dc_2018_22

Contribution to a Class of Combinatorial Optimization Problems

DSc Dissertation

József Békési

University of Szeged
Szeged, 2022

Contents

Preface

Combinatorial optimization is an interdisciplinary field involving discrete mathematics and theoretical computer science. It is a rapidly growing area of research and has become very important from applications point of view during the last forty years. Combinatorial optimization models have been successfully applied in such diverse areas as economics, environmental sciences, public transportation, industrial production and many others. It can be also considered as a part of operations research, which is a more general discipline. Several combinatorial optimization problems are known, mainly originated from applications. Many of these problems are hard to solve, even using modern computers. In the last forty years, much research has focused on such problems. These can be the development and analysis of mathematical models or related algorithms.

In this dissertation we focus on four discrete optimization problems. These are the bin packing, machine scheduling, parallel routing and route planning problems. The first three chapter mainly deal with theoretical analysis of different algorithms, giving bounds on their performance measures. The fourth chapter is rather application oriented, when the aim was to develop an efficient algorithm running on a computer for a real life problem. Some technical parts of the proofs are given in two appendices.

The results given in the first three chapters of the dissertation are part of such papers, that were written with co-authors. However, the author's contribution is significant in the results listed here and the co-authors agree with that. In particular, such results that has been used by co-authors to obtain a scientific degree are not presented here. In the text the source of the result is indicated by a reference to the bibliography. In cases where a result needs to be mentioned for the interpretation of subsequent text, but it is not the result of the author or the work of the author is not significant in it, then the result is given without proof. The author is grateful to his co-authors and colleagues for their support.

Chapter 1

Bin Packing Problems

1.1 Introduction

In the classical one-dimensional bin packing problem a list L of $a_1, a_2, \ldots, a_n$ elements from the interval $(0, 1]$ is given, and we want to assign each item to a unit capacity bin. The objective is to minimize the number of bins. In case of *online problems* the input is not known completely in advance: items come one by one, and an *online algorithm* assigns them to a bin immediately, irrevocably. These online algorithms have been studied widely in the last decades. Their performance can be measured by the asymptotic performance ratio, which is defined as follows.

For a list L, let $\mathrm{OPT}(L)$ denote the number of bins in an optimal packing and let $A(L)$ denote the number of bins that algorithm A uses for packing L. If $R_A(N)$ denotes the supremum of the ratios $A(L)/\mathrm{OPT}(L)$ for all lists L with $\mathrm{OPT}(L) = N$, then the asymptotic competitive ratio (ACR) is defined as

$$R_A^\infty := \limsup_{N\to\infty} R_A(N).$$

The absolute measure $\sup_I \frac{A(I)}{OPT(I)}$ known as the absolute competitive ratio is also used sometimes to analyse bin packing algorithms. A standard method for proving an upper bound for the asymptotic competitive ratio for an algorithm A is to show the existence of a constant $C \geq 0$ independent of the input, such that for any input I, $A(I) \leq R \cdot OPT(I) + C$ and then the value of the asymptotic measure is at most R.

The classical bin packing problem was presented in the early 1970's [84, 54, 55, 56]. It was introduced as an offline problem, but many of the algorithms initially proposed for it were in fact online. Johnson [54, 55] defined and analysed the simple algorithm Next Fit (NF), which tries to pack the next item into the last bin that was used for packing, if such a bin exists, otherwise it opens a new bin for the item. The asymptotic competitive ratio of this algorithm is 2 [54, 55]. If we keep open every bin while packing the actual item we can define several algorithms. Among these the most known ones are *First Fit* and *Best Fit*. These algorithms put the element into the first bin it fits, or into that bin where the item fits the best, resp. For these algorithms $R_{FF}^\infty = R_{BF}^\infty = 17/10$, (see [54]). The so-called bounded space algorithms were introduced by Lee and Lee [61].

They defined the *Harmonic Fit* algorithm, which gets the elements online, but only limited number of bins are available to put the item. They proved that $R_{HF}^{\infty} = 1.69103\ldots$.

