrm{OPT}(L_0)} \geq 1 + \sum_{j=1}^{m} 2 z_j \left(\frac{1}{y_j} - 1 \right). \tag{35}$$

Proof. The cumulative size of the items that have not been packed in any y_i-type bin is $Na - n \sum_{j=1}^{m} z_j$. So we get that

$$A(L_0) \geq \sum_{j=1}^{m} \frac{z_j n}{y_j} + Na - n \sum_{j=1}^{m} z_j = Na + n \sum_{j=1}^{m} z_j \left(\frac{1}{y_j} - 1 \right). \tag{36}$$

Because of the definition of N we have $Na \geq \frac{n}{2} - \varepsilon - a$. This means that

$$A(L_0) \geq n \left(\frac{1}{2} + \sum_{j=1}^{m} z_j \left(\frac{1}{y_j} - 1 \right) \right) - \varepsilon - a. \tag{37}$$

In the following we investigate the behavior of the optimal algorithm on L_0. In the optimal packing each bin B is loaded such that $level(B) > 1 - a$. So we get

$$\mathrm{OPT}(L_0) \leq \frac{Na}{1-a} + 1 \leq \frac{\frac{n}{2} - \varepsilon}{1-a} + 1 \leq \frac{\frac{n}{2}}{1-a} + 1. \tag{38}$$

From (37) and (38) it follows that

$$\limsup_{n \to \infty} \frac{A(L_0)}{\mathrm{OPT}(L_0)} \geq 1 + \sum_{j=1}^{m} 2 z_j \left(\frac{1}{y_j} - 1 \right). \tag{39}$$

□

Lemma 3.9. *For $i = 1, \ldots, m$:*

$$\limsup_{n\to\infty} \frac{A\left(L_0L_i\right)}{\mathrm{OPT}\left(L_0L_i\right)} \geq 1 + y_i + 2y_i\left(\sum_{j=1}^{i-1} z_j\left(\frac{1}{y_j} - 1\right) - \sum_{j=i}^{m} z_j\right). \tag{40}$$

Proof. Consider now the packing of the concatenated list L_0L_i $(1 \leq i \leq m)$ for fixed i. Let B be a bin which contains an item from L_i. Because the size of the big item is $x_i + \varepsilon_i$, the total size of the small elements in the bin is at most $y_i - \varepsilon_i$. This can also happen in such a way, that after L_0 has been packed, the bin contains more items, but during the processing of L_i, some of them have been repacked to other bins. Since the list L_i has exactly $\lceil \frac{n}{2y_i} \rceil$ elements, at most $k\lceil \frac{n}{2y_i} \rceil$ elements could have been repacked. The size of each small item is a, so from the definition of a and ε_i we get that A can repack small items with at most ε_i cumulative size. So we can say that after packing L_0 $level(B) \leq y_i$ must hold for B. From this it follows that the cumulative size of the small items packed together with an item from L_i is $\sum_{j=i}^{m} z_j n$. Other small items from L_0 can be packed in y_j-type bins, where $j = 1, \ldots, i-1$. The total size of the remaining elements of L_0 is $Na - n\sum_{j=1}^{m} z_j$. So we can estimate the number of bins used by A.

$$A\left(L_0L_i\right) \geq \frac{n}{2y_i} + n\sum_{j=1}^{i-1} \frac{z_j}{y_j} + Na - n\sum_{j=1}^{m} z_j, \qquad (i = 1, \ldots, m). \tag{41}$$

For the optimal packing of L_0L_i $(1 \leq i \leq m)$ we get that

$$\mathrm{OPT}\left(L_0L_i\right) \leq \left\lceil \frac{n}{2y_i} \right\rceil + S_i \leq \frac{n}{2y_i} + S_i + 1, \tag{42}$$

where S_i is the total number of bins, which contain items only from L_0 in an optimal packing. A bin containing an L_i-element will be denoted by B_i. We obtain for the cumulative size $small(B_i)$ of the small elements in B_i that

$$\begin{aligned} small(B_i) &\geq \left\lceil \frac{n}{2y_i} \right\rceil (y_i - \varepsilon_i - a) \geq \frac{n}{2} - \left\lceil \frac{n}{2y_i} \right\rceil \varepsilon_i - \left\lceil \frac{n}{2y_i} \right\rceil a \\ &\geq Na - \left\lceil \frac{n}{2y_i} \right\rceil a \geq Na - \varepsilon, \end{aligned}$$

because $a < \frac{\varepsilon}{\left\lceil \frac{n}{2y_i} \right\rceil}$. So

$$S_i \leq 1. \tag{43}$$

From (42) and (43)

$$\mathrm{OPT}\left(L_0L_i\right) \leq \frac{n}{2y_i} + 2. \tag{44}$$

Combining (41) and (44) for each i $(i = 1, \ldots, m)$ we get

$$\limsup_{n\to\infty} \frac{A\left(L_0L_i\right)}{\mathrm{OPT}\left(L_0L_i\right)} \geq 1 + y_i + 2y_i\left(\sum_{j=1}^{i-1} z_j\left(\frac{1}{y_j} - 1\right) - \sum_{j=i}^{m} z_j\right). \tag{45}$$

□

Theorem 3.7. (Balogh, Békési, Galambos and Reinelt, [5]). *The solution of the following linear programming problem is a lower bound for any semi-online k-repacking algorithm:*

$$\begin{aligned}
\min \quad & b & & \\
b \quad & \geq \quad 1 + y_i + 2y_i \left(\sum_{j=1}^{i-1} z_j \left(\frac{1}{y_j} - 1 \right) - \sum_{j=i}^{m} z_j \right), & & (i = 1, \ldots, m), \\
b \quad & \geq \quad 1 + \sum_{j=1}^{m} 2z_j \left(\frac{1}{y_j} - 1 \right), & & (46) \\
z_i \quad & \geq \quad 0, (i = 1, \ldots, m), & & \\
\sum_{j=1}^{m} z_j \quad & < \quad \frac{1}{2}. & &
\end{aligned}$$

Proof. The definition of z_j $(j = 1, \ldots, m)$ obviously implies that $\sum_{j=1}^{m} z_j < \frac{1}{2}$. The statement of the theorem then follows from Lemmas (3.8) and (3.9).

□

3.4.2 Solution of the linear program

To simplify our model we introduce a new variable z, which is the sum of all z_j's, i.e. $z = \sum_{j=1}^{m} z_j$. Using this substitution and some reordering in (46) our LP can be written in the following form:

$$\begin{aligned}
\min \quad & b & & \\
-2y_i z + 2y_i \sum_{j=1}^{i-1} \frac{z_j}{y_j} - b \quad & \leq \quad -(1 + y_i), (i = 1, \ldots, m), & & \\
-2z + 2 \sum_{j=1}^{m} \frac{z_j}{y_j} - b \quad & \leq \quad -1, & & (47) \\
-z + \sum_{j=1}^{m} z_j \quad & = \quad 0 & & \\
z_i \quad & \geq \quad 0, (i = 1, \ldots, m), & & \\
z \quad & < \quad \frac{1}{2}. & &
\end{aligned}$$

Instead of (47) we first solve a linear system of equations related to our problem. The number of equations is $m + 2$. The variables of the system are $z, z_1, \ldots, z_m, b$ while $y_1, \ldots, y_m$ are some fixed parameters, satisfying the conditions of Theorem 3.7.

$$\begin{aligned} -2y_i z + 2y_i \sum_{j=1}^{i-1} \frac{z_j}{y_j} - b &= -(1+y_i), (i = 1, \ldots, m), \\ -2z + 2\sum_{j=1}^{m} \frac{z_j}{y_j} - b &= -1, \\ -z + \sum_{j=1}^{m} z_j &= 0 \end{aligned} \tag{48}$$

Lemma 3.10. *The equation system (48) has a unique solution.*

Proof. Let M be the matrix of (48). Then

$$M = \begin{bmatrix} -2y_1 & 0 & \ldots & 0 & 0 & -1 \\ -2y_2 & 2\frac{y_2}{y_1} & \ldots & 0 & 0 & -1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -2y_m & 2\frac{y_m}{y_1} & \ldots & 2\frac{y_m}{y_{m-1}} & 0 & -1 \\ -2 & \frac{2}{y_1} & \ldots & \frac{2}{y_{m-1}} & \frac{2}{y_m} & -1 \\ -1 & 1 & \ldots & 1 & 1 & 0 \end{bmatrix} \tag{49}$$

Denote by $\det(M)$ the determinant of M. We prove that $\det(M) < 0$, and so from the Cramer's Rule the statement of the lemma follows. Let us expand $\det(M)$ using the last column of M. So we obtain

$$\det(M) = \sum_{i=1}^{m+1} -1\,(-1)^{m+2+i} \det(M_i) = \sum_{i=1}^{m+1} (-1)^{m+1+i} \det(M_i), \tag{50}$$

where M_i is a $(m+1) \times (m+1)$ matrix, which is obtained from M by deleting its last column and i-th row. Expand $\det(M_i)$ by its last row. So

$$\det(M_i) = (-1)(-1)^{m+2} \det(M_{i1}) + \sum_{i=2}^{m+1} (-1)^{m+1+j} \det(M_{ij}). \tag{51}$$

where M_{ij} is a $m \times m$ matrix, which is obtained from M_i by deleting its last row and j-th column. Now we investigate, how M_{ij} looks.

$$M = \begin{bmatrix} -2y_1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ -2y_2 & 2\frac{y_2}{y_1} & \dots & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -2y_{i-1} & 2\frac{y_{i-1}}{y_1} & \dots & 2\frac{y_{i-1}}{y_{j-2}} & 2\frac{y_{i-1}}{y_j} & \dots & 0 & 0 \\ -2y_{i+1} & 2\frac{y_{i+1}}{y_1} & \dots & 2\frac{y_{i+1}}{y_{j-2}} & 2\frac{y_{i+1}}{y_j} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -2y_m & 2\frac{y_m}{y_1} & \dots & 2\frac{y_m}{y_{j-2}} & 2\frac{y_m}{y_j} & \dots & 2\frac{y_m}{y_{m-1}} & 0 \\ -2 & \frac{2}{y_1} & \dots & \frac{2}{y_{j-2}} & \frac{2}{y_j} & \dots & \frac{2}{y_{m-1}} & \frac{2}{y_m} \end{bmatrix} \tag{52}$$

Case A: Suppose that $j \neq i$ and $j \neq i+1$ and consider only the i-th and $i+1$-th columns of M_{ij}. The two columns are the following:

$$\begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 2\frac{y_{i+1}}{y_{i-1}} & 2\frac{y_{i+1}}{y_i} \\ \vdots & \vdots \\ 2\frac{y_m}{y_{i-1}} & 2\frac{y_m}{y_i} \\ \frac{2}{y_{i-1}} & \frac{2}{y_i} \end{bmatrix} \tag{53}$$

It is easy to see that the these columns are not independent, so we obtain that $\det(M_{ij}) = 0$, if $j \neq i$ and $j \neq i+1$.

Case B: If $j = i$ then M_{ii} is a lower triangular matrix of the form:

$$M_{ii} = \begin{bmatrix} -2y_1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ -2y_2 & 2\frac{y_2}{y_1} & \dots & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -2y_{i-1} & 2\frac{y_{i-1}}{y_1} & \dots & 2\frac{y_{i-1}}{y_{i-2}} & 0 & \dots & 0 & 0 \\ -2y_{i+1} & 2\frac{y_{i+1}}{y_1} & \dots & 2\frac{y_{i+1}}{y_{i-2}} & 2\frac{y_{i+1}}{y_i} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -2y_m & 2\frac{y_m}{y_1} & \dots & 2\frac{y_m}{y_{i-2}} & 2\frac{y_k}{y_i} & \dots & 2\frac{y_m}{y_{m-1}} & 0 \\ -2 & \frac{2}{y_1} & \dots & \frac{2}{y_{i-2}} & \frac{2}{y_i} & \dots & \frac{2}{y_{m-1}} & \frac{2}{y_m} \end{bmatrix} \tag{54}$$

From this we get that $\det(M_{11}) = 2^m \frac{1}{y_1}$, $\det(M_{(m+1)(m+1)}) = -2^m y_m$ and that for $2 \leq i \leq m$, $\det(M_{ii}) = -2^m \frac{y_{i-1}}{y_i}$.

Case C: Similarly, if $j = i + 1$ then $M_{i(i+1)}$ is also a lower triangular matrix:

$$M_{i(i+1)} = \begin{bmatrix} -2y_1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ -2y_2 & 2\frac{y_2}{y_1} & \dots & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -2y_{i-1} & 2\frac{y_{i-1}}{y_1} & \dots & 2\frac{y_{i-1}}{y_{i-2}} & 0 & \dots & 0 & 0 \\ -2y_{i+1} & 2\frac{y_{i+1}}{y_1} & \dots & 2\frac{y_{i+1}}{y_{i-2}} & 2\frac{y_{i+1}}{y_{i-1}} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -2y_m & 2\frac{y_m}{y_1} & \dots & 2\frac{y_m}{y_{i-2}} & 2\frac{y_m}{y_{i-1}} & \dots & 2\frac{y_m}{y_{m-1}} & 0 \\ -2 & \frac{2}{y_1} & \dots & \frac{2}{y_{i-2}} & \frac{2}{y_{i-1}} & \dots & \frac{2}{y_{m-1}} & \frac{2}{y_m} \end{bmatrix} \tag{55}$$

It is easy to see that $\det(M_{i(i+1)}) = -2^m$ if $i \neq 1$ and $\det(M_{12}) = 2^m$.
Using these results we get from (51) that

$$\begin{aligned} \det(M_1) &= -2^m \left((-1)^{m+2} \frac{1}{y_1} + (-1)^{m+3} \right), \\ \det(M_i) &= -2^m \left((-1)^{m+1+i} \frac{y_{i-1}}{y_i} + (-1)^{(m+1)+(i+1)} \right), i = 2, \dots, m, \\ \det(M_{m+1}) &= -2^m y_m. \end{aligned}$$

So from (50) we obtain

$$\begin{aligned} \det(M) &= -2^m \left((-1)^{2(m+1)} \frac{1}{y_1} + (-1)^{2(m+1)+1} \right) + \\ &+ -2^m \left(\sum_{i=2}^{m} \left((-1)^{2(m+1+i)} \frac{y_{i-1}}{y_i} + (-1)^{2(m+1+i)+1} \right) + (-1)^{2(m+1)} y_m \right) \\ &= -2^m \left(\frac{1}{y_1} + \sum_{i=2}^{m} \frac{y_{i-1}}{y_i} + y_m - m \right). \end{aligned}$$

Since $y_{i-1} > y_i$ $(i = 2, \dots, m)$ and $\frac{1}{2} \geq y_1$ we get $\frac{1}{y_1} + \sum_{i=2}^{m} \frac{y_{i-1}}{y_i} > m$, and so $\det(M) < 0$. □

Lemma 3.11. *The solution of equation system (48) satisfies the conditions of the linear program (47) and*

$$b = 1 + \frac{1 - y_m}{\frac{1}{y_1} + \sum_{i=2}^{m} \frac{y_{i-1}}{y_i} + y_m - m}. \tag{56}$$

Proof. By substituting the right hand side of (48) into M we obtain the following matrix M_b:

$$M_b = \begin{bmatrix} -2y_1 & 0 & \dots & 0 & 0 & -1-y_1 \\ -2y_2 & 2\frac{y_2}{y_1} & \dots & 0 & 0 & -1-y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -2y_m & 2\frac{y_m}{y_1} & \dots & 2\frac{y_m}{y_{m-1}} & 0 & -1-y_m \\ -2 & \frac{2}{y_1} & \dots & \frac{2}{y_{m-1}} & \frac{2}{y_m} & -1 \\ -1 & 1 & \dots & 1 & 1 & 0 \end{bmatrix} \tag{57}$$

and

$$\det(M_b) = \det(M) + \sum_{i=1}^{m} y_i \left(-1\right)^{m+1+i} \det(M_i) = \det(M) - 2^m \left(1 - y_m\right) \tag{58}$$

Using again Cramer's Rule:

$$b = \frac{\det(M_b)}{\det(M)} = 1 + \frac{1 - y_m}{\frac{1}{y_1} + \sum_{i=2}^{m} \frac{y_{i-1}}{y_i} + y_m - m} \tag{59}$$

By a similar analysis we can prove that the other conditions of (47) also hold. As an example we present only the inequality

$$z = \frac{\det(M_z)}{\det(M)} = \frac{\frac{1}{2}\det(M) - 2^{m-1}\frac{y_m - 1}{y_1}}{\det(M)} = \frac{1}{2} + \frac{y_m - 1}{2y_1\left(\frac{1}{y_1} + \sum_{i=2}^{m} \frac{y_{i-1}}{y_i} + y_m - m\right)} < \frac{1}{2} \tag{60}$$

where

$$M_z = \begin{bmatrix} -1-y_1 & 0 & \dots & 0 & 0 & -1-y_1 \\ -1-y_2 & 2\frac{y_2}{y_1} & \dots & 0 & 0 & -1-y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -1-y_m & 2\frac{y_m}{y_1} & \dots & 2\frac{y_m}{y_{m-1}} & 0 & -1-y_m \\ -1 & \frac{2}{y_1} & \dots & \frac{2}{y_{m-1}} & \frac{2}{y_m} & -1 \\ 0 & 1 & \dots & 1 & 1 & 0 \end{bmatrix} \tag{61}$$

The validity of the other inequalities can be shown easily.

□

Lemma 3.12. *The solution of equation system (48) is an optimal solution of the linear program (47).*

Proof. Consider the solution vector v of (48). Because of Lemma 3.11 it is a feasible solution of (47). Let v be a basis solution. The objective function contains only b, which has a positive coefficient, satisfying the optimum criterion. So we get the statement of the lemma.

□

3.4.3 Getting the lower bound

In the following we deal with the optimal choice of the variables $y_1, y_2, \ldots, y_m$. To choose them in an optimal way, we have to solve the following nonlinear optimization problem:

$$\max\left\{1+\frac{1-y_m}{\frac{1}{y_1}+\sum_{i=2}^{m}\frac{y_{i-1}}{y_i}+y_m-m}\right\} \tag{62}$$
$$\frac{1}{2}\geq y_1>\ldots>y_m>0.$$

It does not seem to be easy to solve (62) analytically for arbitrary fixed m, but it can be done numerically by global optimization tools. Details can be found in [8]. In this paper we analyse (62) for the case when $m\to\infty$.

Lemma 3.13. *The optimal solution of (62) converges to the maximum of the function*

$$f(x):=1+\frac{1-x}{x+1+\ln\left(\frac{1}{x}\right)-\ln(2)} \tag{63}$$

in the interval $\left(0,\frac{1}{2}\right]$ *if* $m\to\infty$.

Proof. We distinguish two cases:

Case 1: In this case we assume that $y_2\leq\frac{y_1}{2}$. Then $2\leq\frac{y_1}{y_2}$. Using $y_1\leq\frac{1}{2}, y_2\leq\frac{1}{4}, y_m\leq y_2$, $4\geq 2+y_m$, and applying the inequality between the arithmetic and geometric means for the numbers $\frac{y_2}{y_3},\frac{y_3}{y_4},\ldots,\frac{y_{m-1}}{y_m},y_m$ in the denominator of (62), we obtain the upper approximation [82].

$$\begin{aligned}1+\frac{1-y_m}{\frac{1}{y_1}+\sum_{i=2}^{m}\frac{y_{i-1}}{y_i}+y_m-m} &\leq 1+\frac{1-y_m}{4+\sum_{i=3}^{m}\frac{y_{i-1}}{y_i}+y_m-m}\\ &\leq 1+\frac{1-y_m}{2+y_m-k+(m-1)\sqrt[m-1]{y_m}}\end{aligned}$$

Using that $\lim_{m\to\infty}(m-1)\sqrt[m-1]{x}-m=\ln(x)-1$ and $\ln(x)<-\frac{5}{4}$ if $x\leq\frac{1}{4}$, we obtain that

$$\lim_{m\to\infty}1+\frac{1-y_m}{\frac{1}{y_1}+\sum_{i=2}^{m}\frac{y_{i-1}}{y_i}+y_m-m}\leq 1+\frac{1-y_m}{y_m+1+\ln(y_m)}<1 \tag{64}$$

Case 2: Suppose that $y_2>\frac{y_1}{2}$. From this we get $2y_2(1-2y_1)\geq y_1(1-2y_1)$ and so

$$\frac{1}{y_1}+\frac{y_1}{y_2}\geq 2+\frac{1}{2y_2} \tag{65}$$

Using (65) and $y_1\leq\frac{1}{2}$ and then applying again the inequality between the arithmetic and geometric means for the numbers $\frac{1}{2y_2},\frac{y_2}{y_3},\ldots,\frac{y_{m-1}}{y_m}$ in the denominator of (62), we obtain the upper approximation

$$1+\frac{1-y_m}{\frac{1}{y_1}+\sum_{i=2}^{m}\frac{y_{i-1}}{y_i}+y_m-m} \leq 1+\frac{1-y_m}{2+\frac{1}{2y_2}+\sum_{i=3}^{m}\frac{y_{i-1}}{y_i}+y_m-m}$$
$$\leq 1+\frac{1-y_m}{2+y_m-m+(m-1)\sqrt[m-1]{\frac{1}{2y_m}}}.$$

Since $\lim_{m\to\infty}(m-1)\sqrt[m-1]{\frac{1}{2x}}-m=\ln\left(\frac{1}{x}\right)-\ln(2)-1$ we get that

$$\lim_{m\to\infty}1+\frac{1-y_m}{\frac{1}{y_1}+\sum_{i=2}^{m}\frac{y_{i-1}}{y_i}+y_m-m}\leq 1+\frac{1-y_m}{y_m+1+\ln\left(\frac{1}{y_m}\right)-\ln(2)} \tag{66}$$

Since $\frac{1-x}{x+1+\ln(\frac{1}{x})-\ln(2)}\geq 0$ if $x\in(0,\frac{1}{2}]$, we get that the maximum of the limit comes from Case 2, which proves the statement of the lemma.

Here we used the fact, that if we compute the maximum of $f(x)$ in the given interval, then we can construct a series of values $y_1,y_2,\ldots,y_m$ which gives an optimal solution for (62). The construction sets y_1 to $\frac{1}{2}$ and defines the geometrical series $y_i=\frac{1}{2}(2y_m)^{\frac{i-1}{m-1}}$, $i=2,\ldots,m$ which ends with y_m.

□

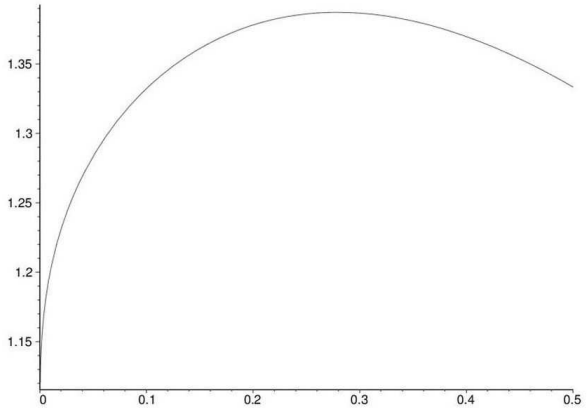

Figure 3.1. Graph of the function $f(x)$ in the interval $\left(0,\frac{1}{2}\right]$.

Lemma 3.14. (Balogh, Békési, Galambos and Reinelt, [5]). *The maximum of the function $f(x)$ is $1-\frac{1}{W_{-1}\left(\frac{-2}{e^3}\right)+1}\approx 1.3871$ in the interval $\left(0,\frac{1}{2}\right]$, where $W_{-1}(x)$ is the real branch of the Lambert W function for which $W(x)\leq -1$ holds.*

Proof. The proof is straightforward by taking the derivative of $f(x)$ and investigating the function analytically.

□

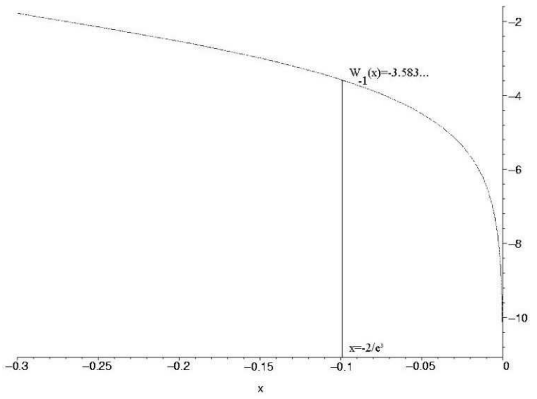

Figure 3.2. Graph of the function $W_{-1}(x)$ in the interval $[-0.3, 0)$.

3.5 Concluding remarks.

For *k-bounded* semi-online algorithms we investigated REP_3 . The asymptotic competitive ratio of REP_3 exactly matched the lower bound of Lee and Lee [77] while used only a constant number of bins. The following question was raised in our paper [51], and has remained open until now. Does repack help in 2-bounded space online bin packing? The ideas of our algorithm do not work for two active bins. For example, we might receive a sequence of items consisting of five items of size $\frac{1}{7} + \varepsilon$ and six items of size $\frac{1}{7}$, where ε is some very small appropriately chosen positive real. From this sequence we cannot select a good subset. If the next item to pack is a size greater than $\frac{2}{7}$, we must close some bin with weight smaller than one.

For *k-repacking* semi-online problem we presented a family of algorithms. With increasing values of k, the asymptotic competitive ratio of our algorithms quickly converges to 1.5 and our method gives better results than the classical online algorithms already for small k. $R^{\infty}_{HR\text{-}2} = 1.5728\ldots$ is better than the asymptotic ratio $1.58889\ldots$ of the best known online algorithm [94]. $R^{\infty}_{HFR\text{-}4} = 1.5389\ldots$ is smaller than the best known lower bound $1.5403\ldots$ for online algorithms published in [100]. These results tell us that our algorithm is capable of exploiting the additional flexibility obtained from the chance of repacking items.

Unfortunately, our algorithm does not improve the upper bound of the best known online algorithm of Seiden [94] for $k = 1$. But in this case our algorithm with a simple interval structure uses only 6 bin classes, while the very sophisticated Harmonic++ of [94] uses a complicated interval structure with 76 subclasses. Furthermore, we strongly conjecture that the upper bound in the case $k = 1$ can be improved to 1.59 by using repacking and around 15 bin classes. These are far fewer bin classes than the 76 used in [94]. This research is in progress.

4 Multidimensional bin packing problems

Now, we consider the following problem: we are given a list $L = \{a_1, a_2, \ldots, a_n\}$ of n items, which are defined with ordered pair of sizes $(w(a_i), h(a_i))$ where $w(a_i)$ and $h(a_i)$ is the width and the height of a_i, respectively. We are also given rectangular bins with sizes W and H. (W.l.g. we can suppose that $W = H = 1$, and $w(a_i) \leq 1$, $h(a_i) \leq 1$.) The aim is to pack the small rectangles into minimum number of bins so that the sides of the items are parallel to the corresponding sides of the bins (i.e no rotation allowed), and no two rectangles in a bin overlap.

By considering lists L where the width of the items is equal to one it is clear that the rectangle bin packing problem is a generalization of the one-dimensional problem and so this problem is also $\mathcal{NP}$-hard ([56]), therefore fast approximation algorithms have been studied.

In the case of off-line algorithms Chung, Garey and Johnson ([24]) introduced an unbounded space algorithm called *Hybrid-First Fit* (HFF) with $\frac{182}{90} \leq R^{\infty}_{\mathrm{HFF}} \leq \frac{17}{8}$, while Frenk and Galambos in [43] discussed a bounded-space algorithm called *Hybrid-Next Fit* (HNF) with $R^{\infty}_{\mathrm{HNF}} = 3.38$. In the last paper the parametric case was also considered.

For online algorithms Coppersmith and Raghavan presented a class of algorithms A with $R^{\infty}_{A} \leq 3.25$ ([28]). Some of these algorithms and their corresponding asymptotic competitive ratios are improved in [32]. Li and Chang [78] introduced a two-dimensional generalization of Harmonic Fit [77] with $R^{\infty}_{A} = 2.86$. Finally, the best known online and bounded-space algorithm is due to Seiden and van Stee [96] who investigate an algorithm with asymptotic competitive ratio of 2.66013.

4.1 Lower bounds for 2D online rectangle packing

In this section we show the results we have for the lower bounds of online algorithms in the two-dimensional – rectangular packing – case. In our proof we apply the *Sylvester sequences*. We will consider the non-parametric case i.e. $r = 1$, and we will use the packing pattern technique we described in Section 2.1.

For the lists satisfying $h(a_j) = \frac{1}{m_j} + \varepsilon$ and $w(a_j) = 1$ we get a trivial lower bound for the asymptotic competitive ratio of an arbitrary two-dimensional online algorithm.

Theorem 4.1. *For any online two-dimensional algorithm A $R^{\infty}_{A} \geq 1.5403\ldots$.*

4.1.1 A simple lower bound

For the two-dimensional rectangular BPP the first non-trivial lower bound was given by Galambos in [47], where the following simple construction was considered. Let $n = 4k$ and consider the lists L_1, L_2, L_3, L_4 with

- L_1 contains n pieces of A-items with sizes $\left(\frac{1}{2} - \varepsilon, \frac{1}{2} - 2\varepsilon\right)$.
- L_2 contains n pieces of B-items with sizes $\left(\frac{1}{2} + \varepsilon, \frac{1}{2} - \varepsilon\right)$.

- L_3 contains n pieces of C-items with sizes $\left(\frac{1}{2}-2\varepsilon, \frac{1}{2}+2\varepsilon\right)$.
- L_4 contains n pieces of D-items with sizes $\left(\frac{1}{2}+2\varepsilon, \frac{1}{2}+\varepsilon\right)$.

Theorem 4.2. (Galambos, [47]). *For any online two-dimensional algorithm A $R_A^\infty \geq 1.6$.*

First, we give upper bounds for the optimal packing.

Lemma 4.1.

$$\mathrm{OPT}(L_1 \dots L_j) \leq j\frac{n}{4}, \qquad j = 1, 2, 3, 4.$$

Proof. We leave to the reader to verify the cases $j = 1, 2$, and we give the construction for the case $j = 3$. One can easily see that the packing patterns $(2, 0, 2, 0)$ and $(1, 2, 1, 0)$ are feasible packing. So, if we have $\frac{n}{4}$ bins from the first pattern and $\frac{n}{2}$ from the second one, we can pack all the items from the lists L_1, L_2, and L_3. Therefore, the optimal packing may not use more than $\frac{3n}{4}$ bins.

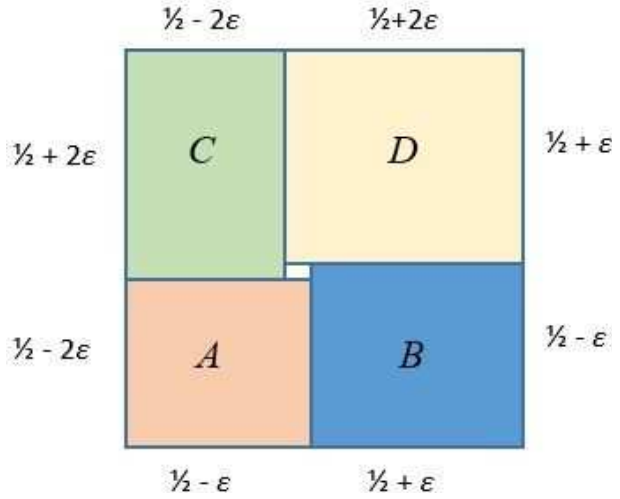

Figure 4.1. Optimal packing of $(L_1L_2L_3L_4)$

□

For a feasible packing $p = (p_1, \dots, p_4)$ let $n(p)$ denote the number of p-type bins. We examine how many bins have been used after having packed the list L_j, $1 \leq j \leq 4$. If A is an arbitrary algorithm then

$$A(L_1 \dots L_j) = \sum_{i=1}^{j} \sum_{p \in P_i} n(p) \tag{67}$$

and the number of packed items

$$n = \sum_{p \in P} p_j n(p), \qquad 1 \leq j \leq 4. \tag{68}$$

Summing all the equations (67) and substracting (68) we get

$$\begin{aligned} & A(L_1) + A(L_1L_2) + A(L_1L_2L_3) + A(L_1L_2L_3L_4) - 4n \\ & = 4\sum_{p\in P_1} n(p) + 3\sum_{p\in P_2} n(p) + 2\sum_{p\in P_3} n(p) + \sum_{p\in P_4} n(p) \\ & - \sum_{p\in P} n(p) \sum_{j=1}^{4} p_j. \end{aligned} \tag{69}$$

Lemma 4.2. *The rigth hand side of (69) is non-negative.*

Proof. Let us consider a bin that is in P_1. We have to prove that for each bin of such type $\sum_{i=1}^{4} \leq 4$. In other words we have to prove that in each bin that contains at least one item from L_1 may not contain more than 4 items, and this is trivial. Similarly, we have to prove obvious statements in the other cases.

□

Let us introduce the following notation

$$R_A^{\infty}(j) = \lim_{n\to\infty} \frac{A(L_1 \dots L_j)}{\mathrm{OPT}(L_1 \dots L_j)}, \qquad 1 \leq j \leq 4. \tag{70}$$

Now, if we replace (70) into the left hand side of (69), and we use the result of Lemma 4.1, after some calculation we get

$$\sum_{j=1}^{4} j R_A^{\infty}(j) \geq 16.$$

If $R_A^{\infty} = \max_j R_A^{\infty}(j)$ then we get the claimed result.

□

When the paper was published, the best upper bound for the one-dimensional online bin packing problem was 1.58889, by Richey [90]. So, this lower bound was the first to prove that the two-dimensional online algorithms could not be as good as the one-dimensional ones.

4.1.2 Improved lower bound

Improving the technique that had been used in [47] we gave a better lower bound (see [50]). In this section we will use the notations applied in [101] in general.

Let $k \geq 2$ and let $L = \{L_1L_2 \dots L_{2k}\}$ be a concatenation of $2k$ sublists L_j, $1 \leq j \leq 2k$. Every sublists L_j contains n items with equal sizes, which will be denoted by a_j. An item has width $w(a_j)$ and height $h(a_j)$. The sizes are as follows for every j, $1 \leq j \leq k$.

$$\begin{array}{llll} w(a_{2j-1}) & = \frac{1}{2} - (k-j+1)\varepsilon & h(a_{2j-1}) = \frac{1}{m_{k-j+1}} + \varepsilon \\ w(a_{2j}) & = \frac{1}{2} + (k-j+1)\varepsilon & h(a_{2j}) = \frac{1}{m_{k-j+1}} + \varepsilon \end{array} \tag{71}$$

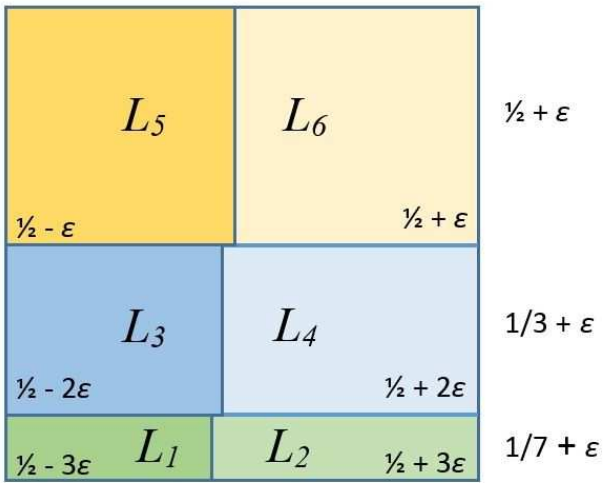

Figure 4.2. The construction for $k = 3$ (We call it W2H3)

We take $\varepsilon > 0$ small enough to satisfy $\sum_{j=1}^{k} h(a_{2j}) \leq 1$, and $w(a_1) > \frac{1}{3}$. So $\varepsilon < \min(\frac{1}{km_{k+1}-1}, \frac{1}{6k})$. The following upper bounds are true for the optimum packing.

Lemma 4.3. *If n is a multiple of $2(m_k - 1)$ then*

a) $\quad \mathrm{OPT}(L_1) \leq \frac{n}{2(m_k-1)}$

b) For every $1 \leq \mathrm{j} \leq \mathrm{k}$: $\quad \mathrm{OPT}(L_1 \dots L_{2j}) \leq \frac{n}{m_{k-j+1}-1)}$

c) For every $2 \leq j \leq k$: $\quad \mathrm{OPT}(L_1 \dots L_{2j-1}) \leq \frac{n}{2}\Big(\frac{1}{m_{k-j+1}-1} + \frac{1}{m_{k-j+2}-1}\Big)$

Proof.
Case a: We can pack 2 items of L_1 side by side and we can pack $m_k - 1$ pieces of L_1 on the top of each other. Therefore $2(m_k - 1)$ items of L_1 can be packed into one bin. So, L_1 can be packed in $\frac{n}{2(m_k-1)}$ bins.

Case b: Let us pack the a_{2i-1} and a_{2i} pair for each i, $1 \leq i \leq j$, side by side, and place m_{k-j+1} pieces of (a_{2i-1}, a_{2i}) pairs on top of each other. Then the total height filled by these items in a bin will be

$$\begin{aligned}(m_{k-j+1} - 1) \quad & \sum_{i=1}^{j}\Big(\frac{1}{m_{k-i+1}} + \varepsilon\Big) \\ & = (m_{k-j+1} - 1)\sum_{i=1}^{j}\Big(\frac{1}{m_{k-i+1} - 1} - \frac{1}{m_{k-i+2} - 1} + \varepsilon\Big) \\ & = 1 - (m_{k-j+1} - 1)\Big(\frac{1}{m_{k+1}-1} - j\varepsilon\Big) \leq 1\end{aligned} \tag{72}$$

The last inequality follows from $\varepsilon \leq \frac{1}{k(m_{k+1}-1)}$. So, $\frac{n}{(m_{k-j+1}-1)}$ bins are sufficient to pack the items from this list.

Case c: We can pack one item a_{2i-1} with one item a_{2i} side by side in a strip S_i with heigth $\frac{1}{(m_{k-i+1})} + \varepsilon$, $1 \leq i < j$. Furthermore, we can pack two items a_{2j-1} together in a strip S_0 of heigth $\frac{1}{m_{k-j+1}} + \varepsilon$. If we pack $\frac{n}{2(m_{k-j+1}-1)}$ bins with each $m_{k-j+1} - 1$ strips S_0 and $m_{k-j+1} - 1$ strips S_i, $1 \leq i < j$, we get a feasible packing with $\frac{n}{2}(\frac{1}{m_{k-j+1}-1} + \frac{1}{m_{k-j+2}-1})$ bins in total.

□

In the next lemma we give adequate weights that fulfill the conditions of the Theorem 2.3.

Lemma 4.4. *Let the weights w_j, $1 \leq j \leq 2k$ be given by $w_{2j-1} = w_{2j} = \frac{k-j+1}{m_{k-j+1}-1}$, $1 \leq j \leq k$. Then*

$$\sum_{j=i}^{2k} w_j p_j \leq 2k - i - 1$$

for every $p \in P_i$, and $1 \leq i \leq 2k$.

Proof. First we consider the case $i = 2k$. Since $w(a_{2k}) > \frac{1}{2}$ and $h(a_{2k}) > \frac{1}{2}$ it follows that that any packing pattern in P_{2k} contains only one item from L_{2k}. Since $w_{2k} = 1$, the condition is fulfilled.

So, we can assume that $i < 2k$. Let $F_i(p) = \sum_{j=i}^{2k} w_j p_j$. We will prove that

$$\max_{p \in P_i} F_i(p) \leq 2k - i + 1.$$

We will introduce the following *dominance rule:*

Definition 4.1. *m items of a_s dominate one item a_t if*

$$w(a_s) \leq w(a_t) \qquad mh(a_s) \leq h(a_t) \qquad mw_s \geq w_t$$

and at least one inequality is strict. We denote the domination by $ma_s \succ a_t$.

From the definition it follows that $a_{2j-1} \succ a_{2j}$ for all $1 \leq j \leq k$ and $(m_{k-j} - 1)a_{2j-1} \succ a_{2j+1}$ for all $1 \leq j \leq k$. We will distinguish two different cases.

Case a: $i = 2j$ for $1 \leq j \leq k-1$. Because of the dominance rule, we need to consider only items of L_{2j} and L_{2j+1} in order to maximize $F_{2j}(p)$. We know that

$$w(a_{2j}) + w(a_{2j-1}) > 1,$$

therefore we can not put elements from L_{2j} and L_{2j-1} side by side. So, any packing pattern with maximum weight consists of some strips S_j with height $h(a_{2j}) = \frac{1}{m_{k-j+1}} + \varepsilon$ which contains one item from $L :_{2j}$ in each, and some strips S_{j+1} with height $h(a_{2j+1}) = \frac{1}{m_{k-j}} + \varepsilon$ which contains two items from $L :_{2j+1}$ in each. Let us denote the number of strip S_j and S_{j+1} by q_j and q_{j+1}, resp. Then

$$\max_{p \in P_{2j}} F_{2j}(p) = \max_{p \in P_{2j}} (q_j w_{2j} + 2q_{2j+1} w_{2j+1}) = \max_{p \in P_{2j}} \left\{ q_j \frac{k-j+1}{m_{k-j+1}-1} + 2q_{j+1} \frac{k-j}{m_{k-j}-1} \right\}$$

under the condition that q_j and q_{j+1} satisfy

$$q_j\Big(\frac{1}{m_{k-j+1}}+\varepsilon\Big)+q_{j+1}\Big(\frac{1}{m_{k-j}}+\varepsilon\Big)\le 1.$$

This maximization problem is solved by $q_j = q_{j+1} = m_{k-j}-1$ and has an optimal solution value of $2(k-j)+\frac{k-j+1}{m_{k-j}}$ which is less than or equal to $2k-i+1$.

Case b: $i = 2j-1$ for $1 \le j \le k$. Because of the dominance rule, we only need to consider items of L_{2j-1}. At most two items of this list can be placed side by side , and at most $m_{k-j+1}-1$ items of L_{2j-1} can be placed on the top of each other. This means that the maximum weight in a packing pattern is equal to

$$2(m_{k-j+1}-1)w_{2j-1} = 2k-i+1$$

because w_{2j-1} is equal to $\frac{k-j+1}{m_{k-j+1}-1}$.

□

The above two lemmas lead us to the following theorem.

Theorem 4.3. (Galambos and van Vliet, [50]). *For any online algorithm for the two-dimensional bin packing problem*

$$R_A^\infty \ge \lim_{k\to\infty}\frac{4\sum_{j=1}^{k}\frac{j}{m_j-1}}{4\sum_{j=1}^{k}\frac{1}{m_j-1}-1} = 1,80288\ldots.$$

Proof. This follows directly from Lemma 4.3, Lemma 4.4, and Theorem 2.3.

□

4.2 Concluding remarks

With the same technique the three-dimensional case were investigated in [50]. We proved the following statement.

Theorem 4.4. (Galambos, van Vliet, [50]). *For any online algorithm for the* 3*-dimensional bin packing problem*

$$R_A^\infty \ge \lim_{k\to\infty}\frac{8\sum_{j=1}^{k}\frac{j}{m_j-1}}{8\sum_{j=1}^{k}\frac{1}{m_j-1}-3} = 1,974\ldots.$$

In the conclusion of the paper the authors suggested to generalize this technique for the d-dimensional case for $d > 3$. We strongly believe that the following conjecture is true.

Conjecture. For any online algorithm for the d-dimensional bin packing problem

$$R_A^\infty \ge \lim_{k\to\infty}\frac{2^d\sum_{j=1}^{k}\frac{j}{m_j-1}}{2^d\sum_{j=1}^{k}\frac{1}{m_j-1}-(2^{d-1}-1)}.$$

If we let the dimension grow to infinity, the lower bound converges to 2.181... The conjecture has not been proved yet.

Later, van Vliet in [101] introduced a new technique that based on a liner programming (LP) model. He improved the above results for the multidimensional cases. To describe an LP model he needed to fix the sizes of the problems by considering limited number of lists. For the two-dimensional case he analysed the W3H3, W3H4, and W3H5 cases. This means that – using lists which have sizes around the Sylvester sequence elements – in the W3H3 case the smallest item sizes were around $\frac{1}{7}$, and in the W3H5 case the smallest heights were around $\frac{1}{1807}$. His best result for the two dimensional case is the following lower bound.

Theorem 4.5. *For any online algorithm for the two-dimensional bin packing problem*

$$R_A^\infty \geq 1.85166\ldots.$$

Similarly, he investigated the three dimensional case as well. Applying the constructions DkW2H2, $2 \leq k \leq 5$ he got a lower bound of 2.043. Presently, these are the best lower bounds.

The improvements seem to be very plausible: in two dimension we construct an LP model with sizes W4H5 or even finer splitting. Van Vliet already warned that huge technical difficulties may arise while we generate all the feasible packing patterns. It is true, we tried it.

On the other side, having the improvements for the one-dimensional online lower bound with the new series, a good opportunity presents itself to use these series to improve the lower bound here, too. We are working on it, but there are some – rather technical – details that we need to solve. So, for the improvement the problem is still open.

5 Probabilistic analysis of bin packing algorithms

In this chapter we deal with the probabilistic analysis of different algorithms. To execute a *probabilistic analysis* we have to know the probability distribution of the elements of our instances. Let us choose the input of an instance independently from the same distribution. Knowing the distribution the expected values of the optimal solution and the approximation algorithm can be calculated. Then having the expected values we can compare them. The wider the conditions of the distribution function, the more general the estimations of the concerning theorems.

In the following subsections we consider different bin packing problems, and we investigate the probabilistic behaviour of various algorithms.

5.1 One-dimensional bin packing problem

Based on our paper [30] we first analyse the *Next Fit Decreasing* (NFD) algorithm.

Algorithm 5.1.1. *The NFD algorithm works as follows. The items on the list are first reindexed so that*

$$a_1 \geq a_2 \geq \ldots \geq a_n$$

Then they are assigned to bins in this order. A new bin is opened whenever there is not enough room left in the most recently opened bin to accommodate the current item.

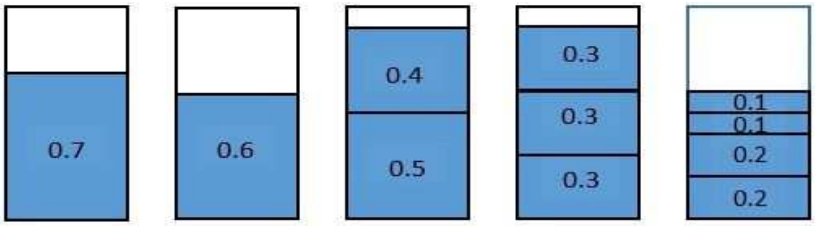

Figure 5.1. Packing items with NFD algoritm.

The maximum number of bins opened according to this rule can be shown to exceed the minimum number of possible by slightly less than 70 percent [4], $R_A^\infty = 1,691\ldots$.

In this section, however, we are interested in a probabilistic analysis of this algorithm, carried out under the assumption that $a_1, \ldots, a_n$ are drawn independently from a uniform distribution on (0, 1]. Contrary to the notations we used before, for a list contains n items, we denote the number of bin used by the algorithm A and OPT by $A(n)$ and $\mathrm{OPT}(n)$, respectivelly.

It is known that the optimal solution value $\mathrm{OPT}(n)$ satisfies

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{OPT}(n))}{n/2} = 1.$$

5.1.1 Expected solution value

To analyse the expected solution value E(NFD(n)) we approximate the performance of the NFD algorithm by that of the *Sliced NFD algorithm with parameter r.*

Algorithm 5.1.2. *The algorithm* (SNFD$_r$), *the items larger than* $1/r$ *are packed according to the* NFD *algorithm, the last opened bin is completed to contain at most* $r-1$ *items and any remaining items are packed in groups of size* r.

Obviously, for any realization of the item sizes,

$$\mathrm{SNFD}_r(n) \geq \mathrm{NFD}(n) \qquad r \geq 2, \quad n \geq 1, \tag{73}$$

and

$$\lim_{r\to\infty} \mathrm{SNFD}_r(n) = \mathrm{NFD}(n) \qquad n \geq 1. \tag{74}$$

Let k_i be the number of items whose size falls in the interval $(\frac{1}{i+1}, \frac{1}{i}]$ $(i = 1, 2, \ldots)$, and let $K_i = k_i + k_{i+1} + \ldots$. Then, clearly,

$$\mathrm{SNFD}_r(n) \leq k_1 + \frac{k_2}{2} + \ldots + \frac{k_{r-1}}{r-1} + \frac{K_r}{r} + r, \tag{75}$$

where the last term is induced to allow for rounding errors. Since $\mathrm{E}(k_i) = n/(i(i+1))$ and $\mathrm{E}(K_r) = n/r$, the expected value of the right hand side of (75) is equal to

$$n\sum_{i=1}^{r-1} \frac{1}{i^2(i+1)} + \frac{n}{r^2} + r = n\sum_{i=1}^{r-1}\left(\frac{1}{i^2} - \frac{1}{i} + \frac{1}{i+1}\right) + \frac{n}{r^2} + r, \tag{76}$$

and hence, choosing r appropriately as a function of n,

$$\limsup_{n\to\infty} \frac{\mathrm{E}(\mathrm{NFD}(n))}{n/2} \leq 2\left(\frac{\pi^2}{6} - 1\right). \tag{77}$$

On the other hand, if the items are packed by the NFD rule and bins containing items from more than one interval $(\frac{1}{i+1}, \frac{1}{i}]$ as well as bins containing items smaller than $\frac{1}{r}$ are ignored, then we have that

$$\mathrm{NFD}(n) \geq (k_1 - 1) + \left(\frac{k_2}{2} - 1\right) + \ldots + \left(\frac{k_{r-1}}{r-1} - 1\right), \tag{78}$$

so that, for any fixed r,

$$\liminf_{n\to\infty} \frac{\mathrm{E}(\mathrm{NFD}(n))}{n/2} \geq 2\sum_{i=1}^{r-1} \frac{1}{i^2} - 2 + \frac{2}{r}. \tag{79}$$

The rigth hand side of (79) is monotonically increasing in r and converges to $2(\pi^2/6 - 1)$. Hence, in combination of (77) we conclude the following theorem.

Theorem 5.1. (Csirik, Frenk, Frieze, Galambos, Rinnooy Kan, [30]). *Let us suppose that the elements of the lists are drawn independently from a uniform distribution on* $(0,1]$, *and we pack the items according to the* NFD *rule. Then*

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NFD}(n))}{n/2} = 2\left(\frac{\pi^2}{6} - 1\right) = 1.289\ldots. \tag{80}$$

5.1.2 Deviation from the expected value

Now, we will study the deviation probability,

$$\Pr\Big\{ \mid \mathrm{NFD}(n) - \mathrm{E}\Big(\mathrm{NFD}(n)\Big) \mid \geq nt \Big\}. \tag{81}$$

The above probability is bounded from above by

$$\Pr\Big\{ \mathrm{NFD}(n) - \mathrm{E}\Big(\mathrm{NFD}(n)\Big) \geq nt \Big\} + \Pr\Big\{ \mathrm{NFD}(n) - \mathrm{E}\Big(\mathrm{NFD}(n)\Big) \leq -nt \Big\}. \tag{82}$$

Using the inequality (75), the first probabilty in (82) is bounded by

$$\Pr\Big\{ \sum_{i=1}^{r-1} \frac{k_i}{i} + \frac{K_r}{r} + r - \mathrm{E}\Big(\mathrm{NFD}(n)\Big) \geq nt \Big\}, \tag{83}$$

Similarly, using the inequality (78), the second probabilty in (82) is bounded by

$$\Pr\Big\{ \sum_{i=1}^{r-1} \frac{k_i}{i} - (r-1) - \mathrm{E}\Big(\mathrm{NFD}(n)\Big) \leq -nt \Big\}. \tag{84}$$

Now, we remind the reader to the Laplace-Stieltjes transform. If $\lambda_i \geq 0$, $i = 1, 2, \ldots, r$ the

$$\mathrm{E}\Big(\exp \sum_{i=1}^{r-1} \lambda_i k_i + \lambda_r K_r\Big) = \left(\sum_{i=1}^{r-1} \frac{e^{\lambda_i}}{i(i+1)} + \frac{e^{\lambda_r}}{r} \right)^n.$$

Therefore, for every $\lambda > 0$,

$$\mathrm{E}\left(\exp\Big(\lambda(\sum_{i=1}^{r-1} \frac{k_i}{i} + \frac{K_r}{r}) \Big) \right) = \left(\sum_{i=1}^{r-1} \frac{e^{\frac{\lambda}{i}}}{i(i+1)} + \frac{e^{\frac{\lambda}{r}}}{r} \right)^n.$$

Using this Laplace-Stieltjes transform, we can verify that, for every r,

$$\sum_{i=1}^{r-1} \frac{k_i}{i} + \frac{K_r}{r}$$

is distributed as $\sum_{j=1}^{n} y_{jr}$, where the y_{jr} are independently identical distributed (i.i.d) random variables with $\Pr\{y_{jr} = 1/i\} = 1/i(i+1)$, $(i = 1, 2, \ldots, r-1)$, and $\Pr\{y_{jr} = 1/r\} = 1/r$. Therefore (83) can be rewritten as

$$\Pr\Big\{\sum_{i=1}^{n} y_{jr} - \mathrm{E}\Big(\mathrm{NFD}(n)\Big) \geq nt - r\Big\}, \tag{85}$$

which – in combination with (78) – easily shown to be bounded by

$$\Pr\left\{\sum_{i=1}^{n}\left(y_{jr} - \mathrm{E}(y_{jr})\right) \geq n\Big(t - \frac{1}{r^2}\Big) - 2r + 1\right\}. \tag{86}$$

Similarly, it is also true that for every r, $\sum_{i=1}^{r-1}(k_i/i)$ is distributed as $\sum_{j=1}^{n} z_{jr}$, where the z_{jr} are i.i.d. random variables with $\Pr\{z_{jr} = 1/i = 1/i(i+1)\}$, $(i = 1, 2, \ldots, r-1)$, and $\Pr\{z_{jr} = 0\} = 1/r$. Therefore (84) can also be bounded by

$$\Pr\left\{\sum_{i=1}^{n}\left(z_{jr} - \mathrm{E}(z_{jr})\right) \leq -n\Big(t - \frac{1}{r^2}\Big) + 2r - 1\right\}. \tag{87}$$

Now, since both, y_{jr} and z_{jr} are bounded by 1, using the Hoeffding's inequality which provides an upper bound on the probability that the sum of random variables deviates from its expected value (see [66]), we get that (86) and (87) are bounded by $\exp(-2n(t - 1/r^2 - 2r/n)^2)$. Taking $r = [n^{1/3}]$, we obtain that for all t,

$$\Pr\left\{\left|\mathrm{NFD}(n) - \mathrm{E}\Big(\mathrm{NFD}(n)\Big)\right| \geq nt\right\} \leq 2\exp\left(-2n\Big(t - \frac{3}{n^{2/3}}\Big)\right).$$

5.1.3 A central limit theorem

Before we state our main theorem of this section we make the following calculations. Let ζ denote the Riemann zeta function (i.e. $\zeta(s) = \sum_{i=1}^{\infty} \frac{1}{i^s}$).

$$\begin{aligned}
\sigma_\infty^2 &= \lim_{n\to\infty} \mathrm{var}(z_{jr}) = \lim_{n\to\infty} \sum_{i=1}^{r-1} \frac{1}{i^3(i+1)} - \left(\sum_{i=1}^{r-1} \frac{1}{i^2(i+1)}\right)^2 \\
&= \lim_{n\to\infty} \sum_{i=1}^{r-1}\left(\frac{1}{i^3} - \frac{1}{i^2(i+1)}\right) - \left(\sum_{i=1}^{r-1} \frac{1}{i^2(i+1)}\right)^2 \\
&= \zeta(3) - \Big(\frac{\pi^2}{6} - 1\Big) - \Big(\frac{\pi^2}{6} - 1\Big)^2 = \zeta(3) + \frac{\pi^2}{6} - \frac{\pi^4}{36} = 0,14118\ldots
\end{aligned} \tag{88}$$

Theorem 5.2. (Csirik, Frenk, Frieze, Galambos, Rinnooy Kan, [30]). *For every x*

$$\lim_{n\to\infty} \Pr\left\{\frac{\mathrm{NFD}(n) - n(\pi^2/6 - 1)}{\sqrt{n}\sigma_\infty} \leq x\right\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2/2} dy.$$

Proof. To prove this central limit theorem we first observe that

$$\begin{aligned} & \Pr\left\{ \frac{\mathrm{NFD}(n) - n(\pi^2/6 - 1)}{\sqrt{n}\sigma_\infty} \le x \right\} \\ \le\ & \Pr\left\{ \frac{\sum_{i=1}^{r-1}(k_i/i) - (r-1) - n(\pi^2/6-1)}{\sqrt{n}\sigma_\infty} \le x \right\} \qquad (89) \\ =\ & \Pr\left\{ \frac{\sum_{j=1}^{n}\Big(z_{jr} - \mathrm{E}(z_{jr})\Big)}{\sqrt{n\mathrm{var}(z_{jr})}} \le \frac{x\sigma_\infty}{\sqrt{\mathrm{var}(z_{jr})}} + \frac{r-1}{\sqrt{n\mathrm{var}(z_{jr})}} + \frac{\sqrt{n}\sum_{i=1}^{\infty}\frac{1}{i^2(i+1)}}{\sqrt{\mathrm{var}(z_{jr})}} \right\}. \end{aligned}$$

Taking $r = [n^{1/3}]$ we get that the rigth hand side of the final inequality converges to x as $n \to \infty$. But then we can apply Theorem 7.1.2 in Chung [23] to conclude that (89) converges to $1/\sqrt{1\pi}\int_{-\infty}^{x}\exp -y^2/2dy$ as $n \to \infty$. The random variables y_{jr} provide a lower bound on $\Pr\{(\mathrm{NFD}(n) - n(\pi^2/6-1))/\sqrt{n}\sigma_\infty \le x\}$ in an exactly similar fashion. Together, the lower and upper bound yield the desired result.