To relax the strict online condition several relaxations have been investigated. The so-called *lookahead algorithms* were considered by Grove [46]. A k-bounded lookahead algorithm delays to pack an element until the next $k-1$ items arrive. Grove proved that the $1.69103\ldots$ ACR is achievable by these algorithms. The presorted versions of NF, FF and BF are called NFD, FFD, BFD. They were studied as well. In these versions, items are still presented one by one, but they are sorted in a non-increasing order of sizes. For example, the approximation ratio of NFD is (approximately) 1.69103 [5] and that of FFD is $\frac{11}{9} \approx 1.22222$ [54]. These presorted variants are not online algorithms. All of the above results use asymptotic ratio. The optimal value of the absolute competitive ratio is known, it is $\frac{5}{3}$ [11]. A more detailed overview on the state of art of the bin packing algorithms can be found in [31].

1.2 Algorithm Advanced Harmonic

1.2.1 Preliminaries

Already in much of the previous work on online algorithms for bin packing, items were partitioned into classes by size. The simplest such classification is based on harmonic numbers, leading to the Harmonic algorithm of Lee and Lee [61]. In the harmonic algorithm of index k for an integer parameter $k \geq 2$, subset j is the intersection of the input and $(\frac{1}{j+1}, \frac{1}{j}]$ (where $1 \leq j \leq k-1$), and subset k of tiny items is the intersection of the input and $(0, \frac{1}{k}]$.

In these algorithms each subset is packed independently from other subsets using NF (so for $j \leq k-1$, any bin for subset j, except for possibly the last such bin, has j items, but for subset k, every bin except for the last bin for this subset has a total size of items above $\frac{k-1}{k}$), and for k growing to infinity, the resulting competitive ratio is approximately 1.69103 [61]. The drawback of those algorithms is that bins of subsets with small values of j can have relatively small total size. For example, a bin of subset 2 may have total size just above $\frac{2}{3}$ and a bin of subset 1 may have just one item of size just above $\frac{1}{2}$.

The first idea that comes to mind is to try to combine items of those two subsets into common bins. However, if items of class 2 arrive first, one cannot just pack them one per bin, as this immediately leads to a competitive ratio of 2 if no items of subset 1 arrive afterwards. Lee and Lee [61] proposed the following method to overcome this. A fixed fraction of items of subset 2 up to rounding errors is packed one per bin and the remaining items are packed in pairs. Thus, there are two kinds of bins for subset 2. The items we refer to here can only be sufficiently small items, so there is a threshold $\Delta \in (\frac{1}{2}, \frac{2}{3})$ such that items of sizes in $(\Delta, 1]$ and $(1-\Delta, \frac{1}{2}]$ are packed as before, while the algorithm tries to combine an item of size in $(\frac{1}{3}, 1-\Delta]$ with an item of size in $(\frac{1}{2}, \Delta]$. Even if those two items (one item of each one of the two intervals) are relatively small, still their

total size is above $\frac{5}{6} \approx 0.83333$. This last algorithm was called Refined-Harmonic, and its competitive ratio is smaller than 1.636. Ramanan et al. [70] designed two algorithms called Modified Harmonic and Modified Harmonic-2. The first one has a competitive ratio below 1.61562, and it allows to combine items of many subsets with items of sizes above $\frac{1}{2}$ and at most Δ. The second algorithm does not use only a single value of Δ, but splits the interval $(\frac{1}{2}, 1]$ further, allowing additional kinds of combinations. Its competitive ratio is approximately 1.612. For most subsets of items (where k is chosen to be in [20, 40] in all these algorithms), the last two algorithms pack some proportion of the items in groups of smaller sizes, to allow it to be combined with an item of size above $\frac{1}{2}$. Intuitively, for an illustrative example, assume that $\Delta = 0.6$, and consider the items of sizes in $(\frac{1}{11}, \frac{1}{10}]$. The items that are not packed into groups of ten items should be packed into groups of four items. For some of the subsets the proportion is zero, and they are still packed using NF. Ramanan et al. [70] showed, that the drawback of such algorithms is that no matter how many thresholds there are, there can be pairs of items that can be combined into the bins of an optimal solution while the algorithm does not allow this as it has fixed thresholds. Specifically, such algorithms allow to combine items of different intervals only in the case when the largest items of the two intervals fit together into a bin. This is the case with the next two harmonic type algorithms as well.