□

The method we used in this section has some consequences:

- The above result can be used to compute $\lim_{n\Rightarrow\infty}\mathrm{E}\Big(\big(\mathrm{NFD(n)}\big)^k\Big)$for any $k > 0$.
- We also observe that the results in this section can be extended to a larger class of distribution function F than the uniform one. More exactly, the results are valid, if the item sizes are generated from any distribution whose density function f satisfies $\lim_{x\downarrow 0} f(x) = c > 0$. One essentially needs to redifine the random variables y_{jr} and z_{jr} by letting $\Pr\{y_{jr} = 1/i\} = \Pr\{z_{jr} = 1/i\} = F(1/i) - F(1/(i+1))$, $i = 1, 2, \ldots, r-1$, and $\Pr\{y_{jr} = 1/r\} = \Pr\{z_{jr} = 0\} = F(1/r)$, and using the information on f to bound the latter right hand side.
- As a final note we observe that our results are also valid for the *Harmonic* algorithms introduced in Lee and Lee [77]. They can be easily adapted to show that the *Revised Harmonic* algorithm satisfies

$$\lim_{n\to\infty}\frac{\mathrm{E}\Big(\mathrm{RH(n)}\Big)}{n/2} = 1.237\ldots.$$

  which is only slightly better than NFD, but it is still surprisingly poor for the algorithm that from a worst case point of view is one of the best online algorithms.

5.2 2D rectangle packing problem

Let us consider the two-dimensional rectangle packing problem: We are given a list L of rectangles, the size of a rectangle $p \in L$ is given with an ordered pair of width and height

$(w(p), h(p))$, and we are given rectangular bins with sizes W and H. W.l.o.g. we can suppose that $W = H = 1$. We have to pack the rectangles into minimal number of bins so that the sides of the rectangles are parallel the corresponding sides of the bins, no ration is allowed, and no two rectangles in a bin overlap.

In [43] we presented the algorithm *Hybrid Next Fit* (HNF). Table 5.1 shows how the HNF algorithm works.

Beside the worst case analysis we also investigated the probabilistic behaviour of HNF. Similarly to the previous section, in order to analyse the expected number of bins used by the HNF algorithm we approximate its performance by that of the *Sliced HNF* with parameter r, HNF_r. Consider now a sequence of positive random vectors $(w_i(p), h_i(p))_{i=1}^{\infty}$, bounded by 1 in each component, with $(w_i(p))_{i=1}^{n}$, $(h_i(p))_{i=1}^{n}$ independent subsequences consisting of independent and identically distributed random variables.

(1) Order the rectangles p of the list in nonincreasing direction according to their heights $h(p)$.

(2) Take the first item, say p, from the ordered list, and place it in the first bin into the lower left hand corner. We call the rectangular area of height $h(p)$ of the bin that is covered by p *the block opened by* p.

(3) Take the next rectangle of L and try to place it into the last opened block within the current bin if it is possible. If this is impossible, open a new block within the current bin if it is possible. If there is no space for the new block in the current bin, open a new bin with a new first block.

(4) If we have items unplaced then goto Step 3, else stop.

Table 5.1. The steps of the HNF algorithm

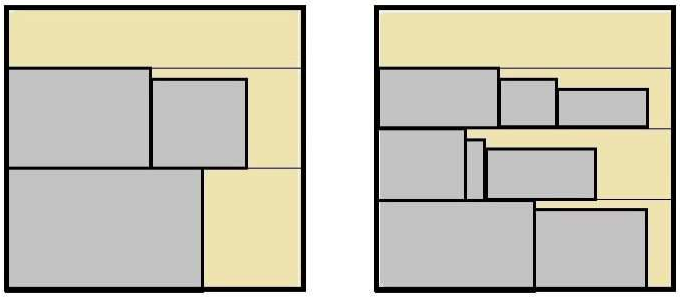

Figure 5.2. Packing rectangles with HNF algoritm.

We denote the number of vectors among the first n whose second component belongs to $(1/(i+1), 1/i]$, $1 \le i \le r-1$ by $k_i(n)$, and $K_i(n) = k_i(n) + k_{i+1}(n) + \ldots$ then one can

easily verify that

$$\mathrm{SHNF}_r(n) \le \sum_{i=1}^{r-1} \frac{\mathrm{NF}(k_i(n))}{i} + \frac{\mathrm{NF}(K_r(n))}{r} + r. \tag{90}$$

On the other hand, if the items are packed by the HNF rule and bins containing items, whose second components are smaller than $1/r$ or belong to different intervals $(1/(i+1), 1/i]$, $1 \le i \le r-1$, are ignored, then we have that

$$\mathrm{HNF}(n) \ge \sum_{i=1}^{r-1} \frac{\mathrm{NF}(k_i(n))}{i} - r. \tag{91}$$

From the last two inequalities we get for every $r \ge 2$ that

$$\sum_{i=1}^{r-1} \frac{\mathrm{E}(\mathrm{NF}(k_i(n)))}{i} - r \;\le\; \mathrm{E}(\mathrm{HNF}(n)) \tag{92}$$

$$\le \sum_{i=1}^{r-1} \frac{\mathrm{E}(\mathrm{NF}(k_i(n)))}{i} + \frac{\mathrm{E}(\mathrm{NF}(K_r(n)))}{r} + r. \tag{93}$$

Notice that for $n, m \ge 0$

$$0 \le \mathrm{NF}(n+m) \le \mathrm{NF}(n) + \mathrm{NF}(m)$$

and so, by the theory of subadditive functions (see Kingman [76])

$$\lim_{n\to\infty} \frac{\mathrm{NF}(n)}{n} = c$$

exist a.e. Moreover, since $\mathrm{NF}(n) \le n$, we get by the dominated convergence theorem

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NF}(n))}{n} = c. \tag{94}$$

Using the observations (93) and (94) and the fact that $k_i(n)$, $1 \le i \le r-1$, resp. $K_r(n)$, are binomially distributed with parameters n, $F(1/i) - F(1/(i+1)$, resp. $F(1/r)$, where F denotes the probability distribution of the height $h(p)$, we obtain by a standard argument

$$\limsup_{n\to\infty} \frac{\mathrm{E}(\mathrm{HNF}(n))}{n} \le c \sum_{i=1}^{r-1} \frac{F\left(\frac{1}{i}\right) - F\left(\frac{1}{i+1}\right)}{i} + \frac{c}{r} F\left(\frac{1}{r}\right) \tag{95}$$

and

$$\liminf_{n\to\infty} \frac{\mathrm{E}(\mathrm{HNF}(n))}{n} \ge c \sum_{i=1}^{r-1} \frac{F\left(\frac{1}{i}\right) - F\left(\frac{1}{i+1}\right)}{i} \tag{96}$$

for every $r \geq 2$. Letting $r \to \infty$, this implies that

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{HNF}(n))}{n} = c \sum_{i=1}^{r-1} \frac{F\left(\frac{1}{i}\right) - F\left(\frac{1}{i+1}\right)}{i} = c \lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NFD}(n))}{n} \tag{97}$$

where we get the last equation from Csirik, Frenk, Frieze, Galambos and Rinnoy Kan [30]. Hence we have proved the following result.

Theorem 5.3. (Galambos and Frenk, [48]).

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{HNF}(n))}{n} = \lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NF}(n))}{n} \lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NFD}(n))}{n}. \tag{98}$$

Proof. If the item sizes $(w_i(p), h_i(p))_{i=1}^{\infty}$ are independent and uniformly distributed in the square $[0,1] \times [0,1]$ then using the results of [84] and [30]

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NF}(n))}{n} = \frac{2}{3} \qquad \text{and} \qquad \lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NFD}(n))}{n} = \left(\frac{\pi^2}{6} - 1\right)$$

we get

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{HNF}(n))}{n} = \frac{2}{3}\left(\frac{\pi^2}{6} - 1\right)$$

and since

$$\lim_{n\to\infty} \mathrm{E}(\mathrm{OPT}(n)) = \frac{n}{4},$$

this implies

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{HNF}(n))}{\mathrm{E}(\mathrm{OPT}(n))} = \frac{8}{3}\left(\frac{\pi^2}{6} - 1\right) = 1.7153\ldots.$$

□

5.3 Dual bin packing problems

There is given n items of size $a_1, \ldots, a_n$ ($a_i \in (0,1), \quad i = 1, \ldots, n$). The *dual bin packing problem* is to assign the items to the largest number of bins, subject to the total size of the items assigned to any bin is at least equal to 1. The problem could also be appropriately called the *bin covering problem* and was first studied by Assmann, Johnson, Kleitman, and Leung [3]. Their main concern was worst-case analysis of approximation algorithms. In this section we carry out an exploration of the dual bin packing problem. We do so from *probabilistic* point of view , i.e. we shall assume that the item sizes $a_1, a_2, \ldots$ are drawn independently from the uniform distribution on $[0,1]$. In this section we lean on the results published in [33] and [36].

5.3.1 Expected value of optimal solution

To investigate the probabilistic behaviour of different algorithms we need an upper bound for the expected value of the optimal solution for the case of n items that are drawn independently from uniform distribution on $(0, 1]$. The $n/2$ is an easy upper bound for the expected value of the optimum. In the next theorem we give a better upper bound.

Theorem 5.4. (Csirik, Frenk, Galambos, Rinnooy Kan, [36]). *Let us suppose that the elements of the lists are drawn independently from a uniform distribution on* $(0, 1]$. *Then*

$$\mathrm{E}(\mathrm{OPT}(n)) \leq \frac{n}{2} - \sqrt{\frac{n}{32\pi}}.$$

i.e. if n *large enough,* $\mathrm{E}(\mathrm{OPT}(n)) = n/2 - \Omega(n^{1/2})$.

Proof. In deriving an upper bound on the optimal solution value to the dual bin packing problem $\mathrm{OPT}(n)$, we shall find it convenient to assume that n is even or, equivalently, to focus on $\mathrm{OPT}(2n)$.

To obtain an upper bound on the expected value of this random variable, we start by defining b_{2n} to be the number of *big* items (i.e. those with size greater than or equal to $\frac{1}{2}$). Since, with probability 1, each bin must contain at least two items in any feasible solution, we almost always have that $\mathrm{OPT}(2n) \leq n$. If, however, we know that $b_{2n} < n$, then the best that we hope for is to pair each big item with a small item to cover a bin, and to divide the remaining small items in groups of three, each covering an additional bin. Hence, in this case

$$\mathrm{OPT}(2n) \leq b_{2n} + \frac{2n - 2b_{2n}}{3} = \frac{2n}{3} + \frac{b_{2n}}{3}.$$

Since obviously $\mathrm{OPT}(2n) \leq \sum_{i=1}^{2n} a_i$, we have that

$$\begin{aligned} \mathrm{E}(\mathrm{OPT}(2n)) \quad \leq \quad & \sum_{k=0}^{n-1} \mathrm{E}\left(\min\left(\sum_{i=1}^{2n} a_i, \frac{2n}{3} + \frac{k}{3} \right) \mid b_{2n} = k \right) \binom{2n}{k} 2^{-2n} \\ + \quad & \sum_{k=n}^{2n} E\left(\min\left(\sum_{i=1}^{n} a_i, n \right) \mid b_{2n} = k \right) \binom{2n}{k} 2^{-2n} \end{aligned} \tag{99}$$

The second term in (99) is bounded from above by

$$n \sum_{k=n}^{2n} \binom{2n}{k} 2^{-2n}, \tag{100}$$

and since (cf. [88], p.34)

$$2 \sum_{k=n}^{2n} \binom{2n}{k} = \sum_{k=0}^{2n} \binom{2n}{k} + \binom{2n}{n} = 2^{2n} + \binom{2n}{n},$$

so, (100) is equal to

$$n2^{-2n-1}\left(2^{2n}+\binom{2n}{n}\right)=\frac{n}{2}+n2^{-2n-1}\binom{2n}{n}. \tag{101}$$

If we define

$$d_i=\begin{cases} a_i & ,\text{if } (0\le a_i<\frac{1}{2}) \\ 1-a_i & ,\text{if } (\frac{1}{2}\le a_i\le 1)\end{cases} \tag{102}$$

we may observe by the exchangeability of $(a_1,\ldots,a_n)$ and the independence of b_{2n} and $\{d_i\}_{i=1}^{2n}$ (see [42]) that for every k,

$$\begin{aligned} & \mathrm{E}\left(\min\left(\sum_{i=1}^{2n}a_i,\frac{2n}{3}+\frac{k}{3}\right)\;\middle|\; b_{2n}=k\right) \\ = \;& \mathrm{E}\left(\min\left(\sum_{i=1}^{k}(1-d_i)+\sum_{i=k+1}^{2n}d_i,\;\frac{2n}{3}+\frac{k}{3}\right)\right). \end{aligned} \tag{103}$$

Hence, the first term in (99) equals

$$\sum_{k=0}^{n-1}k\binom{2n}{k}2^{-2n}+\sum_{k=0}^{n-1}\mathrm{E}\left(\min\left(\sum_{i=k+1}^{2n}d_i-\sum_{i=1}^{k}d_i,\;\frac{2}{3}(n-k)\right)\right)\binom{2n}{k}2^{-2n}. \tag{104}$$

Using the identity of (cf. [88]), p.34), we get that the first term of (104) is equal to

$$\frac{n}{2}-2n2^{-2n-1}\binom{2n}{n}.$$

We bound the minimum in the second term of (104) by $(2n-2)\mathrm{E}(d_i)=\frac{n-k}{2}$, and using the above identity of (cf. [88]) again, we obtain

$$\begin{aligned} \frac{1}{2}\sum_{k=0}^{n-1}(n-k)\binom{2n}{k}2^{-2n} &= \frac{1}{2}n\sum_{k=0}^{n}\binom{2n}{k}2^{-2n}-\frac{1}{2}\sum_{k=0}^{n}k\binom{2n}{k}2^{-2n} \\ &= \frac{1}{2}n\left(\frac{1}{2}+2^{-2n-1}\binom{2n}{n}\right)-\frac{n}{4} \\ &= \frac{1}{2}n2^{-2n-1}\binom{2n}{n}. \end{aligned} \tag{105}$$

Summing up the various components in (100), (104), and (105), we conclude from (99) that

$$\mathrm{E}(\mathrm{OPT}(2n))\le n-\frac{1}{2}n2^{-2n-1}\binom{2n}{n}. \tag{106}$$

Since for large n, $\binom{2n}{n}$ is asymptotic to $(\pi n)^{-1/2}2^{2n}$, we get the desired result

$$\limsup_{n\to\infty} \frac{\mathrm{E}(\mathrm{OPT}(2n)) - n}{(2n)^{1/2}} \leq -(32\pi)^{-1/2}. \tag{107}$$

Now, if we substitute $2n$ by n then we get the desired result.

□

As a "byproduct" of our proof we have two consequences.

- In the above proof it is necessary to require only that the distribution function $F(x)$ of the item sizes a_i satisfies $F(x) = 1 - F(1-x)$ for every $0 \leq x \leq 1/2$, and F is not degenerate at $1/2$. This means that the inequality (107) with $(32\pi)^{-1/2}$ replaced by some other constant depending on $\mathrm{E}(d_i)$ also holds for this larger class of distribution.

- For the optimal solution value to the classical bin packing problem, the above technique yields an asymptotic lower bound equal to $n/2 + (32\pi)^{-1/2}n^{1/2}$ which is slight improvement over the result in [81].

5.3.2 Pairing Heuristic

In this section we demonstrate that the upper bound in (107) is sharp by showing that certain algorithm for the dual bin packing problem produces a solution value that is equal to $n/2 - O(n^{1/2})$ in expectation. For this purpose, we adapt the binary pairing algorithm for the classical packing problem to obtain a pairing algorithm (PA) for dual bin packing.

Algorithm 5.3.1. *In the pairing algorithm, the largest unassigned item is selected and combined with the smallest unassigned item so that together they cover a bin. If no such item exists, all remaining items are added to the most recently opened bin, and the algorithm terminates.*

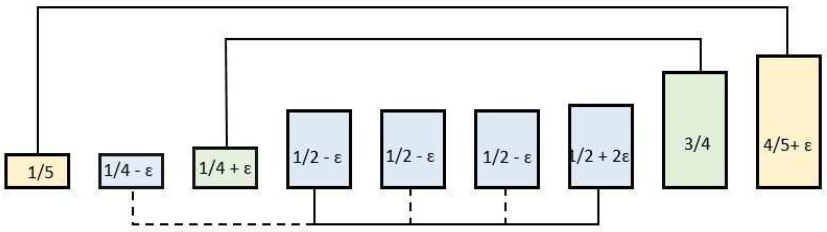

Figure 5.3. Packing items with the Pairing algorithm.

We analyse this algorithm along the line of [74], using the random variables d_i defined in (102). If we label d_i by "+1" and call it *big* if $a_i \geq \frac{1}{2}$, and label it by "-1" and call it *small* if $a_i < \frac{1}{2}$, and consider the labeled sequence d_i in $[0, \frac{1}{2}]$ in increasing order, then the PA algorithm amounts to matching each successive "+1" to the unassigned "-1" that is closest to is right. If there are no unassigned -1's to its right, match it the rightmost $+1$.

If such a $+1$ also does not exist, then put it in the bin most recently opened. If u_n is the number of unmatched small d_i, then one can verify that

$$\mathrm{PA}(n) \geq \frac{n-u_n}{2} - \frac{1}{2}$$

and hence

$$\mathrm{E}(\mathrm{PA}(n)) \geq \frac{n}{2} - \frac{1}{2}\mathrm{E}(u_n) - \frac{1}{2}.$$

To compute $\mathrm{E}(u_n)$, we first observe that the sequence of $+1$'s and -1's can be viewed as a realization of a Bernoulli process, defined by a sequence e_j $(j = 1, 2, \ldots)$ consisting of i.i.d. random variables with $\Pr\{e_j = +1\} = \Pr\{e_j = -1\} = \frac{1}{2}$.

Now, we introduce the following random variables.

$$s_k = \sum_{j=1}^{k} e_j, \qquad\qquad u_n = \max_{0\leq k\leq n}\{-s_k\},$$

where we define $s_0 = 0$. We remark that the random variables s_k and $-s_k$ have the same distribution, so $-s_k$ suffices to compute the expectation of $\max_{0\leq k\leq n}\{s_k\}$.

According to the theory of fluctuation (see e.g. [23], p. 287) we know that (assuming n is even)

$$\mathrm{E}(u_n) = \sum_{k=1}^{n}\frac{1}{k}\mathrm{E}(s_k^+) == \sum_{k=1}^{n/2}\frac{1}{2k}\mathrm{E}(s_{2k}^+) + \sum_{k=1}^{n/2}\frac{1}{2k-1}\mathrm{E}(s_{2k-1}^+),$$

where generally $x^+ = \max(x, 0)$.

Now, using the identity ([88], p.34)

$$\sum_{p=1}^{k} p\binom{2k}{k-p} = \frac{1}{2}k\binom{2k}{k}$$

we see that

$$\begin{aligned}\mathrm{E}(s_{2k}^+) &= 2\sum_{p=1}^{k}\Pr\{s_{2k}=2p\}p = 2\sum_{p=1}^{k} p\binom{2k}{k-p}2^{-2k} \\ &= 2^{-2k+1}\sum_{p=1}^{k}\binom{2k}{k-p}p = 2^{-2k}k\binom{2k}{k} \qquad (108)\end{aligned}$$

Similarly, using the identity ([88], p.34)

$$\sum_{p=1}^{k}(p-1)\binom{2k-1}{k-p} = 2^{2k-3} - 2^{2k-2} + \frac{1}{2}(2k-1)\binom{2k-2}{k-1}$$

we obtain

$$\begin{aligned}
\mathrm{E}(s^+_{2k-1}) &= 2^{-2k+1}\sum_{p=1}^{k}(2p-1)\binom{2k-1}{k-p} \\
&= 2^{-2k+2}\sum_{p=1}^{k}(p-1)\binom{2k-1}{k-p} + 2^{-2k+1}\sum_{p=1}^{k}\binom{2k-1}{k-p} \\
&= 2^{-2k+2}\left(2^{2k-3} - 2^{2k-2} + \frac{1}{2}(2k-1)\binom{2k-2}{k-1}\right) + \frac{1}{2} \\
&= 2^{-2k+1}(2k-1)\binom{2k-2}{k-1}.
\end{aligned}$$

Now, taking into account that ([40], p. 63)

$$2^{-2k}\binom{2k}{k} = (-1)^k\binom{1/2}{k}$$

with $\binom{-1/2}{k}$ defined as $(-\frac{1}{2})(-\frac{1}{2}-1)\ldots(-\frac{1}{2}-k+1)/k!$, so that

$$\sum_{k=1}^{n/2}\frac{1}{2k}E(s^+_{2k}) = \frac{1}{2}\sum_{k=0}^{n/2}(-1)^k\binom{-1/2}{k} - \frac{1}{2} = \frac{1}{2}(-1)^{n/2}\binom{-3/2}{n/2} - \frac{1}{2}. \tag{109}$$

A similar manipulation with respect to $\sum_{k=1}^{n/2}(\mathrm{E}(s^+_{2k-1}/(2k-1))$ yields as a final exact result:

$$\mathrm{E}(u_n) = \frac{1}{2}(-1)^{n/2}\binom{-3/2}{n/2} + \frac{1}{2}(-1)^{n/2-1}\binom{-3/2}{n/2-1} - \frac{1}{2}. \tag{110}$$

Using the identities

$$\binom{3/2}{n/2} = -(n+2)\binom{-1/2}{n/2+1} \qquad \text{and} \qquad \binom{3/2}{n/2-1} = (-\frac{n}{n+1})\binom{-3/2}{n/2}$$

the expression (110) breaks down to

$$\begin{aligned}
\mathrm{E}(u_n) &= \frac{(n+2)(2n+1)}{2n+2}(-1)^{n/2+1}\binom{-1/2}{n/2+1} - \frac{1}{2} \\
&= \frac{(n+2)(2n+1)}{2n+2}\binom{n+2}{n/2+1}2^{-(n+2)} - \frac{1}{2}.
\end{aligned} \tag{111}$$

A refinement of Stirling's formula (see [40]) then produces as an approximation that

$$\mid \mathrm{E}(u_n) - (2n/\pi)^{1/2} \mid \leq \alpha$$

for some constant α. Hence,

$$\mathrm{E}(\mathrm{PA}(n)) \geq \frac{n}{2} - \mathrm{E}(\frac{u_n}{2} - \frac{1}{2} \geq \frac{n}{2} - \left(\frac{n}{2\pi}\right)^{1/2} - \alpha, \tag{112}$$

as was to be proved in [3].

We have two remarks as consequences of the above proof.

- It is easy to see that the above analysis remains valid under much more general conditions than imposed here. Rather than considering independency, all that turns out to be needed is exchangeability and a symmetry condition on the joint distribution of the items.

- For the classical bin packing problem, the expected solution value of the binary pairing algorithm is given by $n/2+\mathrm{E}(u_n/2)$. Thus (111) provides an exact expression for this value, improving on the asymptotic estimates that appeared in the literature earlier.

5.3.3 Next Fit algorithm

A simple – and natural - solution method for the dual bin packing problem is given by an adaptation of the well-known next fit algorithm for classical bin packing. We continue to assume that the item sizes are uniformly distributed on $[0, 1]$.

Algorithm 5.3.2. *In the next fit algorithm (NF) for dual bin packing, one opens a bin and assigns items in arbitrary order until the sum of their sizes exceeds* 1 *and the bin is covered. The process then repeats itself.*

This algorithm is also discussed in [3]. They considered the asymptotic competitive ratio of the NF and NFD algorithms, and proved that

$$R^{\infty}_{\mathrm{NF}} = R^{\infty}_{\mathrm{NFD}} = \frac{1}{2}.$$

From probabilistic point of view the first analysis was given by Csirik and Galambos in [31]. They analysed the NF algorithm and proved the following theorem.

Theorem 5.5. (Csirik and Galambos, [31]). *Let the items of* $L = (a_1 \ldots, a_n)$ *be i.i.d. random variables with distribution*

$$a_i = \begin{cases} b & \text{,with probabilty 1/2} \\ 1-b & \text{,with probabilty 1/2,} \end{cases}$$

where $0 < b < 1/2$. *Let* $l_1 = \lceil 1/b \rceil$. *Then*

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NF})(n)}{n/2} = \frac{2^{l_1}}{2^{l_1} + 2^{l_1-2} - 1}.$$

In this section we examine the NF algorithm with more general conditions. It is clear that the number of items assigned to the first bin is equal to

$$v_1 = \inf\Big\{k \geq 1 \mid \sum_{i=1}^{k} a_i \geq 1\Big\}$$

The NF algorithms is such that the same distribution applies to the number of items v_j assigned to the jth bin, for any j. Since the sequence a_i, $i \geq 1$, consist of independent and uniformly distributed random variables we obtain that the random variables v_j, $j = 1, 2, \ldots$ are also independent and identically distributed.

Thus the random solution value $\mathrm{NF}(n)$ is related to the renewal process R_n, associated with the sequence v_j and defined by

$$R_n = \sup\Big\{m \geq 0 \mid \sum_{j=0}^{m} v_j \leq n, v_0 = 0\Big\}$$

in that $\mathrm{NF}(n) = R_n$. To compute $\mathrm{E}(\mathrm{NF}(n))$, it suffices to compute the discrete renewal function $\mathrm{E}(R_n)$.

We first observe that $E(v_j) = \sum_{k=0}^{\infty} \frac{1}{k!} = e$, that $E(v_j)^2 = 2\sum_{k=1}^{\infty} k \Pr\{v_j \geq k\} - e = 3e < \infty$, and that the distribution of v_j satisfies the property that g.c.d. $\{n \mid n > 0, \Pr\{v_j = n\} > 0\} = 1$. Hence, applying the weak renewal theorem (see. [40], p. 330.) we get

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NF}(n))}{n} = \frac{1}{e}. \tag{113}$$

Using the strong renewal theorem we can obtain a stronger result

$$\lim_{n\to\infty} \left(\mathrm{E}(\mathrm{NF}(n)) - \frac{n}{e}\right) = \frac{2}{e} - 1. \tag{114}$$

5.3.4 Next Fit Decreasing algorithm

In this section we adopt and analyse the next fit decreasing algorithm (NFD) to our model. Given a list of n items of size $a_1, a_2, \ldots, a_n$ $(0 \leq a_i \leq 1)$.

Algorithm 5.3.3. *The NFD algorithm for the dual bin packing problem first reindexes the items in decreasing order and then applies the NF algorithm to this new list.*

To analyse the behavior of the expected solution value $E(\mathrm{NFD}(n))$, we approximate – as in the case of the classical bin packing problem – the performance of NFD algorithm by that of the slices NFD algorithm with parameter r (SNFD_r).

Algorithm 5.3.4. *In the r (SNFD_r) first items larger than $1/r$ are packed according to the NFD algorithm, the last opened bin is completed by adding elements of decreasing size smaller than $1/r$ and any remaining items are packed in groups of size $r+1$ (possibly at the expense of feasibility; but in the time limit, this will not hurt).*

The number of bins used by this algorithm on n times is denoted by $\mathrm{SNFD}_r(n)$. It is clear that

$$\mathrm{SNFD}_r(n) \geq \mathrm{NFD}(n) \qquad (r > 1)$$

and

$$\lim_{r\to\infty} \mathrm{SNFD}_r(n) = \mathrm{NFD}(n) \qquad \text{(a.s.)}$$

Along our proof we follow the technique we used in the classical bin packing investigating the behavior of the NFD algorithm from probabilistic point of view. Let k_i be the number of items whose size falls in the interval $(1/(i+1), 1/i]$ $i \geq 1)$ and let $K_i = k_i + k_{i+1} + \ldots$. Then, for any $r > 1$,

$$\mathrm{SNFD}_r(n) \leq k_1 + \frac{k_1}{2} + \ldots + \frac{k_{r-1}}{r} + \frac{K_r}{r+1} + r,$$

where the last term is induced to allow for the rounding errors.

Since a_i are uniformly distributed and independent, we obtain $\mathrm{E}(k_i) = n/i(i+1))$ and $\mathrm{E}(K_i) = n/i$. Hence

$$\mathrm{E}(\mathrm{SNFD}_r(n)) \leq n \sum_{i=1}^{r-1} \frac{1}{i(i+1)^2} + \frac{n}{r(r+1)} + r$$

and this implies that, if r is suitably chosen as a function of n, then

$$\limsup_{n\to\infty} \frac{\mathrm{E}(\mathrm{NFD}(n))}{n} \leq \sum_{i=1}^{\infty} \frac{1}{i(i+1)^2}. \tag{115}$$

For giving a lower bound on $\mathrm{NFD}(n)$ we ignore those items from the list which are packed by NFD in bins more than one interval $(1/(i+1), 1/i]$, and we also ignore items smaller than $1/r$. Then

$$\mathrm{NFD}(n) \geq \left(\frac{k_1}{2} - 1\right) + \left(\frac{k_2}{3} - 1\right) + \ldots + \left(\frac{k_{r-1}}{r} - 1\right),$$

and we find

$$\liminf_{n\to\infty} \frac{\mathrm{E}(\mathrm{NFD}(n))}{n} \geq \sum_{i=1}^{\infty} \frac{1}{i(i+1)^2}. \tag{116}$$

Since

$$\sum_{i=1}^{\infty} \frac{1}{i(i+1)^2} = \sum_{i=1}^{\infty} \left(\frac{1}{i} + \frac{1}{i+1} + \frac{1}{(i+1)^2}\right) = 2 - \sum_{i=1}^{\infty} \frac{1}{i^2} = 2 - \frac{\pi^2}{6},$$

we obtain from (115) and (116) that

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NFD}(n))}{n} = 2 - \frac{\pi^2}{6} = 0.3551\ldots.$$

We note that

$$\lim_{n\to\infty} \frac{\mathrm{E}(\mathrm{NF}(n))}{n} = \frac{1}{e} = 0.3679\ldots,$$

so that the expected performance of the NF algorithm is better than the (expected) performance of the NFD algorithm. For the classical bin packing problem, exactly the reverse is true.

6 Job scheduling on m identical machines

In this chapter we discuss the problem of online scheduling: there are given a list $\{J_1, \ldots, J_n\}$ of n jobs and m identical machines $\{M_1, \ldots, M_m\}$. Each job J_i has a fixed processing time p_i. The jobs may be processed in any order, but preemption is not allowed. Our goal is to minimize the makespan, i.e. the maximum completion time over all jobs in a schedule. This problem is obviously $\mathcal{NP}$-complete. So we are interested in algorithms that produce "rather" good approximative solutions. The quality of a algorithm H is measured by its worst case ratio

$$R^H(m) = \limsup\{C^H(L)/C^*(L) : \ L \text{ is a list of jobs}\}$$

where $C^H(L)$ denotes the makespan produced by the algorithm H on m machines and the list L of jobs, and $C^*(L)$ denotes the corresponding makespan in an optimum schedule.

We will investigate the *online* case, where we get the jobs one by one and must immediately decide by which machine the job is processed. Once a job is assigned to its machine, it is not allowed to move the job to another machine.

In 1969, Graham [62] suggested a simple – *List Scheduling* (LS) – algorithm to solve this online problem.

Algorithm 6.0.5. *LS always assigns a job to the machine that has the minimum load at the moment.*

Graham showed that LS constructs a schedule with makespan at most $2 - \frac{1}{m}$ times the optimum makespan.

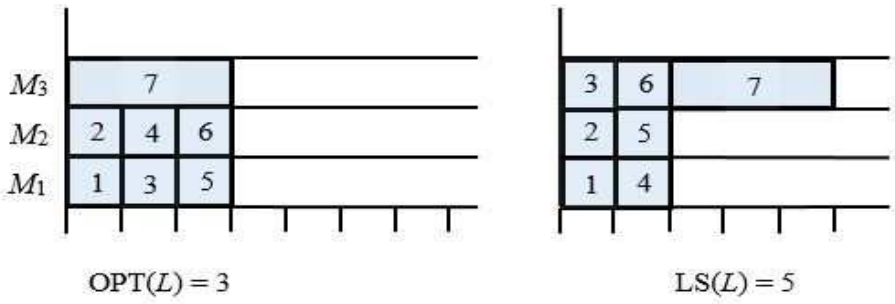

Figure 6.1. LS worst case example for $m = 3$.

As it was proven in [38] for $m = 2$ and $m = 3$, LS cannot be beaten, and so its worst case bounds are the best possible. Faigle, Kern and Turán also proved that for $m \geq 4$, no online algorithm with better competitive ratio than $1 + \sqrt{2}/2 \approx 1.707$. For a long time no algorithm was known with better competitive ratio than LS.

In the papers [52] and [21] machines we presented a algorithm that has an improvement on LSfor $m \geq 4$. To secede from the simple formulation of Graham's algorithm, our algorithm depends on the number of machines and from the processing time of the job that has to be scheduled next. Thus, by exploiting more information than LS does, we were able to improve on the antique $2 - \frac{1}{m}$ worst case bound.

6.1 Lower bounds for online scheduling

All we do in this section is proving the following theorem. This result – in a slightly different form – is also included in the discussion of [38].

Theorem 6.1. (Galambos and Woeginger, [52]). *There is no online scheduling algorithm with worst case guarantee better than*

(i) $3/2$ *for 2 machines*

(ii) $5/3$ *for 3 machines.*

(iii) $1 + \sqrt{2}/2$ *for* $m \geq 4$ *machines.*

Proof. Claims (i) and (ii) are easily proved by using the list $\mathcal{L}_2 = \langle 1, 1, 2\rangle$ for $m = 2$ and the list $\mathcal{L}_3 = \langle 1, 1, 1, 3, 3, 3, 6\rangle$ for $m = 3$. The exact arguments are analogous to those used for $m \geq 4$ and omitted.

For $m \geq 4$, we consider a sequence consisting of m times a job of length 1, followed by m times a job of length $1 + \sqrt{2}$ and ending with a single job of length $2 + 2\sqrt{2}$. If the algorithm puts two or more of the jobs with length 1 on the same machine, it does not get any other job. It produces a makespan of 2 whereas the optimum makespan is only 1 and we are finished. Otherwise, it puts one job with length 1 on every machine. If in the following the algorithm puts two or more of the jobs with length $1 + \sqrt{2}$ on the same machine, we are finished again. Instead of an optimum makespan $2 + \sqrt{2}$, it comes up with a $3 + 2\sqrt{2}$ makespan. Consequently, every machine gets one job with processing time 1 and one job with processing time $1 + \sqrt{2}$. But now the single final job completes the proof.

□

6.2 Upper bounds for online scheduling

We will denote the makespan in the optimal schedule by C^* and the makespan in the schedule constructed by our algorithm by C^H. As the algorithm gets the jobs one by one, the values of C^* and C^H vary during the algorithm. To simplify notation, we will identify each job with its length. The *load* L_i of a machine M_i is the sum of processing times over all jobs assigned to it. When we describe the algorithm, we will use $L^{\max}$ and $L^{\min}$ to denote the maximum and the minimum load over all machines, respectively.

We assume that $m \geq 4$ holds. To define the algorithm, we introduce two real numbers $\alpha(m)$ and $\beta(m)$, where $0 \leq \alpha(m) \leq 1/3$ and $1 \leq \beta(m) \leq 5/4$ holds. Moreover, we will assume that $(\beta(m) + 1)/\beta(m)^3 > \alpha(m) + 1$ holds. These two numbers depend on m and their exact value will be specified later. To simplify our presentation we write α and β instead of $\alpha(m)$ and $\beta(m)$.

Definition 6.1. *For nonnegative reals* x, y *we say that* x *is* similar *to* y $(x \sim y)$ *if*

$$\frac{y}{\beta} \leq x \leq \beta y$$

For a set S of nonnegative reals, we say that $\sim(S)$ holds, iff every two elements in S are similar.

Since the LS maintains the jobs greedily aiming a flat schedule, it barges a serious problem if it gets a big job at the end. We would like to avoid this situation in the constructed new algorithm. The main idea is the following: If the current schedule is not a "flat" one – i.e. the set of loads is not β-similar – we can apply the LS algorithm. If they are similar, but "not much", then we can load the current jobs M_1 until $L^{\min} \sim L^{\max}$. If the set of loads are "strongly" similar – which is measured by $L_1 > \alpha L_m$ – we load the actual job to M_2.

Our algorithm – called *Refined List Scheduling*, RLS – is as follows.

(1) Reorder the machines so that $L_1 \leq L_2 \leq \ldots \leq L_m$ holds. Let x be a new job given to the algorithm.

(2) If $\not\sim (L_1 + x, L_2, L_3, \ldots, L_m)$ then put x on M_1 and goto (1).

(3) Else if $L_1 > \alpha L_m$ then put x on M_2 and goto (1).

(4) Else if $L_1 \leq \alpha L_m$ then

  (4.1) Put x on M_1.

  (4.2) While $L^{\min} \sim L^{\max}$ do: put all new jobs on M_1.

  (4.3) Goto (1).

Table 6.1. The steps of the RLS algorithm

The algorithm terminates when no more job is given to it. Observe that new jobs are given to the algorithm only in steps (1) and (4.2). In step (4.2) M_1 is generally *not* the machine with minimum load any more, since the algorithm is putting jobs on M_1 until it has the maximum load.

First we will derive a sequence of claims and lemmas that lead to a number of different worst case bounds. Having derived all these bounds, we will fix the values of α and β so as to minimize the maximum of all these bounds.

Claim 6.1. *Each time the algorithm reenters step (1), $\not\sim (L_1, \ldots, L_m)$ holds.*

Proof. The proof is a simple inductive argument. When the algorithm reenters step (1) the first time, there is only one nonempty machine and the claim follows. If the algorithm performed step (2)last, the claim holds by definition; if it last performed step (3), the claim holds by induction, as step (3) does not increase $L^{\min} = L_1$. If it reenters from step (4), $L^{\min} \not\sim L^{\max}$ holds.

□

Lemma 6.1. *When the algorithm leaves step (2), the inequality*

$$(L_1 + x)/C^* < (2m - 2 + \beta)/(m - 1 + \beta)$$

holds.

Proof. By Claim 6.1, before x was assigned to M_1 the machine loads were not similar. This implies

$$C^* \geq (\sum L_i + x)/m > \frac{1}{m}(m - 1 + \beta)L_1 + x/m$$

where the first inequality follows from averaging over all loads and the second inequality uses $L_i \geq L_1$ for $i = 1 \ldots m - 1$ and $L_m > \beta L_1$. Using $C^* \geq x$, we simply derive the claimed result.

□

Lemma 6.2. *When the algorithm leaves step (3), the inequality*

$$(L_2 + x)/C^* < \Big(m + (m - 2)\beta - (m - 1)\alpha\Big)/(m - 1)$$

holds.

Proof. As L_2 is the second smallest load, it is less or equal to the average value of the $m - 1$ largest loads. This gives

$$L_2 \leq (L_2 + \ldots + L_m)/(m - 1) \leq (mC^* - L_1 - x)/(m - 1).$$

Moreover, we show that

$$L_1 + x \leq \beta min\{C^*, L_m\}.$$

by considering the following two cases. If $L_2 > C^*$, $L_1 + x < C^*$ must hold (again by the standard averaging argument) and this implies the inequality because of $\beta \geq 1$. On the other hand, if $L_2 \leq C^*$ holds, we may use that $L_1 + x$ and L_2 are similar and derive the inequality. The two displayed inequalities together with $L_1 > \alpha L_m \geq \alpha C^*$ lead to

$$\begin{aligned} L_2 + x &\leq (mC^* - L_1 - x)/(m - 1) + x \\ &= mC^*/(m - 1) - L_1 + (L_1 + x)(m - 2)/(m - 1) \\ &< mC^*/(m - 1) + \Big(\beta(m - 2)/(m - 1) - \alpha\Big) min\{C^*, L_m\} \\ &\leq C^*\Big(m + (m - 2)\beta - (m - 1)\alpha\Big)/(m - 1) \end{aligned}$$

and the proof is complete.

□

Lemma 6.3. *If $L^{min} \sim L^{max}$ holds in step (4.2), then*

$$L^{max}/C^* \leq m\beta/(m - 1 + \beta).$$

Proof. By load averaging, $C^* \geq \sum L_i/m$ holds. Now we simply use $L_i \geq L^{\max}/\beta$ for $i \leq m-1$ and finish the proof.

□

Claim 6.2. *Assume the algorithm enters step (4). Denote by a_i the job assigned last to machine M_i. Then there exists a $k \leq 2$ so that*

$$(a) \quad L_j - a_j \leq L_1, \quad \forall j \geq 2, \;\; j \neq k$$

$$(b) \quad L_k - a_k \leq \beta L_1,$$

Proof. Choose $k \geq 2$ so that M_k is the machine that recieved its last job a_k earlier than all other machines M_j received their last job a_j, $j \geq 2$ Since $\sim (L_1 + x, L_2, \ldots, L_m)$ and $L_1 \leq \alpha L_m$, we have $L_1 \leq \alpha\beta L_k$ and $L_1 \leq \alpha\beta L_j$.

Proof of (a). Since a_j is the last job on M_j, it could not be placed in Step 4.1. We consider the cases where a_j was placed in Step 2, 3, or 4.2. Let us denote the workload of machine M_i by L'_1, $1 \leq i \leq m$, just before machine M_j receives its last job a_j.

Suppose that M_j received its last job in Step 4.2. Before M_j received a_j, the current schedule at that time satisfied $\sim (L'_1, L'_2, \ldots, L'_m)$. Thus

$$L_1 \geq L'_1 \geq \frac{1}{\beta} L'_k \geq \frac{1}{\beta} L_k \geq \frac{1}{\alpha\beta^2} L_1.$$

This contradicts our initial assumption that $\alpha \leq \frac{1}{3}$ and $\beta \leq \frac{5}{4}$. Therefore we conclude that a_j was not placed in Step 4.2.

Suppose that M_j received its last job in Step 3. This means that L'_j was the second smallest workload at that time. Denote the minimum workload at that moment by L'_{min} If we had assigned job a_j to the machine of minimum workload, the schedule would have become similar. This implies that $L'_k \sim L_{min} + a_j$ and $L'_k \sim L'_j$. Since $L_j \sim L_k$, we can write $L_j = xL_k$, where $\frac{1}{\beta} \leq x \leq \beta$. Then

$$L_j = xL_k = xL'_k \leq x\beta L'_j = x\beta(L_j - a_j).$$

On the other hand, we have

$$\frac{1}{\beta} L_j - xa_j = x\Big(\frac{1}{\beta} L'_k - a_j\Big) \leq xL'_{\min} \leq xL_1 \leq x\alpha\beta L_k = \alpha\beta L_j.$$

Combining the above two relations gives us

$$\Big(\frac{1}{\beta} - \alpha\beta\Big) L_j \leq xa_j \leq \Big(x - \frac{1}{\beta}\Big) L_j \quad \rightarrow \qquad \frac{1}{\beta} - \alpha\beta \leq x - \frac{1}{\beta} \leq \beta - \frac{1}{\beta},$$

which contradicts the initial assumption that $\beta + 1 > \beta^3(\alpha + 1)$ and $\beta \geq 1$. We then conclude that a_j was not placed in Step 3.

The only possibility that remains is that M_j received a_j in Step 2. This means that L'_j was the smallest workload at the moment. Hence $L_j - a_j = L'_j \leq L'_1 \leq L_1$.

Proof of (b). We again need to consider the three cases where a_k was placed in Step 2, 3, or 4.2. If a_k was placed in Step 2, machine M_k had the smallest workload at that time. Thus $L_k - a_k \leq L_1$. If a_k was placed in Step 3, then from conclusion (a) we know that all other machines had workloads less than or equal to L_1 at that time. Since M_k had the second smallest workload, we conclude that $L_k - a_k \leq L_1$. The last possibility that remains is that a_k was placed in Step 4.2. Since the schedule was similar just before a_k was placed, we conclude that $L_k - a_k \leq \beta L_1$.

□

Lemma 6.4. *When the algorithm leaves step (4.2), then*

$$L^{max}/C^* \leq \beta^2/(2 - 2\alpha\beta) + 1.$$

Proof. Let a_i, $1 \leq i \leq m$ denote the last job assigned to M_i before the algorithm enters the while-loop (hence, a_1 is equal to x). As $L_1 + x \sim L_m$, we have $L_1 + x \geq L_m/\beta$. Together with Claim 6.2 and $L_1 \leq \alpha L_m$ this gives

$$a_i \geq (\frac{1}{\beta} - \alpha)L_m$$

for all $1 \leq i \leq m$. Finally, we define y to be the last job assigned to machine M_1 before the algorithm leaves step (4.2) and note that after this assignment $L^{\max}$ is the load of machine M_1.

We distinguish two cases. If $y \geq (\frac{1}{\beta} - \alpha)L_m$ holds, we use the pigeon hole principle: There are $m+1$ jobs with length greater or equal $(\frac{1}{\beta} - \alpha)L_m$, hence the optimum schedule must put two of them on the same machine. This gives $C^* \geq (\frac{2}{\beta} - 2\alpha)L_m$. Using $C^* \geq y$ and the fact that $L^{\max} - y \sim L_m$ we have

$$L^{\max} \leq \beta L_m + y \leq \beta^2 C^*/(2 - 2\alpha\beta) + C^*$$

Otherwise, if $y < (\frac{1}{\beta} - \alpha)L_m$ holds, we derive in an analogous way that $C^* \geq (\frac{1}{\beta} - \alpha)L_m + y$ and $C^* \geq y$. This gives

$$L^{\max} - C^* \leq (\beta - \frac{1}{\beta} + \alpha)L_m \leq (\beta^2 - 1 + \alpha\beta)L^{\min}$$

But now $L^{\min} \leq C^*$ and $\beta^2 - 1 + \alpha\beta$ is smaller or equal to $\beta^2/(2 - 2\alpha\beta)$, if $0 \leq \alpha \leq 1/3$ and $1 \leq \beta \leq 5/4$. So we end with the claimed bound in all cases.

□

Theorem 6.2. (Galambos and Woeginger, [52]). *For $m \geq 4$, the worst case performance of RLS is less or equal to $2 - \frac{1}{m} - \varepsilon_m$, with ε_m some small positive real depending on m. As $m \to \infty$, $\varepsilon_m \to 0$.*

Proof. Summarizing, Lemmas 3, 4, 5 and 7 give the following worst case bounds. In each inequality C^* denotes the momentary optimum makespan.

$$(L_1 + x)/C^* < (2m - 2 + \beta)/(m - 1 + \beta) \quad (117)$$

$$(L_2 + x)/C^* < \Big(m + (m-2)\beta - (m-1)\alpha\Big)/(m-1) \quad (118)$$

$$L^{\max}/C^* \leq m\beta/(m - 1 + \beta) \quad (119)$$

$$L^{\max}/C^* \leq \beta^2/(2 - 2\alpha\beta) + 1 \quad (120)$$

Let us examine the worst case performance of RLS. A worst case situation occurs after some job x was assigned to its machine. This happens either in step (2), in step (3), in the middle of step (4) or in the end of step (4); the four inequalities above give upper bounds on the worst case ratios in these four scenarios. Consequently, our goal is to minimize the maximum for each $m \geq 4$ over the four right hand side values under the restrictions

$$0 \leq \alpha \leq 1/3 \qquad 1 \leq \beta \leq 5/4 \qquad (\beta + 1)/\beta^3 > \alpha + 1. \quad (121)$$

We denote this minimum (that is an upper bound on the worst case ratio of RLS) by $R^{RLS}(m)$. First, it is easy to see that for $\beta \leq 2$, the right hand side of (119) is less than or equal to the right hand side of (117) and so we need not consider it in the optimization problem. For the remaining three values, we derive by using standard calculus that the minimum is taken if all three values are equal. Some substitutions lead to

$$\beta^4 + 2\beta^3(2m - 3) + \beta^2(3m^2 - 8m + 7) - 2(m-1)^2\beta - 2(m-1)^2 = 0.$$

For $m \leq 3$ this equality does not have a feasible solution. For each $m \geq 4$, this equality has a solution with $1 < \beta < 5/4$, since its value taken for $\beta = 1$ is negative and its value for $\beta = 5/4$ is positive. The solutions for some small values of m are stated in the third column of Table 6.2. As m tends to infinity, β tends from below to the positive root of $3\beta^2 - 2\beta - 2$. The restriction $(\beta + 1)/\beta^3 > \alpha + 1$ that was essential in the proof of Claim 6.2, is of no consequence for the minimization problem.

Finally, we determine from the right hand side of inequality (1) the value of $R^{RLS}(m)$. As $\beta > 1$ holds, we have $R^{RLS}(m) < 2 - \frac{1}{m}$. This means that RLS beats the worst case performance of List Scheduling for $m \geq 4$. However, asymptotically we have not improved the algorithm LS. Table 6.2 summarizes some of the results of this section.

□

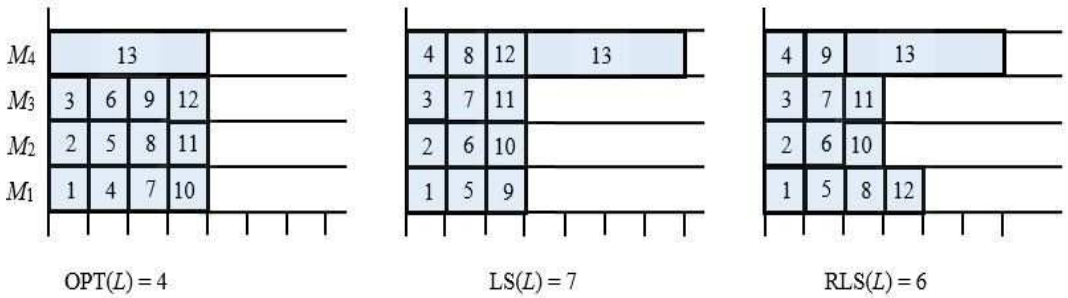

Figure 6.2. Schedules of the list $p_1 = \ldots = p_{12} = 1$, $p_{13} = 4$, for $m = 4$.

m	α	β	$R^{RLS}(m)$	$R^{LS}(m)$	ε_m	L.B.
2	—	—	—	1.50000	—	1.500
3	—	—	—	1.66667	—	1.667
4	0.27136	1.03132	1.74417	1.75000	0.006	1.707
5	0.25041	1.06722	1.78939	1.80000	0.011	1.707
6	0.23892	1.09106	1.82088	1.83333	0.012	1.707
7	0.23196	1.10814	1.84410	1.85714	0.013	1.707
10	0.22219	1.13921	1.88764	1.90000	0.012	1.707
100	0.21479	1.20731	1.98795	1.99000	0.002	1.707
∞	0.21525	1.21525	2.00000	2.00000	0.000	1.707

Table 6.2. Approximate values of α and β for different m.

6.3 Concluding remarks

Our algorithms are no improvement for $m \to \infty$, as their worst case ratio tends to 2. This follows from the fact that we schedule jobs on those machines which have the two smallest workloads. It is not too complicated to prove that any algorithm which schedules jobs only on machines which have the k smallest workloads with a fixed k, can not improve asymptotically the performance of LS (see [101]).

But we "let the jinn out of the bottle", and after publishing our paper we "warmed up" the question: are there online scheduling algorithms with worst case ratio smaller than $2 - \delta$ for all m and some fixed $\delta > 0$? Table 6.2 shows that our algorithm improves the Graham's LS for small m (< 10), but it does not have significant improvements for large m-s. It was very interesting that within a few years the problem was in the centre of research of the research of scheduling algorithms.

The first improvement was given by Bartal, Fiat, Karloff, and Vohra [12]. They presented an algorithm that has a $2 - \frac{1}{70}$ performance ration for every $m > 70$. Their algorithm was generalized by Karger, Philips, and Torng [72] giving an upper bound of 1.945 for all m. Bo Chen and van Vliet refined directly the RLS algorithm, and gave the slightly better algorithm RLS2 [22]. Van Vliet in [101] introduced the Modified List Scheduling (MLS) algorithm and he proved that its performance ratio is equal to 1.7333 for $m = 4$ which is a "visible" improvement for a small m. He also gave two lower bounds: for $4 \leq m \leq 10$ his new lower bound was 1.7310 and for $m = 80 + 8k$, $k \geq 0$, he proved a 1.8319 lower bound. Giving this lower bound for $m = 4$ the gap is getting very narrow: $(1.7310, 1.7333]$. Bartal, Karloff, and Rabani in [11] improved the lower bound to 1.8370 for $m \geq m_0$, with $m_0 = 3454$.

7 Complexity result for an open shop problem

An open shop problem consists of m machines $M_1, \ldots, M_m$ and n jobs $J_1, \ldots J_n$. Each job J_i consists of m operations $O_{i1}, \ldots O_{im}$, and O_{ij} has to be processed on machine M_j for p_{ij} time units without preemption. One machine can process at most one operation at a time and a job can be processed by at most one machine at a time. Operations of the same job can be processed in any order. The problem consists of allocating the machines to the jobs so as to optimize a certain objective function (e.g. maximum completion time, maximum tardiness) subject to various constraints on the processing technology (e.g. precedence constraints, release dates).

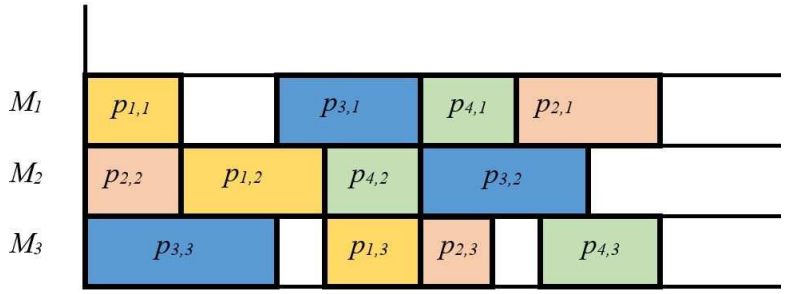

Figure 7.1. A feasible open shop schedule for three machines and four jobs.