The next two papers, that of Richey [73] and that of Seiden [79] deal with a more complicated algorithm where many more subsets can be combined. The general structure is proposed in [73], and a full and corrected algorithm with its analysis is provided in [79]. For illustration, the items packed into smaller groups are called red and those packed into bins with maximum numbers of items of the subset are called blue. The goal is to combine as many bins with blue items with bins having red items as possible. Bins with red items always have small numbers of items, to allow them to be combined with relatively large items of sizes above $\frac{1}{2}$. The analysis is far from being simple, though it leads to a competitive ratio of at most 1.58889. The analysis of [79] is based on a complicated notion called weight system.

The carefully designed subset structure eliminates many worst-case examples, but the drawback mentioned above still remains. Recently, Heydrich and van Stee [74, 50] proposed to use a method introduced by Babel et. al [4], where some items are packed based on their exact size rather than by their subset. We adopt the approach of [74, 50] and we apply the methods of Babel et. al [4] on the largest items of sizes in $(\frac{1}{3}, 1]$. This approach means to combine items of sizes above $\frac{1}{2}$ with items of sizes in $(\frac{1}{3}, \frac{1}{2}]$ based on their exact sizes. Moreover, the approach involves combining pairs of items of subsets of sizes contained in $(\frac{1}{3}, \frac{1}{2}]$ while keeping the smallest items of such a subset to be matched with items of sizes above $\frac{1}{2}$ (and larger items of such a subset are used to be packed into pairs), as much as possible. Prior to the work of [74, 50], all previous algorithms for classic bin packing that partition items into classes assumed that an item of a certain subset has the maximum size when its possible packing was examined. This method simplifies the algorithm and its analysis, but it is not always a good

strategy as it excludes the option of combining items that can fit together into a bin in many cases. Heydrich and van Stee [74, 50] proved a competitive ratio of 1.5816.

In Subsection 1.2.2 we present the algorithm Advanced Harmonic, which has the current best asymptotic approximation ratio. This value is approximately 1.5783 (see [7]).

1.2.2 Algorithm AH

In algorithm AH, we do not just have red and blue items, but we potentially allow several kind of bins (that is, several and potentially a large number of colors for items of a given class), and furthermore items may change their colors once further items arrive. Due to these differences we will not use the illustration via colors of items in the description of our algorithm. For example, for the subset of items of sizes in $(\frac{1}{15}, \frac{1}{14}]$ we group items into subsets of 14 items or three items or just one item. We also use bins of the smallest items (our value of k is 43) where the total size of items is at most $\frac{17}{60}$, to allow them to be combined (among others) with items of sizes in $(\frac{1}{2}, \frac{43}{60}]$. These two features are possible due to the simple nature of our analysis, and they are crucial for getting the improved bound. Note that all items of sizes in $(0, \frac{1}{43}]$ are treated together by the algorithm and its analysis.

To handle the different subsets of items we use the concept of *containers*. A container is a set of items of one class (in the partition of potential inputs into items of similar sizes, called classes), and it can be complete if its planned number of items has already arrived or incomplete otherwise, but it is treated in the same way in both cases. Containers are of two types, either positive or negative, and a bin may contain at most one of each of them. The goal is to have as many bins as possible with both a positive and a negative container. Roughly speaking, positive containers have total sizes above $\frac{1}{2}$ and negative containers have total sizes of at most $\frac{1}{2}$. This last statement is imprecise as in most cases we consider volumes and not exact sizes, where volumes are based on the maximum sizes for the corresponding classes. There is one exception, which is containers with one item of size above $\frac{1}{3}$, where the exact size is taken into account (both by the algorithm and the analysis), and it is defined to be the volume. A positive container and a negative one fit together if their total volume does not exceed 1. Our positive and negative containers have some relation to the concepts used in [79].

Similarly to previous algorithms' definitions, AH has a sequence of boundary points that are used in its precise definition: $1 = t_0 > t_1 = \frac{1}{2} > t_2 > \cdots > t_b = \frac{1}{3} > \cdots > t_M > t_{M+1} = 0$. That is 1/2 and 1/3 are always boundary points, and there is no boundary point in $(1/2, 1)$.