Lot of work has been done on the special case of open shop problems with unit execution time (for short UET) open shop. In this case $p_{ij} =$ holds for all $1 \leq i \leq n$ and $1 \leq j \leq m$. With a small number of exceptions , most open shop problems with arbitrary execution times are $\mathcal{NP}$-hard (e.g. minimizing maximum completion time, minimizing maximum tardiness) and most open shop problems with unit execution times are solvable in polynomial time (e.g. minimizing maximum completion time, minimizing maximum tardiness).

P. Brucker, J. Jurisch, and M. Jurisch in [20] show a nice correspondence between UET open shop problems and some special preemptive scheduling problems on parallel machines. Due to this correspondence, they provided a unified framework for deriving polynomial time algorithms for UET open shop problems. One of the cases that does not fit into this framework is the following: There are $m = 2$ machines and all jobs have release time r_i, due dates d_i, and weights w_i. The aim is to find a schedule that minimizes the sum of weights of all late jobs. Following the notations in [63] we characterize this problem as $O2|p_{ij} = 1, r_i| \sum w_i U_i$, where $U_i = 1$ if the job is late and $U_i = 0$ otherwise. Determining the complexity status of this problem is posed as an open question in [20]. (In fact the unweighted case with $w_i \equiv 1$ was open.)

In the paper [53] we answer this question by deriving a polynomial time algorithm. The main idea is to translate the problem into a related edge-weighted graph, and to compute a kind of minimum weight perfect 2-factor on this graph.

7.1 Graph-theoretical backgrounds

A perfect 2-factor of an undirected graph $G = (V, E)$ is a subgraph $G' = (V', E')$ of G so that $V' = V$ and every vertex in G' has a degree 2. The following result was published in [80].

Proposition 7.1. (Lovász and Plummer, [80]) *For an edge-weighted graph $G = (V, E)$ a minimum weight perfect 2-factor can be computed in $O(|E|^2 \log |V|)$ time.*

Now we introduce the concept of $(2, *)$-factor, which we define as follows. For a graph $G = (V, E)$ with vertex-partition $V = (V_2, V_*)$, a $(2, *)$-factor is a subgraph $G' = (V', E')$ of G such that $V' = V$ and $\deg(v) = 2$ if $v \in V_2$, and $\deg(v) \leq 2$ if $v \in V_*$.

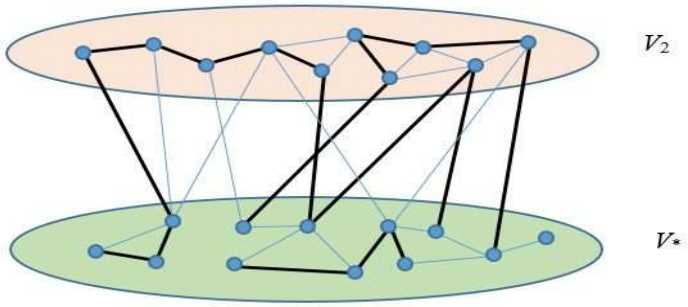

Figure 7.2. A $(2, *)$-factor to a graph $G = (V, E)$

Lemma 7.1. *For an edge-weighted graph $G = (V, E)$ with vertex-partition $V = (V_2, V_*)$ and $|E| \geq |V|$, a minimum weight $(2, *)$-factor can be computed in $O(|E|^2 \log |V|$ time.*

Proof. W.l.o.g. we can assume that $|V_*|$ is a perfect power of two (otherwise we add a set isolated vertices to V_*). Moreover, we add a complete binary tree to G so that its set of leaves exactly coincides with V_* and all interior vertices are newly created. The weights of all edges in the tree are set to zero. For every vertex v in the tree we introduce three new vertices $x(v)$, $y(v)$, and $z(v)$, together with the five edges $[x(v), y(v)]$, $[x(v), z(v)]$, $[z(v), y(v)]$, $[v, x(v)]$, and $[v, y(v)]$. All these new edges receive weight zero. The resulting new graph is called $G'' = (V'', E'')$. It is easy to check that $|V''| \leq |V_2| + 16\ |V_*|$ and $|E''| \leq |E| + 24\ |V_*|$.

We will show how to transform any perfect 2-factor in G'' into a $(2, *)$-factor in G of equal weight and vice versa.

(i) Consider a minimum weight perfect 2-factor in G'' and remove all edges from it which are not in the original edge set E. Since none of the removed edges incident to a vertex in V_2 and all of the removed edges have weight zero, so this operation results in a $(2, *)$-factor in G.

(ii) Now we consider a minimum weight $(2, *)$-factor in G. We claim that its edge set can be extended to a perfect 2-factor of G''. First we deal with vertices in V_* that have degree one in the $(2, *)$-factor. Clearly, it must be an even number of them. We put a rabbit on every such vertex, and then we make all rabbits run simultaneously towards the root of the tree at constant speed. If two rabbits collide, both disappear immediately.

Afterwards, all tree-edges traced by the rabbits are added to the $(2,*)$-factor. This leaves us with two types of vertex-subsets in the tree: vertices with degree zero (T_0) and vertices with degree two (T_2).

(a) If $v \in T_0$ then we add the edges $[x(v), y(v)]$, $[x(v), z(v)]$, $[v, z(v)]$, and $[v, y(v)]$.

(b) If $v \in T_2$ then we add the edges $[x(v), y(v)]$, $[x(v), z(v)]$, $[z(v), y(v)]$.

Having added these edges every vertex in V'' has exactly two edges, and all edges added to the $(2,*)$-factor have of weight zero.

Summarizing, the weight of a minimum weight perfect 2-factor in G'' equals the weight of a minimum weight $(2,*)$-factor in G. Since $|V''| = O(|V|)$ and $|E'' = O(|E|)$ holds, the statement of the Proposition 7.1 completes the proof of the lemma.

□

7.2 Polynomial time result

Let there be given two machines M_1 and M_2 together with n jobs $J_1, \ldots, J_n$. Each jobs J_i must be processed for one time unit on both machines, and the processing can not start before some release time r_i. In case the processing ends later than time d_i a penalty w_i has to be paid. The goal is to find a schedule that minimizes the sum of the weighted number of late jobs. We assume that all the numbers r_i, d_i, w_i are integers.

All the jobs with $d_i - r_i \geq 2n$ can always be scheduled in time, so they need not be considered. Therefore we can assume that for every job J_i, $d_i - r_i < 2n$. We define the set

$$T = \bigcup_i \big\{r_i, d_i - 1\big\},$$

and we define

$$t_{\min} = \min\big\{t \in T\big\}, \qquad t_{\max} = \max\big\{t \in T\big\}.$$

Next, we describe how to construct an appropriate edge-weighted undirected graph $G = (V, E)$ from the open shop problem. The construction is as follows.

- For every job J_i, we introduce four vertices a_i, b_i, c_i, and v_i. Vertex v_i is called the *job-vertex* corresponding to J_i.
- For every time-slot between $t_{\min}$, and t_{max} we introduce a vertex t_j.
- For every job J_i, we introduce the edges $[a_i, b_i]$, $[a_i, c_i]$, $[b_i, c_i]$, and $[a_i, v_i]$, all with edge-weight zero. Moreover, we introduce the edge $[b_i, v_i]$, with edge-weight w_i.
- For every job J_i and for every $r_i \leq t_k \leq d_i$, we define an edge $[v_i, t_k]$ with weight zero.
- We define the vertex sets $V_* = \big\{t_j \mid t_{\min} \leq t_k < d_i\big\}$ and $V_2 = V \setminus V_*$.

Claim 7.1. *If for the above defined open shop problem there exits a schedule for which the weighted number of late jobs is at most W, then there exists a $(2,*)$-factor of the graph G with total edge-weight at most W.*

Proof. For every late job J_i, we chose the edges $[a_i, c_i]$, $[b_i, c_i]$, $[a_i, v_i]$, and $[b_i, v_i]$. This gives a cycle with total weight w_i. For every job J_i that is processed in time, say at time t_1^* on the first machine and at time t_2^* on the second machine with $r_i \leq t_1^* \neq t_2^* < d_i$, we choose the five edges $[a_i, b_i]$, $[a_i, c_i]$, $[b_i, c_i]$, $[v_i, t_1^*]$, and $[v_i, t_2^*]$. This gives a three-cycle with total weight zero and two dangling edges, each with weight zero.

Since a vertex with degree ≥ 3 would imply that in the schedule there is a time-slot where at least three operations are performed concurrently, with these chosen edges we get that every vertex t_k has degree at most two. Hence, all degree conditions are fulfilled and it is easy to verify that the total weight of the chosen edges is at most W.

□

Claim 7.2. *If there exists a $(2,*)$-factor with total edge-weight at most W for the above constructed graph G, then there also exits a schedule for the open shop problem with sum of the weighted number of late jobs at most W.*

Proof. Consider some job J_i for which the edge $[b_i, v_i]$ (with weight w_i) is not in the $(2,*)$-factor. For this i, the vertex b_i has only two possible neighbors left for the $(2,*)$-factor, the vertices a_i and c_i. Since c_i is of degree two in G, both edges $[c_i, a_i]$ and $[c_i, b_i]$ incident to it must be in the $(2,*)$-factor. This forces the three-cycle a_i, b_i, c_i to be in the $(2,*)$-factor. Consequently, v_i must have two other neighbors in the $(2,*)$-factor, and the only remaining candidates are among the vertices t_k with $r_i \leq t_k < d_i$.

Next, we define an auxiliary bipartite graph $G'' = (V'', E'')$ that is a subgraph of the $(2,*)$-factor. V'' contains all vertices v_i for which $[b_i, v_i]$ is not in the $(2,*)$-factor, together with all vertices t_k with $t_{\min} \leq t_k \leq t_{\max}$. E'' contains all edges in the $(2,*)$-factor connecting some v_i in V'' to some t_k. Because of the above argument, for every job-vertex $v_i \in G''$, $\deg(v_i) = 2$. Hence, E'' is the disjoint union of several even cycles and several paths consisting of an even number of edges (since a path must start and end at time-vertices). We two-color the edges of the cycles and path alternating with colors red and blue. Obviously, in the resulting coloring every vertex v_i is incident to one edge of color red and to one edge of color blue.

Finally, we define a schedule S as follows. Jobs J_i for which the corresponding vertex v_i is in G'', have two incident edges $[v_i, t_1^*]$ and $[v_i, t_2^*]$ with $r_i \leq t_1^* \neq t_2^* < d_i$. We can assume that the edges are colored by red and blue, respectively.

In schedule S, job J_i is processed at time t_1^* on machine M_1, and at time t_2^* on machine M_2.

Clearly, ny this ordering all jobs whose corresponding vertices are in G'' are scheduled in time without collisions. The remaining jobs are scheduled arbitrary. The weighted number of late jobs is at most the total weight of this second class of jobs, and by our construction this weight is at most W.

□

Theorem 7.1. (Galambos and Woeginger, [53]). *The UET open shop problem* $O2|p_{ij} = 1, r_i|\sum w_i U_i$ *with* n *jobs is solvable in* $O(n^4 \log n)$ *time.*

Proof. Since every job J_i fulfills $d_i - r_i < 2n$, the number of relevant time-vertices in our construction is $O(n^2)$. The number of edges between job-vertices and time-vertices is at most $O(n^2)$. The number of remaining vertices and edges is linear in n.

Thus, $|V| = O(n^2)$ and $|E| = O(n^2)$ holds and the combination of Lemma 7.1 with Claim 7.1 and Claim 7.2 completes our argument.

□

7.3 Concluding remarks

In the conclusion of our paper we mentioned that that the main open problem concerns the complexity status of $O3|p_{ij} = 1, r_i|\sum w_i U_i$. We proposed the question: "will we encounter once more the usual combinatorial explosion when going from two to three?" For this question the answer was given by P. Baptiste in [10] proving that the problem $Om|p_{ij} = 1, r_i|\sum w_i U_i$ can be solved for any fixed value of m in $O(n^{4m^2-m+10})$ time with the help of dynamic programming.

8 Coupled tasks scheduling problem

The coupled task problem can be defined as follows. We are given n jobs, each of them consisting of two distinct tasks (operations). The sequence of these tasks is fixed and also a fixed delay time has to pass between the two tasks. So, each job i can be described by a triple (a_i, L_i, b_i), where the values represent the processing time of the first task, the delay time between the tasks and the processing time of the second task, respectively. It is important that in this type of problems the second task must be scheduled *exactly* $L_i + a_i$ units after the start of the first one. During the delay time the machine is idle, and so it can process other jobs in this interval. The aim is to schedule n coupled tasks on one machine in such a way that no two tasks overlap and the latest completion time (also called as makespan, and denoted by $C_{\max}$) of the jobs is minimized. Preemption is not allowed, each task has to be processed continuously.

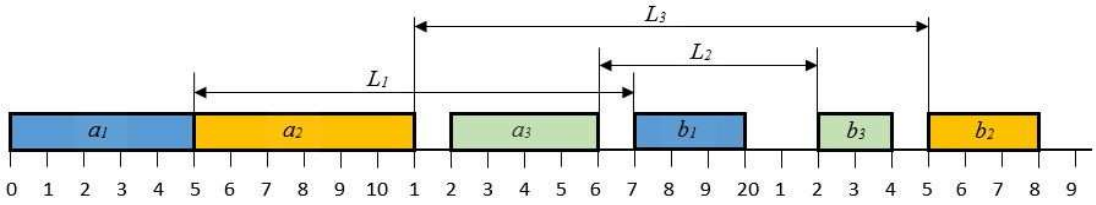

Figure 8.1. A feasible schedule for three jobs

If we use the standard three-fields notation $\alpha \,|\, \beta \,|\, \gamma$ and follow the idea in [85] we will write *Coup-Task* in the second field indicating that in fact all jobs consist of coupled tasks, then the formulation $1 \,|\, \text{Coup-Task}, a_i = a, L_i = L, b_i = b \,|\, C_{\max}$ denotes a coupled task problem where all jobs are identical. We call the problem as *Identical Coupled Task Problem (ICTP).*

The general problem $1 \,|\, \text{Coup-Task} \,|\, C_{\max}$ was first studied by Shapiro [97]. He discussed practical situations where the problem arises and gave three simple algorithms for this $\mathcal{NP}$-hard problem (see [91]). Unfortunately, these algorithms have not been analysed in detail. Their efficiency was shown only by experimental results. Orman and Potts studied the problem from a complexity point of view (see [85]). They covered all subcases except for the identical case $a_i = a$, $L_i = L$, $b_i = b$.

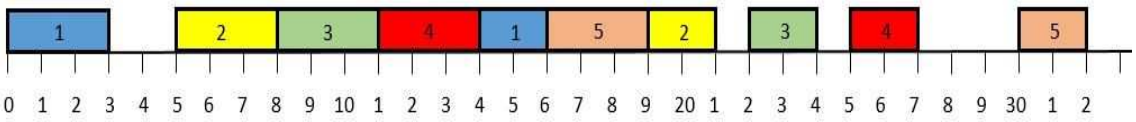

Figure 8.2. An optimal schedule for five identical jobs with values $a = 3$, $b = 2$, $L = 11$

In paper [2] we investigated this special case, and we show the results published in this paper.

8.1 Basic definitions

Throughout the section we denote the operation time of the first and second task of a job by a and b , the common delay time by L, and the number of jobs by n. We always assume that a, b and L are integers, because we can convert rationals to integers by multiplying them with a suitable number.

Without loosing generality we can assume that $a > b$, since the following theorem (proven in [85]) holds.

Theorem 8.1. *For given a and b the problem $1 \,|\, \textit{Coup-Task}, a_i = a, L_i = L, b_i = b \,|\, C_{\max})$ and its reverse $1 \,|\, \textit{Coup-Task}, a_i = b, L_i = L, b_i = a \,|\, C_{\max}$ are equivalent.*

We may also assume that $a < L < (n-1)a$, since otherwise the simple greedy algorithm (i.e., start the first job at time 0, and each subsequent job as soon as possible) gives the optimal solution.

We will use certain patterns to represent idle and busy times of the processor with respect to a given schedule. These patterns consist of 0's and 1's indicating if the processor is idle or busy during a considered time unit.

Definition 8.1. *A $0-1$ sequence of length L is called $P(a, L, b)$ pattern, if it contains 1's only in blocks of length b and if each such block is followed by at least $a - b$ pieces of 0's.*

For example, the sequence 11000110110 is a $P(3, 11, 2)$ pattern, but the sequence 11011100110 is not, because it contains a block of 1's of length 3. The set of all possible $P(3, 7, 2)$ patterns is

$$\{0000000, 1100000, 0110000, 0011000, 0001100, 1101100, 0000110, 1100110, 0110110\}.$$

Lemma 8.1. (Ahr, Békési, Galambos, Oswald and Reinelt, [2]). *The total number $np(a, L, b)$ of possible $P(a, L, b)$ patterns is $O(cr^L)$, where r is the absolute value of the root with the greatest absolute value of the equation $1 + x^{a-1} = x^a$.*

Proof. If $L < a$ we have $\mathrm{np}(a, L, b) = 1$, since in this case only the pattern $0 \ldots 0$ satisfies the condition. First, we show the recursion

$$\mathrm{np}(a, L, b) = \mathrm{np}(a, L-1, b) + \mathrm{np}(a, L-a, b) \quad \text{for} \quad L \geq a.$$

In this case the patterns can be classified into two groups. The first group contains those patterns, which end with a block of 1's and $a-b$ 0's. The second group consists of patterns which end with more than $a-b$ 0's. The number of patterns in the first group is obviously $\mathrm{np}(a, L-a, b)$ and the second group contains $\mathrm{np}(a, L-1, b)$ elements. And therefore the recursion holds.

The characteristic equation of the recursion is $1 + x^{a-1} = x^a$, and the statement of the lemma continues with the well-known solution method of linear recursions.

□

Suppose we have started exactly k jobs and according to the schedule, no new job can be started before the first task of the last job. This means that job $k+1$ can only be started after the start of the last job, possibly before but in any case after its second task. The starting time of the new job depends on the idle time periods between the two tasks of the last job. We can represent the schedule in the gap of the last job by a $P(a, L, b)$ pattern. In principle, situations corresponding to every $P(a, L, b)$ pattern are possible. Figure 7.3 presents an example of a $P(a, L, b)$ pattern.

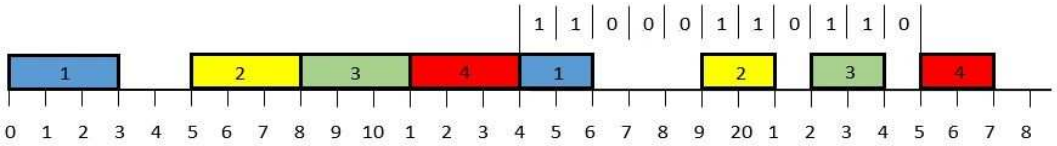

Figure 8.3. Gap of last job represented by a $P(3, 11, 2)$ pattern

We now investigate how a further job effects the schedule. It is obvious, that we can start a new job only at those positions, where we have an idle period of length at least a. This means that the corresponding $P(a, L, b)$ pattern contains a block of 0's of length a at that position. If we start the new job at a given position, the new gap pattern for the last job will be generated by a rule. First we copy some bits – their number depending on the position of the new job – from the right hand side of the original pattern, then add b bits of 1's and fill the pattern with 0's up to length L. It is also obvious that the increment of the makespan depends on the starting position of the new job. (Figure 7.4. illustrates the method of generating the new pattern.) More exactly, we can define the following operator.

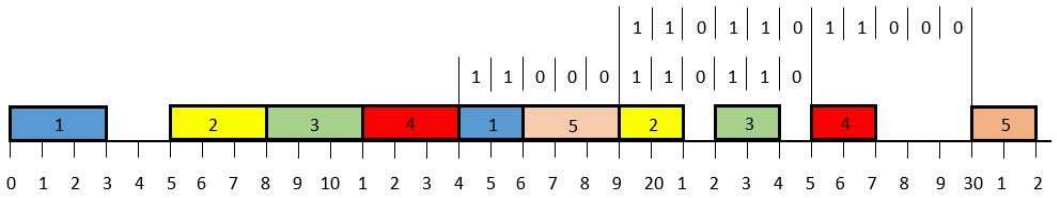

Figure 8.4. Generating the new pattern after starting a new job

Definition 8.2. *Let p be a $P(a, L, b)$ pattern and i, $1 \leq i \leq L - a + 1$, be an integer such that*

$$p[i] = p[i+1] = \ldots = p[i+a-1] = 0. \tag{122}$$

Then $S(p, i)$ is the 0-1 sequence

$$p[i+a]\ p[i+a+1]\ \ldots\ p[L]\ 1^b\ 0^{i+a-b-1}$$

where 1^k (0^k) denotes a string of k 1's (0's). Furthermore we define $w(p, S(p, i)) = i + a - 1$, which gives the increment of the makespan when applying $S(p, i)$.

Lemma 8.2. ([2]). *If p is a $P(a, L, b)$ pattern, then $S(p, i)$ is a also a $P(a, L, b)$ pattern for $1 \leq i \leq L - a + 1$.*

Proof. The statement follows directly from the definition of $S(p, i)$, because we replace some right hand side bits of p by one block of 1's of length b followed by at least $a - b$ 0's. □

8.2 Graph model and algorithm

In the above section we have shown that with every schedule we can associate a $P(a, L, b)$ pattern characterizing the state of the machine in the gap of the job currently started as the last job. It is obvious, that this pattern will usually change, when we add a new job to the schedule.

An evident idea is to represent the relations between the patterns by a directed graph. The vertices of the graph are the possible patterns and we connect two vertices by an arc, if the corresponding pattern can be followed by the other one when a job is added. We can also assign a weight to each arc, giving the increment of the makespan caused by this change of schedule.

Definition 8.3. *For given integers a, b and L we define the directed graph $G = (V, A)$ with arc weights $w(p, q)$, for every $(p, q) \in A$ as follows.*

- $V := \{p \mid p \text{ is a } P(a, L, b) \text{ pattern}\}$,
- $A := \{(p, q) \mid q = S(p, i) \text{ for } i,\ 1 \leq \mathrm{i} \leq \mathrm{L} - \mathrm{a} + 1\} \cup \left\{\left(p, 0^L\right) \mid p \in V\right\}$,
- $w(p, q) := \min_{1 \leq i \leq L-a+1} \{w(S(p, i)) \mid S(p, i) = q\}$.
- $w(p, 0^L) := a + b + L$.

Note that the same patterns may occur repeatedly in a schedule and can even be followed by itself.

So, the coupled tasks problem is transformed to find a minimal weight path in G starting from pattern 0^L consisting of exactly n arcs. Since the weight of each arc represents the increment of the makespan, the total weight of such a path is the real makespan minus the time of the first job, which is $a + b + L$. If we minimize the weight of the paths of length n, we minimize the makespan for n jobs.

More precisely, the makespan M of the identical coupled tasks scheduling problem with parameters a, L, b, n can be expressed as

$$M = \min_{p \in P(a,L,b)} \left\{w\left(p_0, p_1, ..., p_n\right) \mid p_0 = 0^L, (p_i, p_{i+1}) \in A, p_n = p\right\}$$

where

$$w\left(p_0, p_1, ..., p_n\right) = \sum_{i=0}^{n-1} w(p_i, p_{i+1})$$

and $p_0, .., p_n$ is a minimal weight path between 0^L and p.

This way our problem is reduced to a special shortest path problem in a directed graph with positive arc weights. In the following we deal with this task.

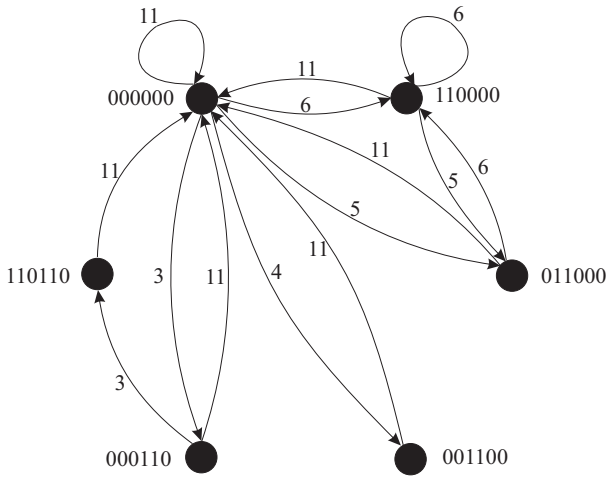

Figure 8.5. The weighted pattern graph for $P(3, 6, 2)$ patterns

Let $w_{0n}(p)$ denote the shortest path length from 0^L to p with n arcs. The following recursive formula applies for $w_{0n}(p)$:

$$w_{0n}(p) = \min_q \{w_{0n-1}(q) + w(q, p) \mid (q, p) \in A\} .$$

Now we can give an algorithm computing $w_{0n}(p)$ for each $P(a, L, b)$ pattern. $E(u)$ denotes the set of vertices that are connected to u by an arc.

Algorithm 4 :Algorithm MinWeight(G, n)

```
1. for each v ∈ V do
2.     D_v[0] := 0
3.     for i:=1 to n do
4.         D_v[i]:= ∞     P_v[i]:= −1
5.     endfor
6. endfor
7. for i:=1 to n do
8.     for each u ∈ V do
9.         for each v ∈ E(u) do
10.            if D_u[i − 1] + w(u, v) < D_v[i] then
11.                D_v[i]:=D_u[i − 1] + w(u, v)     P_v[i]:=u
12.            endfor
13.        endfor
14.    endfor
15. endfor
```

After executing *MinWeight* we have $w_{0n}(p) = D_p[n]$ for each pattern. Let us denote p^* the pattern which corresponds to the endnode of a shortest path having n edges. We

obtain the length of this path as $D_{p^*}[n] = \min\{D_v[n] \mid v \in V\}$ and the associated schedule by recursively visiting the nodes $P_v[n]$, starting with $v = p^*$.

Obviously, the time complexity of this algorithm is dominated by time complexity $O(|V|^2 n)$ of *MinWeight*. Since $|V| = O(r^L)$, the overall complexity of our algorithm is $O(nr^{2L})$.

Until now we have not said anything about the characteristics of the shortest path in the above defined graph. The graph can contain some circles, and each circle can be characterized by its length and total weight. Usually we are interested in the circles with the best weight/length ratio. If n is big, then an optimal solution contains the periodic repeats of the edges of a best circle and some other edges at the beginning and at the end of the shortest path.

We have implemented our algorithm using the above techniques and tested it for $n = 1000$ jobs and gap values up to $L = 30$ (in order to be able to use the machine arithmetic). The following tables presents running times and optimal values on a 433 MHz P-II PC and shows that solutions for fairly large problems can be computed in short time.

a	b	L	n	Time(ms)	n	Time(ms)
4	2	10	1000	60	2000	110
4	2	15	1000	381	2000	751
4	2	20	1000	3074	2000	6149
6	3	25	1000	2013	2000	4006
6	3	30	1000	12660	1500	18667
8	4	20	1000	140	2000	261
8	4	25	1000	470	2000	941
10	5	20	1000	60	2000	131
10	5	30	1000	561	2000	1091

Table 8.1. Running times of ICTP on a 433 MHz P-II PC.

a	b	L	BCL	BCW	$OPT1000$
4	2	10	4	26	6506
4	2	15	3	21	7013
4	2	20	4	26	6518
6	3	25	18	176	9797
8	4	25	3	37	12329
10	5	20	2	35	17485
10	5	30	1	15	15000

Table 8.2. Parameters of the optimal solutions of some ICTP problems.
(BCL = Best cycle length, BCW = Best cycle weight, $OPT1000$ = Optimum for 1000 jobs)

8.3 Concluding remarks

Our algorithm is linear in n, but it is exponential in L. Therefore, if the idle time is large then our algorithm may not be useful for such cases. Now, it is also true that $\lim_{a\to\infty} \sqrt[a-1]{a} = 1$, and so if a increases then the exponential factor will be less dominant. Some open questions remain for future research.

- The most important one is the question of the true complexity of the above problem.
- Another question is whether the size of the pattern graph can be decreased without losing the optimal solution.
- Furthermore, it would be interesting to construct good algorithms for non-polynomial cases of the coupled tasks problem with a thorough worst-case or average-case analysis.

9 Data compression

As the popularity of the Internet - and certain other local and larger computer networks - are growing rapidly, experts pay more and more attention to the efficiency of data transfer. Since it is influenced by both hardware techniques and software procedures, essentially there are two possibilities to increase the performance of data communication: either we speed up the computer-hardware, or *compress* the data to be transferred by effective processes. We will concentrate on the latter case. Depending on the characteristics of the data, different methods have been investigated for compressing textual data, pictures or voices. Text-compression differs from the other ones basically, since in that case - executing a decompressing process - one has to get the original text back without any distortion, while the other techniques allow some "smack" loose over the processes. In the field of the text-compression the early methods were dealing with character-coding techniques.

Using some classical results from the information and the probability theory such well known techniques were born as the Shannon-Fano code [39], the Huffman-code [67] and the Arithmetic coding [89]. These methods are based on the relative frequencies of the characters in a given string and so they are useful if one can keep the whole string in hand before encoding it, or one knows at least the stochastic behavior of the character-emitting source. Unfortunately, sometimes this is not the case, and the compressing process is to be pressed to decide on partial information or it needs some help based on the earlier processes.

One way to use the earlier observations in a data compression process is to construct a *dictionary*, which consists of pairs – (*source–words, code–words*) – of strings over two finite alphabets. Thereafter, a dictionary-based compression process divides the source string into substrings – each of them corresponds to some source-word – and substitutes them by code-words from the dictionary.

One class of the dictionary encoding methods uses *static dictionary*, i.e. a fixed dictionary that cannot be changed or extended during the encoding–decoding procedure. In this case our aim is to translate (encode) the source–string with the help of dictionary strings into a code–text with minimal length; in other words, we want to find a space-optimal encoding procedure.

There are certain parameters which can characterize a dictionary. If we consider the dictionary $D = \{(w_i, c_i) : i = 1, \ldots, k\}$, we will denote the length of each symbol of the source string S in bits by Bt. Furthermore, we will use the following notations:

$$\begin{aligned} \mathrm{lmax}(D) &= \max\{|w_i| \; i = 1, \ldots, k\}, \\ \mathrm{cmin}(D) &= \min\{\|c_i\| \; i = 1, \ldots, k\}, \\ \mathrm{cmax}(D) &= \max\{\|c_i\| \; i = 1, \ldots, k\}, \end{aligned}$$

where $|w_i|$ denotes the length of a string w_i in characters and $\|c_i\|$ the length of a code word c_i in bits. If the meaning is clear from the context, we will simply denote the bit length of each input character by Bt and omit the reference to the dictionary, i.e. we write lmax instead of $\mathrm{lmax}(D)$.

We always suppose that dictionaries contain all symbols of the input alphabet as a source-word (this ensures that any algorithm will terminate the encoding with a feasible solution), and we will refer to them as *general dictionaries.*

Definition 9.1. *A general dictionary is called*

- code uniform*, if all code-words are of equal length (i.e.* $\|c_i\| = \|c_j\|$, $1 \le i, j \le k$*),*
- non-lengthening*, if the length of any code-word never exceeds the length of the corresponding source-word (i.e.* $\|c_i\| \le |w_i|\mathrm{Bt}$, $1 \le i \le k$*),*
- suffix*, if with every source-word* w *also all of its proper suffices are source-words (i.e. if* $w = \omega_1\omega_2\cdots\omega_q$ *is a source-word* $\Rightarrow$ $\omega_h\omega_{h+1}\cdots\omega_q$ *is a source-word for all* $2 \le h \le q$*),*
- prefix*, if with every source-word* w *also all of its proper prefixes are source-words (i.e. if* $w = \omega_1\omega_2\cdots\omega_q$ *is a source-word* $\Rightarrow$ $\omega_1\omega_2\cdots\omega_h$ *is a source-word for all* $1 \le h \le q-1$*).*

Generally, an on–line algorithm will generate a suboptimal compression. One way to know how bad its performance is – compared to the optimal compression – a *worst–case analysis* can be devoted. As usually, if we want to construct a competitive algorithm, the *worst–case behavior* of an approximation algorithm is generally measured by its *asymptotic competitive ratio* which is defined as follows.

Let $D = \{(w_i, c_i) : i = 1, \ldots, k\}$ be a static dictionary and consider an arbitrary data compression algorithm A. Let $A(D, S)$ resp. $OPT(D, S)$ denote the compressed string produced by the algorithm A resp. the optimal encoding for a given source–string S. The length of these codings (expressed in the number of bits) will be denoted by $\|A(D, S)\|$ resp. $\|OPT(D, S)\|$. Then the *asymptotic competitive ratio* of algorithm A is defined as

$$R_A(D) = \limsup_{n\to\infty} \left\{ \frac{\|A(D,S)\|}{\|OPT(D,S)\|} : S \in S(n) \right\},$$

where $S(n)$ is the set of all text–strings with exactly n characters.

Schuegraf and Heaps in [93] proved that finding a space-optimal encoding is equivalent to the problem of finding a *shortest path* in a related directed edge-weighted graph as follows: Let $S = s_1 s_2 \ldots s_n$ be a source string. Then the corresponding graph $N = (V, A)$ has the vertex-set $V = \{v_0, v_1, \ldots, v_n\}$, and there is an edge $(v_i, v_{i+d}) \in A$ iff there exists a pair (source-word, code-word) in the dictionary so that the source-word consists of d characters that exactly match the original source-string in the positions $i+1, \ldots, i+d$. The weight of this edge is the length of the corresponding code–word. (If the code alphabet is binary then usually the weight is the number of bits of the corresponding code-word.)

Example 8.1. Consider the string THIS_IS_IT and the following dictionary [75]

source–word	T	H	I	S	_	TH	HIS	IS	IS_	IT
code-word	t	h	i	s	a	b	c	d	e	f
weight	x	x	x	x	x	x	x	x	x	x

The corresponding graph is

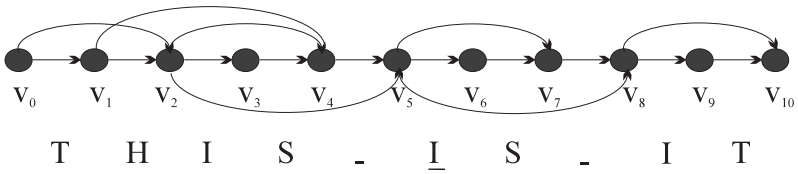

Figure 8.1. The graph belonging to the dictionary of the Example 8.1.

It is easy to see that in this model executing an optimal compression of the source–string S is equivalent to finding the shortest path from v_0 to v_n in the graph N. Unfortunately, sometimes our string is too large, so we can not choose this easy way to compress our data. In such situations we need to solve our problem in online way: Read a segment (record) of our text into a buffer, encode it using the dictionary, read the next segment etc. In the remaining part of the paper we suppose that the problem is represented by the graph model.

9.1 Upper bounds for general dictionaries

In the first part of this section we give general upper bounds for any compression algorithm. Then we investigate different online algorithms and analyse them from asymtotic worst case point of view.

Theorem 9.1. (Békési, Galambos, Pferschy, Woeginger, [14]). *Let D be a general dictionary. Then for any encoding algorithm A*

$$R_A^{\infty}(D) \leq \frac{(\mathrm{lmax}-1)\mathrm{cmax}}{\mathrm{cmin}}.$$

Proof. We consider source–strings where the A–path and an OPT–path are vertex disjoint with common endvertices v_0 and v_j. SO, the number of characters between the endvertices is j. Let us denote the number of vertices on the OPT–path by r. Because the two paths are vertex disjoint and each arc "consumes" at least one character, the number of arcs of the A–path between v_0 and v_j is at most $j-r$. However, the arcs on the optimal path have to cover the distance between v_0 and v_j. Hence, we have

$$r \geq \left\lceil \frac{j}{\mathrm{lmax}} \right\rceil. \tag{123}$$

This yields

$$R_A^{\infty}(D) \leq \frac{(j-r)\mathrm{cmax}}{r\mathrm{cmin}} = \left(\frac{j}{r}-1\right)\frac{\mathrm{cmax}}{\mathrm{cmin}} \leq \frac{(\mathrm{lmax}-1)\mathrm{cmax}}{\mathrm{cmin}}.$$

Figure 8.2. An example for upper bound, $j = 10$, $r = 2$, and $\mathrm{lmax} = 5$.

□

Setting cmax = cmin and using arguments as above we get the following

Corollary 9.1. *Let D be a code–uniform dictionary. Then for any encoding algorithm A*

$$R_A^\infty(D) \leq \mathrm{lmax} - 1.$$

9.2 Longest matching algorithm

Algorithm 9.2.1. *The longest matching algorithm (LM) processes the test string from left to right and at each position chooses the longest dictionary source-word that matches to original text. This substring is then replaced by the corresponding code word.*

LM is one of the oldest algorithms used for data compression (see [93]). Katajainen and Raita conjectured in [75] that "the coding result for prefix dictionaries can be weaker than the bounds derived for suffix dictionaries".

In this section, we will derive tight worst–case bounds for all four combinations of the property *prefix* with the properties code uniform / nonlenghtening. We will show that the LM algorithm for prefix dictionaries can behave as badly as possible: All bounds for *prefix* dictionaries with additional properties $\mathcal{P}$ are the same as the corresponding bounds for *general* dictionaries with properties $\mathcal{P}$; in other words, adding the prefix property does not improve any worst–case bound for *LM*.

Theorem 9.2. (Békési, Galambos, Pferschy and Woeginger, [15]). *Let D be a prefix dictionary. Then*

$$R_{LM}^\infty(D) \leq \frac{(\mathrm{lmax} - 1)\mathrm{cmax}}{\mathrm{cmin}},$$

and this bound can be attained.

Proof. The upper bound follows immediately from Theorem 9.1. The matching lower bound is derived by choosing the 3–symbol alphabet $\{u, v, w\}$ and the following prefix dictionary:

source–word	u	v	w	uv	vw^j $j=1,\ldots,\mathrm{lmax}-2$	$vw^{\mathrm{lmax}-2}u$
code–word	a	b	c	d	e_j	f
weight	cmax	cmax	cmax	cmax	cmax	cmin

where w^j is source-word of length j, and its each character is w. For $i > 0$, we consider the strings $S_i = u(vw^{\text{lmax}-2}u)^i$ with length $n = i(\text{lmax}+1)+1$.

Visualizing the corresponding network see Figure 8.3. It can be checked that $OPT(D, S_i) = af^i$ and $LM(D, S_i) = (dc^{\text{lmax}-2})^i a$.

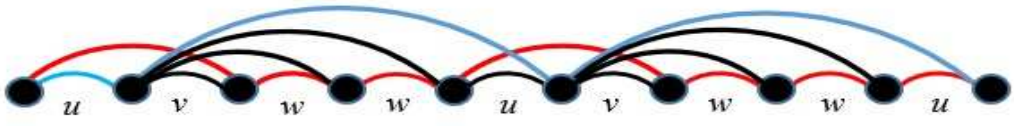

Figure 8.3. An example(lmax = 4): Longest Matching path (red) and Optimal path (blue).

Hence

$$R^{\infty}_{LM}(D) \geq \lim_{i\to\infty} \frac{i(\text{lmax}-1)\text{cmax}+\text{cmax}}{i\,\text{cmin}+\text{cmax}} = \frac{(\text{lmax}-1)\text{cmax}}{\text{cmin}}.$$

□

Corollary 9.2. *Let D be a prefix and code uniform dictionary. Then*

$$R^{\infty}_{LM}(D) \leq \text{lmax}-1,$$

and this bound can be attained.

Theorem 9.3. (Békési, Galambos, Pferschy and Woeginger, [15]). *Let D be a prefix and nonlengthening dictionary. Then*

$$R^{\infty}_{LM}(D) \leq \frac{\text{lmaxBt}}{\text{cmin}},$$

and this bound can be attained.

Proof. To derive the upper bound, we follow the proof of Theorem 9.1 and modify the weights of the edges in an appropriate way (we take into account that a code–word corresponding to a unit–length source–word cannot be longer than Bt).

To get a matching lower bound example, we use the dictionary given in the proof of Theorem 9.2 and modify the weights as follows:

source–word	u	v	w	uv	vw^j $_{j=1,\ldots,\text{lmax}-2}$	$vw^{\text{lmax}-2}u$
code–word	a	b	c	d	e_j	f
weight	cmin	Bt	Bt	2Bt	$(j+1)$Bt	cmin

Thereby we get

$$R^{\infty}_{LM}(D) \geq \lim_{i\to\infty} \frac{i\,\text{lmax}\,\text{Bt}+\text{cmin}}{(i+1)\text{cmin}} = \frac{\text{lmax}\,\text{Bt}}{\text{cmin}}.$$

□

By applying arguments analogous to those in the preceding proofs, it is easy to show the following:

Corollary 9.3. *Let D be a prefix, non-lengthening, and code uniform dictionary. Then*

$$R^{\infty}_{LM}(D) \leq \mathrm{lmax} - 1,$$

and this bound can be attained.

9.3 Differential Greedy algorithm

Algorithm 9.3.1. *The differential greedy (DG) algorithm always chooses the best possible local compression at the current position. This is done by calculating the differences $|w_i|\mathrm{Bt} - \|c_i\|$ for all matching dictionary source-words w_i and taking the source-word which maximizes the difference. Ties are broken arbitrary.*

The DG algorithm was introduced by Gonzales-Smith and Storer [61] and it was partially analysed from worst-case point of view in [75].

9.3.1 Prefix dictionaries

It is easy to see that the DG and LM behaves similarly for prefix algorithm. More exactly, the following statements are true.

Theorem 9.4. (Békési, Galambos, Pferschy and Woeginger, [15]). *Let D_1, D_2, D_3, and D_4 be prefix, prefix and code-uniform, prefix and non-lengthening, and prefix, non-lengthening and code-uniform dictionary, respectively. Then*

$$\begin{aligned} R^{\infty}_{DG}(D_1) &= (\mathrm{lmax} - 1)\tfrac{\mathrm{cmax}}{\mathrm{cmin}}, & R^{\infty}_{DG}(D_2) &= \mathrm{lmax} - 1, \\ R^{\infty}_{DG}(D_3) &= \tfrac{\mathrm{lmaxBt}}{\mathrm{cmin}}, & R^{\infty}_{DG}(D_4) &= \mathrm{lmax} - 1. \end{aligned}$$

Proof. To show that the bound from Theorem 9.1 can also be attained for a prefix dictionary D_1, the same dictionary introduced in the proof of Theorem 9.2 can be used. Again, we have $\mathrm{OPT}(S_i) = af^i$ and $DG(S_i) = (dc^{\mathrm{lmax}-2}i)a$, which yields the desired bound.

Along the same line one can easily prove the tightness of the bounds for the other dictionaries.

□

9.3.2 Suffix dictionaries

Obviously, for code uniform dictionaries DG and LM yield identical coding results. Hence, this property will not be considered in the following. In [75] for nonlenghtening, suffix dictionaries the following proposition was given.

Proposition 9.1 (Katajainen and Raita, [75]). *Let D be a nonlenghtening, suffix dictionary. Then*

$$R^{\infty}_{DG}(D) \leq \frac{\min\{\text{lmaxBt}, 2\text{cmax} - \text{Bt}\}}{\text{cmin}}.$$

Among other open problems they asked for the exact value of the worst–case ratio. In the next theorem we prove that this upper bound is indeed the best possible.

Theorem 9.5. (Békési and Galambos, [15]). *For infinitely many quadruples of positive integers* Bt, lmax, cmin *and* cmax *with* cmin $\leq$ Bt, cmin $\leq$ cmax *and* cmax $\leq$ lmax Bt, *there exists a nonlenghtening, suffix dictionary D such that*

$$R^{\infty}_{DG}(D) \geq \frac{\min\{\text{lmaxBt}, 2\text{cmax} - \text{Bt}\}}{\text{cmin}}.$$

Proof. Case I. If lmaxBt $\leq$ 2cmax $-$ Bt, then we consider the following dictionary:

source–word	u	w^j j=1,...,lmax−1	uw	$w^j u$ j=1,...,lmax−2	$w^{\text{lmax}-1}u$
code–word	a	b_j	c	d_j	e
weight	Bt	jBt	2Bt	$(j+1)$Bt	cmin

For $i \geq 1$, we define strings $S_i = u(w^{\text{lmax}-1}u)^i$ of length $n = i$ lmax $+ 1$. With this we get $DG(D, S_i) = (cb_{\text{lmax}-2})^i a$ and $\text{OPT}(D, S_i) = ae^i$. Calculating the worst–case ratio we have

$$R^{\infty}_{DG}(D) \geq \lim_{i\to\infty} \frac{i\,\text{lmaxBt} + \text{Bt}}{\text{Bt} + i\,\text{cmin}} = \frac{\text{lmaxBt}}{\text{cmin}}.$$

Case II: If lmaxBt $\geq$ 2cmax $-$ Bt $+ 1$, then we can suppose that cmax $= \alpha$Bt. This implies $2\alpha - 1 <$ lmax. We consider the following dictionary:

source word	u^j $j\in[1,\alpha-1]$	u^α	uw^j $j\in[1,\alpha-1]$	w^j $j\in[1,\alpha-2]$	$w^{\alpha-1}$	$w^{2(\alpha-1)}u$	$w^j u$ $j\in[1,\alpha-3]$ $j\neq\alpha-1$	$w^{\alpha-1}u$
code	a_j	b	c_j	d_j	e	f	g_j	h
weight	Bt	2Bt	$(j+1)$Bt	jBt	Y	cmin	cmin	cmax

where $Y =$ cmax$-$Bt. For $S_i = u^\alpha(w^{2(\alpha-1)}u)^i$, we get $OPT(D, S_i) = bf^i$ and $DG(D, S_i) = a_{\alpha-1}(c_{\alpha-1}e)^i a_1$.

There are some points where the choice of DG is not unique. As DG chooses $u^{\alpha-1}$, there also is another possible candidate u^α. The two differences computed by DG are $(\alpha-1)$Bt $-$ Bt resp. αBt $-$ 2Bt. W.l.o.g. we suppose that DG chooses $u^{\alpha-1}$. In a similar way the algorithms chooses $uw^{\alpha-1}$ instead of u and $w^{\alpha-1}$ instead of $w^{\alpha-1}u$. With this the worst–case ratio is given by

$$R^{\infty}_{DG}(D) \geq \lim_{i\to\infty} \frac{i(2\text{cmax} - \text{Bt}) + 2\,\text{Bt}}{i\,\text{cmin} + 2\text{Bt}} = \frac{2\text{cmax} - \text{Bt}}{\text{cmin}}.$$

□

Concerning the analysis of the DG algorithm for suffix dictionaries, the authors in [75] mention that "*the behaviour of the algorithm is inherently more difficult to analyse*". The only upper bounds known for this type of dictionary are the same as for general dictionaries; they are summarized in the following proposition. Deriving tighter bounds was posed as another open problem.

Proposition 9.2. (Katajainen and Raita [75]). *Let D be a suffix dictionary. Then*

$$R^{\infty}_{DG}(D) \leq \begin{cases} \dfrac{\mathrm{cmin} + (\mathrm{lmax}-1)\mathrm{cmax}}{\mathrm{cmin} + (\mathrm{lmax}-1)\mathrm{Bt}} & \textit{if } (\mathrm{lmax}-1)^2\,\mathrm{cmax}\,\mathrm{Bt} < \mathrm{cmin}^2 \textit{ and} \\ & \lfloor(\mathrm{cmax}-\mathrm{cmin})/\mathrm{Bt}\rfloor \geq \mathrm{lmax}-1 \\ \dfrac{(\mathrm{lmax}-1)\mathrm{cmax}}{\mathrm{cmin}} & \textit{otherwise.} \end{cases}$$

In the next theorem we give tight bounds for the asymptotic worst–case behaviour of the DG algorithm for suffix dictionaries.

Theorem 9.6. (Békési and Galambos, [15]). *Let D be a suffix dictionary with* $\mathrm{lmax} \geq 3$. *Then*

$$R^{\infty}_{DG}(D) \leq \begin{cases} \dfrac{2\mathrm{cmax}-\mathrm{Bt}}{\mathrm{cmin}} & \textit{if } \mathrm{cmax} \leq 3/2\,\mathrm{Bt} \\ \dfrac{(2\mathrm{cmax}+\mathrm{Bt})^2}{8\mathrm{Bt}\,\mathrm{cmin}} & \textit{if } 3/2\,\mathrm{Bt} < \mathrm{cmax} \\ & \quad \leq (\mathrm{lmax}-3/2)\mathrm{Bt} \\ \dfrac{(\mathrm{lmax}-1)(2\mathrm{cmax}-(\mathrm{lmax}-2)\mathrm{Bt})}{2\mathrm{cmin}} & \textit{if } (\mathrm{lmax}-3/2)\mathrm{Bt} < \mathrm{cmax} \end{cases}$$

All bounds can be attained.

Proof. Let $N = (V, A)$ be the network for a string S and the dictionary D as defined in the introduction of this chapter. Let v_i and v_j be two consecutive *cutvertices* of N. This implies that both of them lie on the optimal path and on the DG–path.

First, we will prove an upper bound on $R_{DG}(D)$. W.l.o.g. we may assume that the optimal path has only a single edge from v_i to v_j with (maximum) length lmax and (minimum) weight cmin. We introduce the following notation. We assume that the DG–path consists of the sequence $(v_i, v_{i_1}, v_{i_2}, \ldots, v_{i_{k+1}}, v_j)$. The length resp. weight of an edge $(v_{i_p}, v_{i_{p+1}})$, $1 \leq p \leq k$, is denoted by t_p resp. c_p.

Since the given dictionary has the suffix property, at vertex v_{i_p} the algorithm DG has to choose between the two edges $(v_{i_p}, v_{i_{p+1}})$ and (v_{i_p}, v_j). The latter has weight at most cmax. Since the DG algorithm uses the path through vertex $v_{i_{p+1}}$,

$$t_p\mathrm{Bt} - c_p \geq \left(\sum_{l=p}^{k} t_l + 1\right)\mathrm{Bt} - \mathrm{cmax}$$

must hold. Summing up over all p we get

$$\sum_{p=1}^{k} c_p \leq k\,\mathrm{cmax} + \mathrm{Bt}\sum_{p=1}^{k} t_p - \mathrm{Bt}\sum_{p=1}^{k}\sum_{l=p}^{k} t_l - k\,\mathrm{Bt}. \tag{124}$$

First, we will estimate the third term in the right hand side.

$$\sum_{p=1}^{k}\sum_{l=p}^{k} t_l = \sum_{p=1}^{k} p t_p. \tag{125}$$

We denote

$$T = \sum_{l=1}^{k} t_l + 1 \leq \mathrm{lmax} - 1.$$

It is easy to verify that the minimum of (125) is attained iff $t_1 = T - k$ and $t_p = 1,\ p = 2, \ldots, k$. Hence, we get

$$\min_{t_l} \sum_{p=1}^{k}\sum_{l=p}^{k} t_l = (T-k) + \sum_{i=2}^{k} i = T + \frac{(k+1)(k-2)}{2}$$

Substituting this result into (124) yields

$$\begin{aligned}\sum_{p=1}^{k} c_p &\leq \max_{1\leq k\leq \mathrm{lmax}-2}\left\{k\,\mathrm{cmax} - \mathrm{Bt}(k+1) - \mathrm{Bt}\frac{(k+1)(k-2)}{2}\right\} \\ &= \frac{1}{2}\max_{1\leq k\leq \mathrm{lmax}-2}\{2k\,\mathrm{cmax} - (k^2+k)\mathrm{Bt}\}. \end{aligned}\tag{126}$$

This expression is a concave function in k and becomes maximum for $k = \frac{2\mathrm{cmax}-\mathrm{Bt}}{2\mathrm{Bt}}$. Depending on the feasible range for k, we distinguish three cases:

1. If $\mathrm{cmax} \leq \frac{3}{2}\mathrm{Bt}$, the maximum is taken at $k = 1$.
2. If $\frac{3}{2}\mathrm{Bt} < \mathrm{cmax} \leq (\mathrm{lmax} - \frac{3}{2})\mathrm{Bt}$, the maximum is taken at $k = \frac{2\mathrm{cmax}-\mathrm{Bt}}{2\mathrm{Bt}}$.
3. If $(\mathrm{lmax} - \frac{3}{2})\mathrm{Bt} < \mathrm{cmax}$, the maximum is taken at $k = \mathrm{lmax} - 2$.

Substituting the corresponding values of k into the right hand side of (126) and assigning the weight cmax to the final edge from $v_{i_{k+1}}$ to v_j, we arrive at the desired results for the upper bounds.

In the second part of the proof we show that all the above upper bounds are indeed tight.