For every j, all items of sizes in the interval $(t_j, t_{j-1}]$ are called items of class j. We say that a class of items (and every item of this class) is *huge* if $j = 1$, it is *large* if $1 < j \leq b$ (these are all items of sizes above 1/3 and at most 1/2), *small* if $b < j \leq M$, and *tiny* if this is the class of items of size at most t_M. That is, the last class is the one of tiny items, and in general the index of a class corresponds to the index j such that t_j is the minimal size of any item of the class.

Our algorithm will pack items into containers and pack containers into bins. As the algorithm is online, a container will be packed into a bin immediately when it is created, even though it may receive additional items later. In the last case, when we say that an item is packed into a container, this means that the bin containing the container receives that item. Any container will contain items of a single class, and at most two different containers can be combined (packed) into a bin. We provide additional details on combining two containers into a bin later. Every container of items that are not tiny has a cardinality associated with it, and this is the maximum number of items that it is supposed to receive.

Let $\gamma_j = \lfloor \frac{1}{t_{j-1}} \rfloor$ for $j \leq M$. For class j that is either large or small (but not huge or tiny, i.e., for values of j such that $2 \leq j \leq M$ holds), and for every i (where $1 \leq i \leq \gamma_j$) there is a non-negative parameter α_{ij}, where $0 \leq \alpha_{ij} \leq 1$. The value α_{ij} will denote the proportion of number of containers of cardinalities i of class j items among the number of containers of class j. The sum of proportions satisfies $\sum_i \alpha_{ij} = 1$ for all j. Such containers that will eventually receive i items of class j (unless the input terminate before this becomes possible) will be called *type i containers of class j*. That is, intuitively if we let x denote the number of containers for items of class j, we will have approximately $\alpha_{ij} \cdot x$ type i containers each of which having exactly i items of class j. For every j such that $2 \leq j \leq M$ and every i, we let $A_{i,j} = i \cdot t_{j-1}$. While the values α_{ij} are defined so far only for large and small classes, we consider one huge item as a type 1 container. Note that the values of α_{ij} are not proportions of items but of containers for class j, and the resulting proportions of items can be computed from them.

For classes of *large* items the notion of the cardinality of a container is slightly more delicate, and we will have exactly four possible types of containers. The first type is a *regular type 2* container (already) containing exactly two items of this class. The second type is a *declared type 2* container, where this type consists of containers for which the algorithm already decided to pack two items of this class in the container (so the planned cardinality of the container is 2) but so far only one such item was packed into the container. One of the few next arriving items of this class, if they exist, will be packed there, in which case the type will be changed into a regular type 2 container. The third is a *regular type 1* container, where such a container has one item of the class and cannot ever have an additional item of this class in future steps. Such a container will be combined with a container of another class that is packed into the same bin. The fourth and last type of a container of large items is a *temporary type 1* container. A container of this last type currently has one item of the class but sometimes it will get an additional item of this class in future steps and in this case its type will be changed to regular type 2 at that time. Its type can change to declared type 2 or regular type 1 as well, but in those cases it does not happen as a result of packing a new item to this container. Given a class of large items, the number of declared type 2 containers will be a constant throughout the execution of the algorithm. The numbers of containers of type 1 of both kinds and containers of regular type 2 can grow unbounded as the length of the input grows. The set of the union of containers of regular type 2 and declared type 2 is called type 2 containers, and

the set of the union of containers of regular type 1 and temporary type 1 is called type 1 containers. The parameters α_{1j} and α_{2j} of a large class j determine the approximate proportions of type 1 containers and type 2 containers, respectively.

For class $M+1$ of the tiny items, instead of the definitions above, there is a sequence of p possible upper bounds on the total sizes of items packed into containers of this class: $1 \geq A_{p,M+1} > A_{p-1,M+1} > \cdots > A_{1,M+1} \geq t_M$, and we let the positive parameters $\alpha_{i,M+1} > 0$ for $i = 1, \ldots, p$ denote the proportions of numbers of containers of class $M+1$ with items of total size in the interval $(A_{i,M+1} - t_M, A_{i,M+1}]$ (this is the planned total size of items for such a container). Such containers will be called *type i containers of class $M+1$*. The values of α_{ij} for all i, j are selected heuristically via a search procedure carried out by a computer program. Any such set of parameters give a different algorithm and our proof of the numerical value of the upper bound is for one specific set of parameters that we provide explicitly.