Case I: For $\mathrm{cmax} \leq \frac{3}{2}\mathrm{Bt}$, we consider the following dictionary.

source–word	u	w	uw	$w^j u$ $_{j=1,\ldots,\mathrm{lmax}-2}$	$w^{\mathrm{lmax}-1}u$	w^j $_{j=1,\ldots,\mathrm{lmax}-1}$
code–word	a	b	c	d_j	$d_{\mathrm{lmax}-1}$	e_j
weight	cmax − Bt	cmax	cmax	cmax	cmin	cmax − Bt

We compress the string $S_i = u(w^{\mathrm{lmax}-1}u)^i$ of length $n = i\ \mathrm{lmax} + 1$. Since $\mathrm{OPT}(D, S_i) = ad^i_{\mathrm{lmax}-1}$ and $DG(D, S_i) = (ce_{\mathrm{lmax}-2})^i a$,

$$R^\infty_{DG}(D) \geq \lim_{i\to\infty} \frac{i(2\mathrm{cmax} - \mathrm{Bt}) + \mathrm{cmax} - \mathrm{Bt}}{(\mathrm{cmax} - \mathrm{Bt}) + i\ \mathrm{cmin}} = \frac{2\,\mathrm{cmax} - \mathrm{Bt}}{\mathrm{cmin}}$$

Case II: If $\frac{3}{2}\mathrm{Bt} < \mathrm{cmax} \leq (\mathrm{lmax} - \frac{3}{2})\mathrm{Bt}$, we suppose that Bt is even, $\mathrm{cmin} \leq \frac{\mathrm{Bt}}{2}$ and that $\mathrm{cmax} = (2\alpha+1)\frac{\mathrm{Bt}}{2}$ for some integer α, $1 \leq \alpha \leq \mathrm{lmax} - 4$. We construct a dictionary on lmax letters. The letters of the alphabet are denoted by $u, v, w_1, \ldots, w_{\mathrm{lmax}-2}$. Let $k = \frac{2\mathrm{cmax}-\mathrm{Bt}}{2\mathrm{Bt}}$

source–word	u	uv	v^j $_{j=1,\ldots,\mathrm{lmax}-2-k}$	$v^{\mathrm{lmax}-1-k}$
code–word	a	b	c	d_0
weight	cmax	cmax	cmax	$\frac{1}{2}\mathrm{Bt}$

source–word	w_j $_{j=1,\ldots,k-1}$	$v^j w_1 \ldots w_{k-1}u$ $_{j=1,\ldots,\mathrm{lmax}-1-k}$	$v^{\mathrm{lmax}-k} w_1 \ldots w_{k-1}u$	$w_j \ldots w_{k-1}u$ $_{j=1,\ldots,\mathrm{lmax}-1}$
code–word	d_j	e_j	$e_{\mathrm{lmax}-k}$	f_j
weight	$(2j+1)\frac{\mathrm{Bt}}{2}$	cmax	cmin	cmax

and the source–string $S_i = u(v^{\mathrm{lmax}-k} w_1 \ldots w_{k-1}u)^i$. It is easy to check that $\mathrm{OPT}(D, S_i) = ae^i_{\mathrm{lmax}-k}$. Following the DG–path one can see that at the beginning the DG algorithm chooses the edge uv. At the endvertices of this edge there are several edges: One skips the string v and the others skip the strings v^j, $j = 1, \ldots, \mathrm{lmax} - 1 - k$. So DG chooses the edge $v^{\mathrm{lmax}-1-k}$. On the remaining part of the string (until it meets again a letter u) the DG algorithm has to decide between the edges $w_j \ldots w_{k-1}u$, $j = 1, \ldots, k-1$ and the single character edge w_j. In each vertex it resolves the arising tie by choosing w_j. In this way we get $DG(D, S_i) = (bd_0 d_1 \ldots d_{k-1})^i a$ which yields

$$\begin{aligned} R^\infty_{DG}(D) &\geq \lim_{i\to\infty} \frac{i\left(\mathrm{cmax} + \frac{\mathrm{Bt}}{2}\sum_{j=0}^{k-1}(2j+1)\right) + \mathrm{cmax}}{\mathrm{cmax} + i\ \mathrm{cmin}} \\ &= \lim_{i\to\infty} \frac{i\left(\mathrm{cmax} + \frac{\mathrm{Bt}}{2}\frac{(2\mathrm{cmax}-\mathrm{Bt})^2}{4\mathrm{Bt}^2}\right) + \mathrm{cmax}}{\mathrm{cmax} + i\ \mathrm{cmin}} = \frac{(2\mathrm{cmax} + \mathrm{Bt})^2}{8\mathrm{Bt}\ \mathrm{cmin}} \end{aligned}$$

Case III: Finally, for $(\mathrm{lmax} - \frac{3}{2})\mathrm{Bt} < \mathrm{cmax}$ we again consider a suffix dictionary on lmax letters called $u, w_1, \ldots, w_{\mathrm{lmax}-1}$.

source-word	u	w_j $_{j=1,\ldots,\mathrm{lmax}-1}$	$uw_{\mathrm{lmax}-1}$	$w_j \ldots w_1 u$ $_{j=\mathrm{lmax}-1,\ldots,1}$	$w_{\mathrm{lmax}-1} \ldots w_1 u$
code–word	a	b_j	c	d_j	e
weight	cmax	cmax $-\, j$ Bt	cmax	cmax	cmin

For the source-string $S_i = u(w_{\mathrm{lmax}-1} \ldots w_1 u)^i$ of length $n = i\ \mathrm{lmax} + 1$, we get $\mathrm{OPT}(D, S_i) = ae^i$ and $DG(D, S_i) = (cw_{\mathrm{lmax}-2}w_{\mathrm{lmax}-3} \ldots w_1)^i a$. Hence

$$\begin{aligned} R^{\infty}_{DG}(D) &\geq \lim_{i\to\infty} \frac{i\left(\mathrm{cmax} + \sum_{j=1}^{\mathrm{lmax}-2}(\mathrm{cmax} - j\,\mathrm{Bt})\right) + \mathrm{cmax}}{\mathrm{cmax} + i\,\mathrm{cmin}} \\ &= \lim_{i\to\infty} \frac{i(\mathrm{lmax}-1)(2\mathrm{cmax} - (\mathrm{lmax}-2)\mathrm{Bt}) + \mathrm{cmax}}{2\mathrm{cmax} + 2i\,\mathrm{cmin}} \\ &= \frac{(\mathrm{lmax}-1)(2\mathrm{cmax} - (\mathrm{lmax}-2)\mathrm{Bt})}{2\mathrm{cmin}} \end{aligned}$$

which completes the proof.

□

9.4 Fractional Greedy algorithm

In [75] Katajainen and Raita raised the question whether there exist other types of algorithms which behave "good enough" even in the worst case. Starting from the practical point of view that usually the *proportion* of the size of the compressed text relative to the source–text is considered we introduce the *fractional greedy* algorithm (FG).

Algorithm 9.4.1. *The fractional greedy algorithm takes the fractionally best possible local compression at any actual position in the source-string, i.e. if I is the set of indices of the outgoing arcs from the current node of the corresponding graph then an arc i_0 will be chosen so that*

$$i_0 = \min_{i\in I} \frac{\|c_i\|}{|w_i|\mathrm{Bt}} \,.$$

It is intuitively clear that in many cases this type of algorithm will perform better than the LM-algorithm, which does not care about code lengths at all, and the DG-algorithm, which tries to maximize the absolute gain in each encoding step, thereby missing chances of encoding smaller parts of the string with a good relative compaction rate. These facts together may yield a better total compression. This is illustrated by the following example.

Example: Let us consider the following nonlenghtening dictionary with cmax = 4, cmin = 1 and, as usual for ASCII encodings, Bt = 8.

source–word	u	v	uv	$v^{\mathrm{lmax}-1}u$
code–word	$1\emptyset$	$11\emptyset 1$	$11\emptyset\emptyset$	$\emptyset$
weight	2	4	4	1

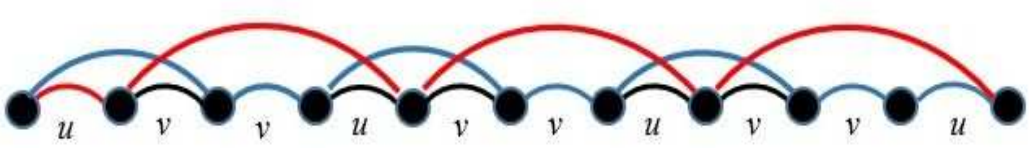

Figure 8.4. Longest Matching path (blue) and Fractional Greedy path (red) for $S_i = u(v^{\mathrm{lmax}-1}u)^i$ ($\mathrm{lmax} = 3,\ \ i = 3$).

Compressing the source–string $S_i = u(v^{\mathrm{lmax}-1}u)^i$ consisting of $8(\mathrm{lmax}\, i + 1)$ bits with the *LM*– or the *DG*–algorithm in both cases yields the code–string $(11\emptyset\emptyset(11\emptyset 1)^{\mathrm{lmax}-2})^i 1\emptyset$ with $4(\mathrm{lmax} - 1)i + 2$ bits. Applying the *FG*–algorithm to the same problem generates the code–string $1\emptyset(\emptyset)^i$ which is only $i + 2$ bits long.

Although this example is based on a very special dictionary it demonstrates the advantages of choosing the fractionally best compression.

Analysing FG in [14] we concluded that for general, code-uniform, nonlenghtening and prefix dictionaries and for all combinations of these properties the worst-case bounds for the LM-, the DG- and the FG-algorithm are identical.

9.4.1 Suffix Dictionaries

Throughout this section we will assume $\mathrm{lmax} \geq 3$ since every dictionary with $\mathrm{lmax} = 2$ is a suffix dictionary and the results of the previous section can be applied. The following theorem shows that for *suffix dictionaries* the fractional greedy algorithm behaves quite different from the other algorithms.

Theorem 9.7. (Békési, Galambos, Pferschy and Woeginger, [14]). *Let D be a suffix dictionary. Then*

$$R^{\infty}_{FG}(D) \leq \frac{\mathrm{cmax}(\ln(\mathrm{lmax} - 1) + 1)}{\mathrm{cmin}}$$

and there exists a suffix dictionary D_0 with

$$R^{\infty}_{FG}(D_0) > \frac{\mathrm{cmax}(\ln(\mathrm{lmax} - 1) + 1 - \ln 2)}{\mathrm{cmin}}.$$

Proof. Again, we consider those source–strings where the FG–path and an OPT–path are vertex disjoint and have common endvertices. Let us denote the FG–path between any two adjacent vertices v_a, v_{a+1} of the OPT–path by $z_1, z_2, \ldots, z_{k+1}$. We assume that there exists an arc on the FG–path connecting z_1 with a vertex preceding v_a and an arc leading from z_{k+1} past v_{a+1} (so called "skipping arcs"), except at the beginning and the end of the string. The following analysis can be performed for any a, i.e. for any arc of the OPT–path.

The length resp. weight of the arc (z_i, z_{i+1}) on the *FG*–path is denoted by t_i resp. c_i. Furthermore, the number of characters between z_{k+1} and v_{a+1} is denoted by β.

In the following two simple observations will be used:

$$\sum_{i=1}^{k} t_i \leq \mathrm{lmax} - \beta - 1 \tag{127}$$

$$\frac{c_i}{t_i} \leq \frac{\text{cmax}}{\sum_{s=i}^{k} t_s \; + \; \beta} \quad \forall\, i \tag{128}$$

To simplify the notation we introduce

$$T := \sum_{i=1}^{k} t_i \; + \; \beta \leq \text{lmax} - 1.$$

Considering observation (128) and summing up over all $i,\ 1 \leq i \leq k$, we get an upper bound for the sum of the weights:

$$\sum_{i=1}^{k} c_i \leq \text{cmax} \sum_{i=1}^{k} \frac{t_i}{T - \sum_{s=1}^{i-1} t_s} \leq \text{cmax} \, \ln T$$

The second inequality follows from Lemma 9.1 (see at the end of the proof).

Putting things together and taking into account that the weight of the arc emanating from z_{k+1} and "skipping" v_{a+1} is at most cmax yields

$$R_{FG}^{\infty}(D) \leq \frac{\sum_{i=1}^{k} c_i + \text{cmax}}{\text{cmin}} \leq \frac{\ln(\text{lmax} - 1)\text{cmax} + \text{cmax}}{\text{cmin}}.$$

The following construction gives dictionary D_0 and the claimed lower bound. Let us consider an alphabet with lmax letters $u, v, w_1, \ldots, w_{\text{lmax}-2}$ and the following dictionary D_0:

source–word	u	v	w_j $_{j=1,\ldots,\text{lmax}-2}$	uv	$w_j \ldots w_{\text{lmax}-2}u$ $_{j=1,\ldots,\text{lmax}-2}$	$vw_1 \ldots w_{\text{lmax}-2}u$
code–word	a	b	c_j	d	e_j	f
weight	cmax	cmax	$\frac{\text{cmax}}{\text{lmax}-j}$	cmax	cmax	cmin

Let $\text{cmax} = (\text{lmax}-1)!\,\text{cmin}$ and $S_i = u(vw_1 \ldots w_{\text{lmax}-2}u)^i$ be the string to be compressed with length $n = i\,\text{lmax} + 1$. It is easy to see that $\text{OPT}(D_0, S_i) = af^i$ and $FG(D_0, S_i) = (dc_1 \ldots c_{\text{lmax}-2})^i a$, if we suppose that the FG algorithm resolves all ties by choosing the arc corresponding to c_j. This example yields

$$\begin{aligned} R_{FG}^{\infty}(D_0) &\geq \lim_{i\to\infty} \frac{i\left(\text{cmax} + \sum_{j=1}^{\text{lmax}-2} \frac{\text{cmax}}{\text{lmax}-j}\right) + \text{cmax}}{\text{cmax} + i\,\text{cmin}} \\ &= \lim_{i\to\infty} \frac{i\,\text{cmax}\left(1 + \sum_{j=2}^{\text{lmax}-1} \frac{1}{j}\right) + \text{cmax}}{\text{cmax} + i\,\text{cmin}} \\ &\geq \frac{\text{cmax}\left(1 + \frac{1}{\text{lmax}-1} + \ln(\text{lmax} - 1) - \ln 2\right)}{\text{cmin}}, \end{aligned}$$

with the last inequality due to the fact that

$$\sum_{k=2}^{n} \frac{1}{k} \geq \ln(n+1) - \ln 2\,. \tag{129}$$

□

Lemma 9.1. *For integers* $s_1, \ldots, s_k, b$ *with* $s_i \geq 1$, $b \geq 1$ *and* $S = \sum_{i=1}^{k} s_i + b$

$$\sum_{i=1}^{k} \frac{s_i}{S - \sum_{j=1}^{i-1} s_j} \leq \sum_{i=1}^{S-b} \frac{1}{S-i+1} \leq \ln S$$

holds.

Proof. To prove the first inequality we show that the left hand side attains its maximum for $s_i = 1,\ i = 1, \ldots, k$.

If one of the s_i, namely s_m is greater or equal to 2 we can replace it by two values $s_{m_1} = s_m - 1$ and $s_{m_2} = 1$.

Thereby the left hand side is changed by

$$\frac{s_{m_1}}{S - \sum_{j=1}^{m-1} s_j} + \frac{s_{m_2}}{S - \sum_{j=1}^{m-1} s_j - s_{m_1}} - \frac{s_m}{S - \sum_{j=1}^{m-1} s_j}.$$

Setting $\bar{S} := S - \sum_{j=1}^{m-1} s_j$ the amount of change can be written as

$$\frac{1}{\bar{S} - s_m + 1} - \frac{1}{\bar{S}} > 0$$

which contradicts the assumption that the left hand side can be increased be setting $s_i > 1$ for some i.

The second inequality is shown by bounding a harmonic series.

$$\sum_{i=1}^{S-b} \frac{1}{S-i+1} = \sum_{i=b+1}^{S} \frac{1}{i} \leq \sum_{i=2}^{S} \frac{1}{i} \leq \ln S$$

□

If we compare the worst-case performance of the LM– and the DG–algorithms for suffix dictionaries with the FG algorithm, we can see that the LM has a better worst–case ratio than the FG which is in most relevant cases superior to the DG–algorithm. To be precise, if cmax $\leq 3/2\,$Bt R_{DG} is smaller than R_{FG}. If cmax is between $3/2\,$Bt and (lmax$-3/2$)Bt no strict dominance exists. (e.g. if cmax = (lmax $-$ 2)Bt FG is superior for lmax ≥ 7, if cmax $= 2\,$Bt DG is superior for all lmax ≥ 3.) For the last case (cmax $>$ (lmax$-3/2$)Bt) elementary calculations show that FG is always better than DG.

A more complicate analysis is necessary for the combination of the suffix and the nonlenghtening property.

Theorem 9.8. (Békési, Galambos, Pferschy and Woeginger, [14]). *Let D be a suffix and nonlenghtening dictionary. Then*

$$R_{FG}^{\infty}(D) \leq \frac{\min\{\text{lmax}\,\text{Bt}, \text{cmax}\left(\ln\left(\frac{\text{lmax}\,\text{Bt}}{\text{cmax}}\right)+3\right)-\text{Bt}\}}{\text{cmin}}.$$

Proof. The first expression in the minimum clause is a trivial upper bound for any nonlenghtening dictionary which dominates the second bound if cmax is close to lmax Bt.

To prove the latter expression we will use the same notation as in the proof of Theorem 9.7. Furthermore, we suppose that $\text{cmax} = \alpha\,\text{Bt}$ for some $\alpha > 0$. If the FG algorithm chooses at a vertex z_i an arc with length t_i and weight c_i this arc has to be "better" than the suffix arc leading directly to v_{a+1} which may have length cmax. This implies again

$$\frac{c_i}{t_i} \leq \frac{\text{cmax}}{\sum_{s=i}^{k} t_s + \beta}$$

whereas the nonlenghtening property is given by

$$\frac{c_i}{t_i} \leq \text{Bt}.$$

Let z_j denote the vertex with the smallest index $i \in \{1,\ldots,k\}$ for which $\text{Bt} \leq \text{cmax}/(\sum_{s=i}^{k} t_s + \beta)$ and let $T_i = \sum_{s=i}^{k} t_s + \beta$. It follows that $\text{Bt} \leq \alpha\,\text{Bt}/T_j$ and hence $T_j \leq \alpha$.

Summing up over all weights yields

$$\begin{aligned}
\sum_{i=1}^{k} c_i &\leq \left(\sum_{i=1}^{j-1} \frac{t_i}{T_i}\text{cmax} + \sum_{i=j}^{k} t_i \text{Bt}\right) \\
&= \text{cmax}\left(\sum_{i=1}^{j-1} \frac{T_i - T_{i+1}}{T_i} + \frac{1}{\alpha}(T_j - \beta)\right) \\
&\leq \text{cmax}\left((j-1) - \sum_{i=1}^{j-2} \frac{T_{i+1}}{T_i} - \frac{T_j}{T_{j-1}} + 1 - \frac{1}{\alpha}\right).
\end{aligned}$$

(In the last inequality, we used $\beta \geq 1$ and $T_j \leq \alpha$.) Bounding the arithmetic by the geometric mean we get

$$\sum_{i=1}^{j-2} \frac{T_{i+1}}{T_i} \geq (j-2)\sqrt[j-2]{\frac{T_{j-1}}{T_1}} \geq (j-2)\sqrt[j-2]{\frac{\alpha}{T_1}}. \tag{130}$$

Using $x - x\sqrt[x]{y} \leq \ln(1/y)$ for $x, y > 0$ leads to

$$\begin{aligned}
\sum_{i=1}^{k} c_i &\leq \text{cmax}\left((j-2) - (j-2)\sqrt[j-2]{\frac{\alpha}{T_1}} + 2 - \frac{1}{\alpha}\right) \\
&\leq \text{cmax}\left(\ln\left(\frac{T_1}{\alpha}\right) + 2 - \frac{1}{\alpha}\right) \qquad (131) \\
&\leq \text{cmax}\,\ln\left(\frac{\text{lmax}\,\text{Bt}}{\text{cmax}}\right) + 2\,\text{cmax} - \text{Bt}\,.
\end{aligned}$$

Adding the weight of the skipping arc and dividing by cmin yields the desired upper bound.

□

To illustrate that the upper bound shown above is almost as good as possible we now give a general lower bound for the suffix, nonlenghtening dictionary. A bound exactly matching the given upper bound can not be constructed because of the bounding techniques used in the above proof. Of course, the trivial upper bound $\mathrm{lmax}\,\mathrm{Bt}/\mathrm{cmin}$ can be exactly attained (see [75]).

Theorem 9.9. (Békési, Galambos, Pferschy and Woeginger, [14]). *There exists a suffix and nonlenghtening dictionary D_1 such that*

$$R_{FG}^{\infty}(D_1) \geq \frac{\mathrm{cmax}\left(\ln\left(\frac{\mathrm{lmax}\,\mathrm{Bt}}{\mathrm{cmax}}\right)+1\right)}{\mathrm{cmin}}.$$

Proof. We consider the same dictionary D_1 as in the proof of Theorem 9.7 with lmax letters $u, v, w_1, \ldots, w_{\mathrm{lmax}-2}$ but different weights. Let $\mathrm{cmax} = (\mathrm{lmax}-1)!\,\mathrm{cmin}$ and $\mathrm{cmax} = \alpha\,\mathrm{Bt}$ with $\alpha \in \mathbb{N}$.

source–word	u	v	w_j $_{j=1,\ldots,\mathrm{lmax}-2}$	uv	$w_j \ldots w_{\mathrm{lmax}-2}u$ $_{j=1,\ldots,\mathrm{lmax}-2}$	$vw_1 \ldots w_{\mathrm{lmax}-2}u$
code–word	a	b	c_j	d	e_j	f
weight	Bt	Bt	γ_j	$2\,\mathrm{Bt}$	$\gamma_j(\mathrm{lmax}-j)$	cmin

where

$$\gamma_j = \begin{cases} \frac{\mathrm{cmax}}{\mathrm{lmax}-j} & j = 1, \ldots, \mathrm{lmax}-\alpha \\ \mathrm{Bt} & j = \mathrm{lmax}-\alpha+1, \ldots, \mathrm{lmax}-2 \end{cases}$$

Let $S_i = u(vw_1 \ldots w_{\mathrm{lmax}-2}u)^i$ be the string to be compressed with length $n = i\,\mathrm{lmax}+1$. As in Theorem 9.7 we have $OPT(D_1, S_i) = af^i$ and $FG(D_1, S_i) = (dc_1 \ldots c_{\mathrm{lmax}-2})^i a$, if we suppose, as before, that the FG algorithm resolves all ties by choosing the arc corresponding to c_j. Using inequality (129) this example yields

$$\begin{aligned} R_{FG}^{\infty}(D_1) &\geq \lim_{i\to\infty} \frac{i\left(2\,\mathrm{Bt} + \sum_{j=1}^{\mathrm{lmax}-\alpha} \frac{\mathrm{cmax}}{\mathrm{lmax}-j} + \sum_{j=\mathrm{lmax}-\alpha+1}^{\mathrm{lmax}-2} \mathrm{Bt}\right) + \mathrm{Bt}}{\mathrm{Bt} + i\,\mathrm{cmin}} \\ &= \frac{2\,\mathrm{Bt} + \mathrm{cmax}\sum_{j=\alpha}^{\mathrm{lmax}-1} \frac{1}{j} + (\alpha-2)\mathrm{Bt}}{\mathrm{cmin}} \\ &\geq \frac{\mathrm{cmax}\ln\left(\frac{\mathrm{lmax}}{\alpha}\right) + \alpha\,\mathrm{Bt}}{\mathrm{cmin}} \\ &= \frac{\mathrm{cmax}\left(\ln\left(\frac{\mathrm{lmax}\,\mathrm{Bt}}{\mathrm{cmax}}\right)+1\right)}{\mathrm{cmin}}. \end{aligned}$$

□

If we compare FG with the other algorithms for suffix, and non-lengthening dictionaries, surprisingly, it turns out that both LM and DG yield a (slightly) better worst–case ratio than FG.

9.5 Iterated Longest Fragment algorithm

In [93] the *longest fragment first* (LFF) algorithm was introduced. It uses the following idea: we divide the string into equal-length segments. Then the longest word fragment - within the actual segment - matching a word from the dictionary is chosen and encoded. Ties are broken arbitrarily. The remaining part of the record is compressed with the LM algorithm.

In the same paper the authors presented an algorithm - we will refer to it as *iterated longest fragment (ILF)* - which improved the above idea: starting from the maximum match length, it goes down to length 1 and in each step it finds the maximum collection of non overlapping matches of the given length. These matchings will be selected for coding. Finally, in each iteration the algorithm eliminates all the matchings overlapping the selected ones. A formal description of the algorithm is the following:

Algorithm 5 : Iterated Longest Fragment Algorithm

---

1. **for** $l := L$ **downto** 1 **by** -1
2. Find a maximum collection C of non-overlapping matches of length l.
3. Select the elements of C for coding.
4. Eliminate the elements overlapping the matchings of C from the match lists.
5. **endfor**

However, they described the algorithm for parallel architecture. In this model a processor is assigned to each character of the input string and the processors can work parallel. This parallel algorithm can be implemented on "classical" one-processor architecture too. But in this form the algorithm is off-line, since we have to know the total source string to compute the list of match lengths. To avoid this problem Nagumo, Lu, and Watson in [83] defined the online version of the above process, which can be effective for conventional computer architectures. They used a lookahead (buffer) of length

$$\frac{1}{2}\,(\mathrm{lmax}-1)\,(\mathrm{lmax}-2)+\mathrm{lmax}.$$

Algorithm 9.5.1. *The basis of the sequential iterated longest fragment algorithm is the same as that of the Stauffer and Hirschberg algorithm, but they use a modified selection and elimination technique: choose the longest edge within the buffer (ties are broken by choosing the first one), eliminate all overlapping edges. The start of the buffer-window is moved to the first start-vertex of the first non-eliminated edge. Iterate this process till the end of the string. The proof of the correctness of the algorithm has been given in [83], but there was no result on its asymptotic behavior.*

Based on the paper [17] in this section we will prove that the ILF algorithm is 2/3 times better than LM for general dictionaries. Especially, for prefix and suffix dictionaries this constant decreases to 1/3. We also consider non–lengthening and uniform dictionaries, and similar tight bounds will be given for these cases as well.

9.5.1 General dictionaries

Since the ILF algorithm is identical with the LM algorithm if $\mathrm{lmax} = 2$, so we will consider only those cases where $\mathrm{lmax} \geq 3$. In this section we will first show that in these cases the ILF algorithm behaves better than the previously discussed algorithms in worst-case sense.

Theorem 9.10. (Békési and Galambos, [17]). *Let D be a general dictionary with* $\mathrm{lmax} \geq 3$. *Then*

$$R_{ILF}^{\infty}(D) = \frac{2\,(\mathrm{lmax}-1)}{3}\frac{\mathrm{cmax}}{\mathrm{cmin}}.$$

Proof. Let S be an arbitrary string over the source alphabet given by D and let $N = (V, A)$ be the suitable weighted, directed graph. Let v_i and v_j be two consecutive *common vertices* (i.e. vertices which belong to both the LM and the ILF–path) of N. This implies that the OPT-path and on the ILF–path have only these two common vertices in this interval. We order cmax as weight to each edge on the ILF-path and cmin to those of lying on the OPT-path. Consider the optimal path between v_i and v_j. Let $v_i^{'}$ be the end of the OPT-edge starting from v_i. Because the edge $\left(v_i, v_i^{'}\right)$ has not been chosen by ILF, there is a longer edge $\left(u_i, u_i^{'}\right)$ on the ILF–path, which overlaps the $\left(v_i, v_i^{'}\right)$. Let v_i^1 be the end of the last edge on the OPT-path, which overlaps $\left(u_i, u_i^{'}\right)$. We continue this idea starting from v_i^1. We stop when the OPT-path reaches v_j. This way we divide the optimal path between v_i and v_j into sub-paths. Let the dividing points be $v_i = v_i^0, v_i^1, v_i^2, ..., v_i^t = v_j$.

Consider now two arbitrary neighboring points from this list and denote them by v_i^q and v_i^{q+1}, $0 \leq q \leq t-1$. We analyse the OPT- and the ILF-path in this interval: we know that the longest edge in this interval can be found on the ILF-path, otherwise ILF would choose an optimal edge. Denote the length of the longest edge by f, $2 \leq f \leq \mathrm{lmax}$, and the total number of characters between v_i^q and v_i^{q+1} by s.

Since a longest edge on the OPT-path can not be longer than f, we get that the total number of edges on the OPT-path is at least $\left\lceil \frac{s}{f} \right\rceil$. It is also obvious that the ILF-path can not have more than $s - f + 1$ edges between v_i^q and v_i^{q+1}. By the definition of the ILF algorithm, the first edge on the OPT-path can not be longer than $f-1$ which yields the $2 \leq s \leq 3f - 3$ relation. Hence

$$R_{ILF}^{\infty}(D) \leq \frac{s - f + 1}{\left\lceil \frac{s}{f} \right\rceil}\frac{\mathrm{cmax}}{\mathrm{cmin}}. \tag{132}$$

The right hand side of this inequality attains its maximum when $f = \mathrm{lmax}$ and $s = 3f - 3 = 3\mathrm{lmax} - 3$. Substituting these values into (132) and using that $\left\lceil \frac{3\mathrm{lmax}-3}{\mathrm{lmax}} \right\rceil = 3$

if lmax > 3, we get the upper bound. For lmax $= 3$ the total number of edges on the OPT-path can not have less than 3 edges, because the length of the first edge is at most 2 and the total number of characters is 6.

To show that this bound is tight we construct the following D dictionary:

s.word	u	v	$v^{\text{lmax}-2}$	v^{lmax}	$u^{\text{lmax}-2}v$	$vu^{\text{lmax}-1}$
c.word	a	b	c	d	e	f
weight	cmax	cmax	cmin	cmax	cmin	cmin

where v^{lmax} denotes a string with lmax pieces of character v. Let us consider the following strings for $i \geq 1$: $S_i = \left(u^{\text{lmax}-2}v^{\text{lmax}}u^{\text{lmax}-1}\right)^i$. For S_i the optimal encoding is $\text{OPT}(D, S_i) = (ecf)^i$, and it is also easy to check that $ILF(D, S_i) = \left(a^{\text{lmax}-2}da^{\text{lmax}-1}\right)^i$. Therefore

$$R^{\infty}_{ILF}(D) \geq \lim_{i\to\infty} \frac{(2\,(\text{lmax}-1)\,\text{cmax})\,i}{(3\text{cmin})\,i} = \frac{2\,(\text{lmax}-1)}{3}\frac{\text{cmax}}{\text{cmin}},$$

and this completes the proof.