The *volume* of a container of type i of class j is defined as follows: If $i = 1$ and $1 \leq j \leq b$ (that is, for items of sizes above $1/3$), the volume of the container is the size of its (unique) item, and otherwise ($i = 2$ and $2 \leq j \leq b$ or $i \geq 1$ and $j > b$) it is $A_{i,j}$. That is, the volume is usually simply the largest total size that the container can occupy, but for a container that contains a single large or huge item, the volume is the *exact* size of the item. There is one exception where the bin already contains one large item and it is planned to contain another item of the same class. In most cases we would like the volume of a container to be known when it is created, which is possible for containers such that their planned contents are known (in the sense that for example type i containers of a non tiny class j are planned to contain i items finally). However, for large items such containers with a single item may be temporary type 1 containers, in which case there is still no planning of contents for them. In this last case, the volume of the container is the size of its unique item. However, the volume of such a container may change in the case the algorithm decides to pack another item of the same class into this container and transforms it into a type 2 container.. The volume of a declared type 2 container of class j is $A_{2,j} = 2 \cdot t_{j-1}$. Thus, the volume is based on its complete contents, no matter whether they are present already or not, as it is the case for classes of small or tiny items.

We say that a container is *negative* if its volume is at most $1/2$ and otherwise it is *positive*. Obviously, two positive containers cannot be packed into one bin. We will also not pack two or more negative containers into a bin together. Thus, a bin containing two containers will contain one positive container and one negative container, and no bin will contain more than two containers.

The rules for packing containers. The algorithm AH will pack items into containers and pack containers into bins according to the rules we will define. Recall that the packing of containers into bins will be such that every bin will have at most one positive container and at most one negative container. Obviously, a bin is non-empty if it has at least one container and at most two containers. We say that a non-empty bin is negative if it has a negative container and does not have a positive container, it is positive if it has a positive container and does not

have a negative container, and it is neutral if it has both a negative container and a positive container.

It is unknown whether a temporary type 1 container will eventually be positive or negative. Therefore, such a container will not be combined in a bin with another container as long as its type is not changed. Moreover, it is considered as a negative container until it changes its type (so its bin is negative as long as the container is of temporary type 1). Specifically, it remains a negative container if a positive container joins it and its bin becomes neutral. In this case it becomes a regular type 1 container and remains negative, and it becomes a positive container if its type changes to type 2. It can also happen that a temporary type 1 container will remain such till the termination of the input and the action of AH and its bin remains negative. It is important to note that the difference between regular type 1 containers of a large class and temporary type 1 containers of the same class is that each of the former containers is already packed into a bin with a positive container (of some class), while the latter are not packed with other containers. In fact, the corresponding items are placed into their own bins, one item per bin.

For every class j, we denote by n_j the number of containers of class j. Let n_{ij} denote the number of containers of type i of class j. We often consider the values n_j and n_{ij} just prior to the packing of a new item.

We say that two containers *fit together* if their total volume is at most 1. In what follows, when we refer to packing an item e - or packing a container containing e - into existing bins using Best Fit, we refer to packing e into the bin with a container of largest volume where the existing container and e fit together. For the original version of Best Fit, actual sizes are taken into account, but here we base this rule on volumes. Since for a container with a single large or huge item the volume is equal to the size of the item, if we select one such container then our action is equivalent to the standard application of Best Fit.

Packing rules of a new item. Next, we define the packing rules of the algorithm when a new item of class j arrives. The algorithm is defined for each step, based on the class of the new item.

A huge item. Recall that a huge item is immediately packed into a positive container containing only this item. Best Fit is applied on volumes, as explained above, to pack the created container into an existing negative bin, such that the two containers fit together. The only case where the new huge item joins a bin with a large item of some class j' is the case where the container of class j' is a temporary type 1 container, and in this case the type of this container of class j' is changed into regular type 1. If no bin can accommodate the container of the new item according to those packing rules, that is, for every negative bin, the total volume together with the new item is too big or there is no negative bin at all, then we use a new bin for the positive container of the new item and this new bin becomes a positive bin.

An item of a class of small or tiny items. For these classes we define the concept of an open container. Informally, an open container of class j can receive at least one additional item of class j. As a new container is introduced in order to pack an item, any container (of any type and class) already has at least one

item of the corresponding class. If $b < j \leq M$, an open type i container of class j is one where the total number of the items in the container is strictly smaller than i. Once such a container receives i items, it is closed. For $j = M + 1$, a type i container of this class will be open starting the time it is created and while the total size of items in it is positive and at most $A_{i,M+1} - t_M$. Once it reaches a total size above $A_{i,M+1} - t_M$, it will be closed. For all cases of packing a small or tiny item, a new container of some class will be used only if there is no open container of the same class, and thus, in particular, there will be at most one open container for each j (and the corresponding value of i will always be one such that $\alpha_{ij} > 0$).

When a new item of class j (such that $j > b$) arrives, if there is an open container of some type i of class j, then pack the item there. There can be at most one such container, so there are no ties in this case. Otherwise, open a new container for it (the details of the type are given below). After packing the new item into the container (and packing its container into a bin if it is a new container), close the container if necessary, based on its type and the rules above.

In the case that a new container is used for the item, we define the process of packing the item in more detail. Prior to packing the item, we define the type of the new open container. As the item is not packed yet, n_j is the number of containers of class j excluding the container opened for the new item. Find the minimum value of i such that $\alpha_{ij} > 0$ and so far there are at most $\lfloor \alpha_{ij} \cdot n_j \rfloor$ type i containers of class j (i.e., $n_{ij} \leq \lfloor \alpha_{ij} \cdot n_j \rfloor$, where the values n_{ij} do not include the new container which will be opened). Such an index i exists as otherwise there are more than n_j containers of class j. More precisely, since $\sum_i \alpha_{i,j} = 1$, there is always a value of i satisfying that $\alpha_{ij} > 0$ such that so far we opened at most $\lfloor \alpha_{i,j} \cdot n_j \rfloor$ type i containers of class j. Open a new type i container of class j containing the new item (increasing both n_j and n_{ij}). Observe that this opening of a new container defines its volume as well as whether it is a positive container or a negative container.

Next, we decide where to pack this new container. First consider the case where this container is a negative container. Then, if there is a positive bin, such that the new container fits into the bin according to its volume, then use that bin to pack the new container. This last case includes the possibility that the positive container is a type 2 container of a large class (regular or declared). If there are multiple options for choosing a bin, one of them is chosen arbitrarily.

Otherwise, (there is no positive bin where the new negative container can be added), the algorithm checks the option of using a bin with a temporary type 1 container of some class of large items. Assume that there is a negative bin B such that the following two conditions are satisfied. The first condition is that the bin B has a temporary type 1 container of class j' such that a positive container of class j' (with two items) will fit together with the new (negative) container. The second condition is that there are at most $\lfloor \alpha_{2j'} \cdot n_{j'} \rfloor - 1$ type 2 containers of class j' (before the packing of the new item is performed). Then, pack the new negative container into B, and define the container of class j' packed into B as a declared type 2 container. This last container of class j' will get one of the next items of

class j' that will arrive, which will happen before any new container is opened for any new class j' item, see below. If there are multiple options for choosing B, one of the classes of large items is chosen arbitrarily (among those that can be used), and a temporary type 1 container of this class with maximum volume is selected, i.e., we use Best Fit in this case. This last packing step is possible as a temporary type 1 container is never packed with another container into a bin. If another container joins it, its type is changed.

Otherwise, (if there is no suitable positive bin and no class of large items has a suitable temporary type 1 container that can be used under the required conditions), pack the new negative container into a new bin.

Finally, consider the case where the new container is a positive container. Then, if there is a negative bin whose container is not a temporary type 1 container, such that the new container fits together with it, then use such a bin to pack the new container. Otherwise, if there is a temporary type 1 container with one large item of a class j' where the new container fits, then pack the new positive container into this bin and define the container of class j' in this bin as a regular type 1 container. The class j' can be chosen arbitrarily if there are multiple options, and among the temporary type 1 containers of class j', one of maximum volume (out of those that can be used) is selected, i.e., once again we use Best Fit. Otherwise, pack the new positive container into a new bin.