□

For the case of code-uniform dictionary we can get the desired result setting cmax $=$ cmin. It is not surprising that this constant factor $\frac{2}{3}$ remains valid for non-lengthening dictionaries too: on one-hand, if cmax $\leq$ Bt then every dictionary is non–lengthening and the statement of Theorem 9.10 remains valid. In the opposite the following theorem is true:

Theorem 9.11. (Békési and Galambos, [17]). *Let D be a non-lengthening general dictionary, where* Bt $\leq$ cmax. *Then*

$$R^{\infty}_{ILF}(D) = \frac{(2\text{lmax}-3)\ \text{Bt} + \text{cmax}}{3\text{cmin}}$$

Proof. Now we can apply the same idea as in the proof of Theorem 9.10, but while calculating the worst possible weight of the ILF-path we have to take into consideration the non-lengthening property. As a result we get:

$$R^{\infty}_{ILF}(D) \leq \frac{(s-f)\,\text{Bt} + \text{cmax}}{\left\lceil \frac{s}{f} \right\rceil \text{cmin}}. \tag{133}$$

Let us consider the right-hand side as a two variable function again. If we look for the maximum within the feasible values of the variables we get the required formula.
To prove that the bound is tight we need to modify the weights in the dictionary given in Theorem 9.10 so, that if cmax $> l \cdot$ Bt then instead of cmax we write $l \cdot$ Bt, where l is the length of the corresponding source-word. This can be done easily and it is left to the reader.

□

If the dictionary is code uniform and non-lengthening, the length of each code-word is the same and at most Bt. So we immediately get the following

Corollary 9.4. *Let D be a code uniform and non-lengthening general dictionary. Then*

$$R^{\infty}_{ILF}(D) = \frac{2\,(\mathrm{lmax}-1)}{3}.$$

9.5.2 Prefix dictionaries

It was conjectured in [75] that the "the coding result for prefix dictionaries can be weaker than the bounds derived for suffix dictionaries." The next theorems perfectly contradict the above conjecture: we will prove that for the ILF algorithm the prefix property improves the worst-case behavior almost twice relative to the general case, and the theorems also show that the prefix and suffix dictionaries have asymptotically almost the same performance.

Theorem 9.12. (Békési and Galambos, [17]). *Let D be a prefix general dictionary. Then*

$$R^{\infty}_{ILF}(D) = \frac{\mathrm{lmax}+1}{3}\frac{\mathrm{cmax}}{\mathrm{cmin}}.$$

Proof. To prove the upper bound we use the earlier idea. However, we have to count with the prefix property of the dictionary. Following the earlier notation we consider the edges between v^q and v^{q+1}. Denote the first OPT-edge - starting at the vertex v^q - by o and its length by $l(o)$. Similarly, we denote the longest ILF-edge and its length by p and $l(p)$.

We know that $l(o) \le l(p) - 1$. Because o and p overlaps each other, the length of the part of o which does not overlap p is at most $l(o) - 1$. Denote this part by $o^{'}$ and its length by $l(o^{'})$. Since the dictionary has the prefix property, ILF can code $o^{'}$ in one step. Moreover, it really does, since $o^{'}$ is the longest edge among those of the prefixes of o which do not overlap p, so it should be chosen by ILF. We know that $l(o^{'}) \le l(o)-1 \le l(p)-2$. So we get the following:

$$R^{\infty}_{ILF}(D) \le \frac{s - l(o^{'}) - l(p) + 2}{\left\lceil \frac{s}{l(p)} \right\rceil}\frac{\mathrm{cmax}}{\mathrm{cmin}}. \tag{134}$$

This function attains its maximum when $l(p) = \mathrm{lmax}$, $l(o^{'}) = l(p) - 2 = \mathrm{lmax} - 2$ and $s = 3l(p) - 3 = 3\mathrm{lmax} - 3$. Substituting this into (134) it yields an upper bound.
To prove that this bound can be reached we can construct the following prefix dictionary:

source–word	u^i	v^j	w	$u^{\mathrm{lmax}-2}v$	vw^k	$v^{\mathrm{lmax}-2}$
code-word	a_i	b_j	c	d	e_k	f
weight	cmax	cmax	cmax	cmin	cmin	cmin

where $i = 1, ..., \text{lmax}$, $j = 1, ..., \text{lmax}$, $j \neq \text{lmax} - 2$, $k = 1, ..., \text{lmax} - 1$.

Now consider the following $S_i = \left(u^{\text{lmax}-2}v^{\text{lmax}}w^{\text{lmax}-1}\right)^i$ strings for $i \geq 1$. The optimal encoding of S_i is $\text{OPT}(D, S_i) = (dfe_{\text{lmax}-1})^i$, and the ILF algorithm produces $ILF(D, S_i) = \left(a_{\text{lmax}-2}b_{\text{lmax}}c^{\text{lmax}-1}\right)^i$. So we get that

$$R^{\infty}_{ILF}(D) \geq \lim_{i\to\infty} \frac{((\text{lmax}+1)\,\text{cmax})\,i}{(3\text{cmin})\,i} = \frac{\text{lmax}+1}{3}\frac{\text{cmax}}{\text{cmin}}.$$

□

The above statement remains valid for those of non-lengthening dictionaries where $\text{cmax} \leq \text{Bt}$. Using the Theorems 9.11 and 9.12 one can easily derive the following

Corollary 9.5. *Let D be a prefix, non-lengthening general dictionary. Then*

$$R^{\infty}_{ILF}(D) = \begin{cases} \frac{(\text{lmax}-1)\text{Bt}+2\text{cmax}}{3\text{cmin}}, & \text{if } \text{Bt} < \text{cmax} \leq (\text{lmax}-2)\,\text{Bt} \\ \\ \frac{(2\text{lmax}-3)\ \text{Bt}+\text{cmax}}{3\text{cmin}}, & \text{if } (\text{lmax}-2)\,\text{Bt} < \text{cmax} \end{cases}$$

Similarly, we can combine the results of Corollary 9.5 and Theorems 9.11 and 9.12. Since for code-uniform and non-lengthening dictionaries the length of each code-word is the same, and at most Bt, the following holds.

Corollary 9.6. *Let D_1 be a prefix, code-uniform and D_2 be a prefix, code uniform and non-lengthening general dictionary. Then*

$$R^{\infty}_{ILF}(D_i) = \frac{\text{lmax}+1}{3}, \text{ for } i = 1, 2.$$

9.5.3 Suffix dictionaries

Finally, we present some results for suffix dictionaries. As earlier, we can suppose that $\text{lmax} \geq 3$.

Theorem 9.13. (Békési and Galambos, [17]). *Let D be a suffix general dictionary. Then*

$$R^{\infty}_{ILF}(D) = \frac{\text{lmax}}{3}\frac{\text{cmax}}{\text{cmin}}.$$

Proof. The analysis of the ILF algorithm for the case of the suffix dictionaries is very similar to the prefix ones. The only difference is that now we consider the *last* optimal edge o between v^q and v^{q+1}. Denote by o' the longest suffix of o which does not overlap p and its length by $l(o')$. In this case $l(o') \leq l(p) - 1$, and the remaining part of the proof is the same as in Theorem 9.12.
To prove that the upper bound is strict we have to construct the following suffix dictionary:

source–word	u	v^i	w^j	$u^k v$	$vw^{\text{lmax}-1}$	$v^{\text{lmax}-2}$
code-word	a	b_i	c_j	d_k	e	f
weight	cmax	cmax	cmax	cmin	cmin	cmin

where $i = 1, ..., \text{lmax}$, $i \neq \text{lmax} - 2$, $j = 1, ..., \text{lmax}$, and $k = 1, ..., \text{lmax} - 1$.
For any integer $i \geq 1$, let S_i be the string defined the same as in Theorem 9.12, i.e. $S_i = \left(u^{\text{lmax}-2} v^{\text{lmax}} w^{\text{lmax}-1}\right)^i$. Now, apply our algorithm to these strings. Then we get $ILF\,(D, S_i) = \left(a^{\text{lmax}-2} b_{\text{lmax}} c_{\text{lmax}-1}\right)^i$ and $OPT\,(D, S_i) = (d_{\text{lmax}-2} f e)^i$. So we get that

$$\begin{aligned} R^{\infty}_{ILF}(D) &\geq \lim_{i\to\infty} \frac{\|ILF(D,S_i)\|}{\|OPT(D,S_i)\|} = \lim_{i\to\infty} \frac{i\,(\text{lmax})\,\text{cmax}}{3\;i\cdot\text{cmin}} = \\ &= \frac{\text{lmax}}{3}\frac{\text{cmax}}{\text{cmin}}. \end{aligned}$$

which completes the proof.

□

For non-lengthening and code-uniform dictionaries we can use the ideas what we considered in the prefix cases again. Disregarding the subcase cmax $\leq$ Bt, one can get the following two corollaries:

Corollary 9.7. *Let D be a suffix, non-lengthening general dictionary. Then*

$$R^{\infty}_{ILF}(D) = \begin{cases} \frac{(\text{lmax}-2)\text{Bt}+2\text{cmax}}{3\text{cmin}}, & \text{if } \text{Bt} < \text{cmax} \leq (\text{lmax}-1)\,\text{Bt} \\ \\ \frac{(2\text{lmax}-3)\;\text{Bt}+\text{cmax}}{3\text{cmin}}, & \text{if } (\text{lmax}-1)\,\text{Bt} < \text{cmax} \end{cases}$$

Corollary 9.8. *Let D_1 be a code-uniform, suffix and D_2 be a code-uniform, suffix and non-lengthening general dictionary. Then*

$$R^{\infty}_{ILF}(D_i) = \frac{\text{lmax}}{3} \text{ for } i = 1, 2.$$

9.6 Concluding remarks

We analysed some old - earlier only experimentally investigated - algorithms from worst case point of view. It turned out that the ILF algorithm is recently the best among the ever analysed compression algorithms which use a static dictionary. There are some open questions which may be interesting for the future research:

- Construct better approximation algorithms that exploit the special properties of some dictionary types (e.g. it would be interesting to have a algorithm able to deal with suffix dictionaries more efficiently).
- Develop better algorithms by relaxing the on–line condition. E.g. the next 3lmax characters could be used to decide on the next replacement.

- The ILF algorithm in this form chooses the longest edge starting from a vertex, and it does not care about code length at all. This is not always the best solution. As it was pointed out in [14], if we choose the fractionally best possible local compression, we can expect a better result. Will we get a better performance using this algorithms iteratively?
- Is there any competitive data compression algorithm using static dictionary with better constant than $\frac{2}{3}$?
- Are there other reasonable properties for dictionaries which can be usable in the practice?
- To find out more about the practical behaviour of the proposed algorithms, a *probabilistic analysis* might be useful.
- Is it possible to extend the the analysis to dynamic (adaptive) dictionaries (i.e., find tight bounds for the Ziv-Lempel algorithm [104])?

References

[1] A. V. Aho and N. J. A. Sloane, Some doubly exponential sequences, *Fibonachi Quarterly,* **11**, 1973, 429–437.

[2] D. Ahr, J. Békési, G. Galambos, M. Oswald, G. Reinelt, An Exact Algorithm for Scheduling Identical Coupled Tasks, *ZOR,* **598**, 2004, 193–203.

[3] S. F. Assmann, D. S. Johnson, D. J. Kleitman, and J. Y.-T. Leung, On a dual version of the one-dimensional binpacking problem, *J. Algorithms,* **5,** 1984, 502–525.

[4] B. S. Baker, and E. G. Coffman, Jr., A tigth asymptotic boubd for Next Fit Decreasing bin packing, *SIAM J. Alg.,* **2,** 1981, 147–152.

[5] J. Balogh, J. Békési, G. Galambos, and G. Reinelt, Lower bound for the online bin packing problem with restricted repacking, *SIAM J. Compututing* **38(1)** 2008, 398–410.

[6] J. Balogh, J. Békési, and G. Galambos, New Lower Bounds for Certain Classes of Bin Packing Algorithms, *In Klaus Jansen and Roberto Solis-Oba, editors, Approximation and Online Algorithms - 8th International Workshop, WAOA 2010, Lecture Notes in Computer Science, Springer* **6534**, 2011, 25–36.

[7] J. Balogh, J. Békési, and G. Galambos, New Lower Bounds for Certain Classes of Bin Packing Algorithms, *Theoretical Comp. Sci.,* **440-441**, 2012, 1–13.

[8] J. Balogh, J. Békési, G. Galambos, M. C. Markot, Improved lower bounds for semi-online bin packing problems *Computing,* **4(1-2)**, 2009, 139–148.

[9] J. Balogh, J. Békési, G. Galambos, and G. Reinelt. Online bin packing with restricted repacking, J. of Comb. Opt., **27(1)**, 2014, 115–131, 2014.

[10] P. Baptiste, On minimizing the weighted number of late jobs in unit execution time open-shops, *EJOR,* **149(1)**, 2003, 344–354.

[11] Y. Bartal, H. Karloff, and Y. Rabani, A Better Lower Bound fo online Scheduling, *Inf. Proc. Letters,* **50**, 1994, 113–116.

[12] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra, New Algorithms for an Ancient Scheduling Problem, *Proc. of 24th ACM STOC*, 1992, 51–58.

[13] J. Békési, G. Galambos, M. Oswald, and G. Reinelt, Improved analysis of an algorithm for the coupled tasks problem with UET jobs, *Operation Res. Letters,* **37(2)**, 2009, 93–96.

[14] J. Békési, G. Galambos, U. Pferschy, G.J.Woeginger, The Fractional Greedy Algorithm for Data Compression, *Computing,* **56(1)**, 1996, 29-46.

[15] J. Békési, G. Galambos, U. Pferschy, G. Woeginger, Greedy Algorithms for online Data Compression, *Journal of Algorithms,* **25**, 1997, 274-289.

[16] J. Békési, G. Galambos, T. Raita, Longest Fragment First Algorithms for Data Compression. in F. Gianessi et al. (eds.) New Trends in Mathematical Programming, Kluwer Academic Publishing, 1998, 13-28.

[17] J. Békési, G. Galambos, Worst-case analysis of the Iterated Longest Fragment algorithm, *Information Processing Letters,* **97**, 2001, 147–153.

[18] J. Békési, G. Galambos, Data Compression: Theory and Techiques, *in: Database and Data Communication Network System Techniques and Applications. Ed. C.T. Leondes,* Academic Press, 2002, 233–276.

[19] D. J. Brown, A lower bound for online one-dimensional bin packing algorithms, Tech. Rept. R-864, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1979.

[20] P. Brucker, J. Jurisch, and M. Jurisch, Open shop problems with unit time operations, *ZOR,* **37**, 1993, 57–73.

[21] R. Chandrasekaran, B. Chen, G. Galambos, P. R. Narayanan, A. van Vliet, G. W. Woeginger. A note on an online scheduling heuristic with better worst case ratio than Graham's list scheduling", *SIAM J. on Computing,* **26,(3)**, 1997, 870–872.

[22] B. Chen, and A. van Vliet, On the online Scheduling Algorithm RLS, *Report 9325/A, Economic Institute, Erasmus University, Rotterdam,* 1993.

[23] K. L. Chung, A Course in probability Theory, *Academic Press, New York,* 1974.

[24] R. F. K. Chung, M. R. Garey, D. S. Johnson, On packing two-dimensional bins, *SIAM Alg.Discr.Meth.,* **3,** 1982, 66–76.

[25] E. G. Coffmann, M. R. Garey, and D. S. Johnson, Dynamic Bin Packing, *SIAM Journal on Computing,* **12(2),** 1983, 227–260.

[26] E. G. Coffman, G. Galambos, S. Martello, and D. Vigo, Bin Packing Approximation Algorithms: Combinatorial Analysis, *In "Handbook of Combinatorial Optimization" Supplement Volume. (Eds. D.-Z. Du and P.M. Pardalos Eds.). Kluwer Academic Publishers*, 2002, 151–208.

[27] E. G. Coffman, J. Csirik, G. Galambos, S.Martello, and D.Vigo, Bin Packing Approximation Algorithms: Survey and Classification, *In "Handbook of Combinatorial Optimization," New York, Springer Science+Business Media, Second Edition, (Eds. D.-Z. Du and P.M. Pardalos Eds.). Kluwer Academic Publishers*, 2013, 1455–531.

[28] P. R. Coppersmith, P. Raghavan, Multidimensional online bin-packig: algorithms and worst-case analysis, *Op. Res. Letters* **8**, 1989, 17–20.

[29] J. Csirik, G. Galambos, and Gy. Turán, Some Results on Bin Packing, *Proceedings of EURO VI.*, Vienna, 1983.

[30] J. Csirik, J. B. G. Frenk, A. M. Frieze, G.Galambos, and A.H.G.Rinnooy Kan, A probabilistic analysis of the next fit decreasing bin packing heuristic, *Op. Res. Letters*, 1986, 233–236.

[31] J. Csirik, G. Galambos. On the expected behaviour of the NF algorithm for the dual bin packing problem, *Acta Cybernetica,* **8(1)**, 1987, 5–9.

[32] J. Csirik, J .B. G. Frenk, and M. Labbe. Two dimensional rectangle packing: On line methods and results, *ARIDAM V, Rutgers University*, 1990.

[33] J. Csirik and G. Galambos, An O(n) bin packing algorithm for uniformly distributed data, *Computing,* **36**, 1986, 313–319.

[34] J. Csirik, and B. Imreh, On the worst-case performance of the NkF bin packing algorithm, *Acta Cybernetica,* **9**, 1989, 89–105.

[35] J. Csirik, and D.S. Johnson, Bounded space online bin packing: Best is better than First, *Proc. 2nd Ann. ACM-SIAM SODA,San Francisco*, 1991, 309–319.

[36] J. Csirik, J. B. G. Frenk, G.Galambos, and A.H.G.Rinnooy Kan, A probabilistic analysis of algorithms for dual bin packing problems, *J. of Algorithms,* **12**, 1991, 189–203.

[37] J. Csirik, and G. J. Woeginger, Online Packing and Covering Problems, *in: online Algorithms, Lecture Notes in Computer Science, eds. A. Fiat and G. Woeginger, Vol. 1442,* Berlin, 1998, 147–177.

[38] U. Faigle, W. Kern and Gy. Turán, On the performance of online algorithms for partition problems, *Acta Cybernet.,* **9**, 1989, 107–119.

[39] R. M. Fano, Transmission of Information, *Wiley, New York*, 1961.

[40] W. Feller, An Introduction to Probability Theory and its Applications, Vol. 1. *Wiley, New York,* 1970.

[41] W. Feller, An Introduction to Probability Theory and its Applications, Vol. 2. *Wiley, New York,* 1971.

[42] J. B. Frenk, On a pairing algorithm in binpacking, *Unpublished TR COSOR, Dept. of Math. and Comp. Sci., TU Eindhoven*, 1986.

[43] J. B. Frenk, and G. Galambos, Hybrid next-fit algorithm for the two-dimensional rectangle bin packing problem, *Computing,* **39**, 1987, 201–217.

[44] G. Galambos, A New Heuristic for the Classical Bin Packing Problem. Technical Report 82, Institut für Mathematik, Universität Augsburg, 1985.

[45] G. Galambos, Parametric Lower Bound for online Bin Packing, *SIAM J. on Alg. and Discr. Meth.* **7**, 1986, 362–367.

[46] G. Galambos, Notes on the Lee's harmonic fit algorithm, Computatorica, *Annales Univ. Sci. Budapest, Sect. Comp.* **9**, 1988, 121–126.

[47] G. Galambos, A 1.6 Lower Bound for the Two-Dimensional online Rectangle Bin Packing, *Acta Cybernetica,* **10**, 1991, 21–24.

[48] G. Galambos and J. B. G. Frenk, A simple proof of Liang's lower bound for online bin packing and the extension to the parametric case, *Discrete Applied Mathematics,* **41(2)**, 1993, 173–178.

[49] G. Galambos, H. Kellerer, and G. J. Woeginger, A Lower Bound for online Vector-packing Algorithms, *Acta Cybernetica,* **11,** 1993, 23–34.

[50] G. Galambos and A. van Vliet, *Lower bounds for 1,2 and 3-dimensional online bin packing algorithms, Computing,* **52**, 1994, 281–297.

[51] G. Galambos and G. J. Woeginger, Repacking helps in bounded space online bin packing,*Computing,* **49**, 1993, 329–338.

[52] G. Galambos and G. J. Woeginger, An online scheduling heuristic with better worst case ratio than Grahams list scheduling, *SIAM Journal on Computing,* **22**, 1993, 349–355.

[53] G. Galambos and G. J. Woeginger, Minimizing the Weighted Number of Late Jobs in UET Open Shops, *ZOR* **41**, 1995, 109–114.

[54] G. Galambos and G. J. Woeginger, On-line Bin Packing – A Restricted Survey, *ZOR* **42**, 1995, 25–45.

[55] G. Gambosi, A. Postiglione, and M. Talamo, Algorithms for the Relaxed online Bin Packing Model, *SIAM Journal on Computing,* **30(5),** 2000, 1532–1551.

[56] M. R. Garey and D. S. Johnson, Computers and Intractibility: A Guide to the Theory of NP-completeness, *W.H. Freeman & Co, San Francisco*, 1979.

[57] M. R. Garey, R. L. Graham, D. S.vJohnson, and A. C. C. Yao, Resource Constrained Scheduling as Generalized Bin Packing, *J. Comb. Th. Ser. A.,* **21**, 1976, 257–298.

[58] M. R. Garey, D. S. Johnson, A 71/60 theorem for bin packing, *J. Complex.,* **1**, 1985, 65106.

[59] S. W. Golomb, On certain nonlinear recurring sequences, *American Mathematical Monthly,* **70**, 19763, 403–405.

[60] T. Gonzales, and S. Sahni, Open Shop Scheduling to Minimize Finish Time, *Journal of ACM,* **23**, 1976, 665–679.

[61] M. E. Gonzalez–Smith and J. A. Storer, Parallel algorithms for data compression, *Journal of the ACM,***32**, 1985, 344–373.

[62] R. L. Graham, Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.*, **17**, 1969, 416–429.

[63] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey, *Annals of Discrete Mathematics,* **5,** 1979, 287–326.

[64] E. F. Grove, online Bin Packing with Lookahead, *in: Proc. of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms,* 1995, 430–436.

[65] G. Gutin, T. Jensen, and A. Yeo, Batched Bin Packing, *Discrete Optimization,* **2(1),** 2005, 71–82.

[66] W. Hoeffding, Probality ineqalities for sums of bounded random variables, *Journal of Am. Stat. Assoc.,* **58,** 1963, 13–30.

[67] D. A. Huffman, A method for the construction od minimum-redundancy codes. *Proc Institute of Electrical and Radio Engineers*, **40(9)**, 1952,1098-1101.

[68] Z. Ivkovič, and E. L. Lloyd, A Fundamental Restriction on Fully Dynamic Maintenance of Bin Packing, *Information Processing Letters,* **59(4),** 1996, 229–232.

[69] Z. Ivkovič, and E. L. Lloyd, Fully Dynamic Algorithms for Bin Packing, Being (Mostly) Myopic Helps, *SIAM Journal on Computing,* **28(2),** 1998, 574-611.

[70] D. S. Johnson, Fast algorithms for bin packing, *Computer Syst. Sciences,* **8**, 1974, 272–314.

[71] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, Worst case paerformance bounds for simple one-dimensional packing algorithms, *SIAM J. on Computing,* **3**, 1974, 299–325.

[72] D. R. Karger, S. J. Philips, and E. Torng, A Better Algorithm for an Ancient Scheduling Problem, *Proc. 5th Ann. ACM-SIAM SODA*, 1994, 132–140.

[73] N. Karmarkar, and R. M. Karp, An efficient approximation scheme for the one-dimensional bin packing problem, *in: Proc. of the 23rd Annual Symposium Foundation Computer Science,* 1982, 312–320.

[74] R. M. Karp, Lecture Notes, *Unpublished,* 1984.

[75] J. Katajainen and T. Raita, An analysis of the longest matching and the greedy aheuristic in text encoding, *Journal of the ACM,* **39**, 1992, 281–294.

[76] J. F. C. Kingman, Subadditive Processes, *Lect. Notes in Mathematics, Springer Verlag,* **539,** 1976, 168–222.

[77] C. C. Lee, and D. T. Lee, A simple online bin packing algorithm, *J. Assoc. Comput. Mach.,* **32**, 1985, 562–572.

[78] K. Li, K. H. Chang, A generalized Harmonic algorithm for online multi-dimensional bin packing. *Technical Report TR UH-Cs-90-2, University of Houston, January*, 1990.

[79] F. M. Liang, A Lower Bound for online Bin Packing, *Inf. Proc. Letters,* **10,** 1980, 76–79.

[80] L. Lovász, M. D. Plummer, Matching Theory, *Akadémiai Kiadó, North Holland,* 1986.

[81] G. S. Lueker, An average case analysis of bin packing with uniformly distributed item sizes, *Report 181, University of California, Irvine* 1982.

[82] M. Markót, Private communication.

[83] H. Nagumo, M. Lu, and K. Watson, On-line longest fragment first algorithm. *Information Processing Letters* **59** (1996) 91-96.

[84] H. L. Ong, M. J. Magazine, and T. S. Wee, Probabilistic analysis of bin packing heuristics, *Operation Research,* **32,** 1984, 983–998.

[85] A. J. Orman, and C. N. Potts, On the Complexity of Coupled-Task Scheduling, *Discrete Applied Mathematics,* **72**, 1997, 141–154.

[86] P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee, online Bin Packing in Linear Time, *J. of Algorithms,* **10**, 1989, 305–326.

[87] W. T. Rhee, and M. Talagrand, Martingale Inequalities, Interpolation and NP-Complete Problems, *Math. of Op.Res.,* **14(1),** 91–96.

[88] J. Riordan, Combinatorial Identitties, *Wiley, New York*, 1968.

[89] J. J. Risassen, G. G. Langdon, Arithmetic coding, *IBM J. Research and Development*, **23(2),** 1979,149-162.

[90] M. B. Richey, Improved bounds for Refined Harmonic Bin Packing, *unpublished manuscript*, 1990.

[91] A. H. G. Rinnooy Kan, Machine Scheduling Problems, *Martinus Nijhoff, The Hague*, 1976.

[92] H. E. Salzer, The approximation of numbers as sums of reciprocals, *Am. Math. Monthly,* **54**, 1947, 135–142.

[93] E. J. Schuegraf and H. S. Heaps, A comparison of algorithms for data base compression by use of fragments as language elements, *Inf. Stor. Ret.,* **10**, 1974, 309–319.

[94] S. Seiden, On the online Bin Packing Problem, *Journal of ACM,* **49,** 2002, 640–671.

[95] S. Seiden, R. van Stee, and L. Epstein, New Bounds for Variable-sized Online Bin Packing, *SIAM J. on Computing,* **32,** 2003, 455–469.

[96] S. Seiden, and Rob van Stee, New bounds for multi-dimensional packing. *Algorithmica,* **36(3),** 2003, 261293.

[97] R. D. Shapiro, Scheduling Coupled Tasks, *Naval Research Logistics Quarterly,* **20**, 1980, 489–498.

[98] J. Sylvester, On a Point in the Theory of Vulgar Fractions, *American Journal of Mathematics,* **3,** 1880, 332–335.

[99] F. W. de la Vega, and G. S. Lueker, Bin Packing can be Solved Within $1 + \varepsilon$ in Linear Time, *Combinatorica,* 1981, 349–355.

[100] A. van Vliet, An Improved Lower Bound for online Bin Packing Algorithms, *Inf. Proc. Letters,* **43** 1992, 274–284.

[101] A. van Vliet, Lower Bound and Upper Bounds for online Bin Packing and Scheduling Algorithms, PhD Thesis, *Tinbergen Institute Res. Ser.* no. 93.

[102] G. J. Woeginger, Improved space for bounded-space online bin packing, *Technical Report No. 187, TU Graz*, 1991.

[103] A. C. C. Yao, New Algorithms for Bin Packing, *J. Assoc. Comp. Mach.,* **27**, 1980, 207–227.

[104] J. Ziv, and A. Lempel, A universal algorithm for sequential data compression, *IEEE Trans. Inf. Theory* **23**, 1977, 337–343.