A large item of a class j. If there is a declared type 2 container of class j, pack the item there as a second item and change it into a regular type 2 container, breaking ties arbitrarily. This packing rule is checked first, and we apply it whenever possible. We continue to the other cases in the situation where there is no such declared type 2 container.

If the number of type 2 containers equals $\lfloor \alpha_{2j} \cdot n_j \rfloor$ (that is, we should not increase the number of type 2 containers at this stage), then pack the new item into a new negative container. To pack the container into a bin, do as follows. If there is a positive bin where the new negative container fits, then use Best Fit to pack it as a regular type 1 container of class j (its volume is defined accordingly as the size of the new item) together with a positive container. This positive container is not of large items, as three large items cannot be packed into a bin together. Otherwise the new container is packed into a new bin, in which case it is defined to be a temporary type 1 container.

Otherwise (that is, the number of type 2 containers is strictly smaller than $\lfloor \alpha_{2j} \cdot n_j \rfloor$), we will increase the number of regular type 2 containers or the number of declared type 2 containers of this class in the current iteration as follows. If there is a negative bin B where a type 2 container of class j fits, then pack the item into a new declared type 2 container of class j and pack this container into this bin B. Otherwise, if there is a temporary type 1 container of class j, then we pack the new item using Best Fit (considering only temporary type 1 containers of class j, and selecting such a container of largest volume), and change the type of this container into a regular type 2 container. Otherwise (all containers of class j are either regular type 1 or regular type 2, we should increase the number of type 2 containers, and a new container with two items of this class cannot be packed

into an existing bin), we open a new declared type 2 container for the new item and open a new bin for this declared type 2 container and pack it there.

1.2.3 The method of the analysis

Let $a' = 1 - s_{\min}/2$ where $s_{\min}$ is the smallest item size in the examined input, and let a be the smallest volume of a positive container that is unmatched, if it exists. If no unmatched positive container exists, let $a = a'$. If $a > a'$, decrease the value of a to be a'. A simple property of the algorithm is that it tries to match a positive container and a negative container whenever possible. Thus every positive container of volume smaller than a is matched.

We define a *finite* set of scenarios according to the value of $1-a$. To do that we define a set of values V as follows. $V = \{A_{i,j}, 1 - A_{i,j} : j = 2, 3, \ldots, M+1, \alpha_{ij} > 0\} \cup \{t_1, t_2, \ldots, t_M, t_{M+1}\}$ and $V' = \{x \in V : x \leq 1/2\}$ (in particular, $\frac{1}{2} \in V'$). Note that the set V' contains (among other) all boundary points t_j (for all $j \geq 1$), even for values of j for which $\alpha_{1j} = 0$. The name of a scenario is an interval $(x, y]$ between consecutive values in V'. Using this partition, we ensure that if the scenario is $(x, y]$, then there is no $i \geq 2$ and class j such that $\alpha_{ij} > 0$ and the volume of a container of type i of class j is in (x, y) or in $(1 - y, 1 - x)$.

The first step for analyzing each scenario is to obtain a good weight function for the scenario, in the sense that the analysis will be as tight as possible and can be done using a computer assisted proof within a small running time. The weight function defines size based weights for values in $(0, 1]$. The goal is to define weights such that the cost of the algorithm is roughly the total weight of all input items. A weight function satisfying this requirement is called here *valid*, and if the target competitive ratio is R, the cost of an optimal solution is at least the total weight divided by R. This can be proved by showing that no bin can contain items of total weight above R. Then, for an input I, letting $w(I)$ denote its total weight, and as defined above, letting $OPT(I)$ the optimal cost for I, and $A(I)$ the number of bins used by A, we will have $A(I) \leq w(I) + c$, $OPT(I) \geq \frac{w(I)}{R}$, which shows that $A(I) \leq R \cdot OPT(I) + c$. This last argument is the standard argument for weight functions based analysis [54, 55, 56, 61, 70].

In order to define a suitable function, we solve a linear program which has only four variables w, u, v and R (in some cases it actually has only two variables w and R). More precisely, we will provide a feasible solution for this linear program that is very close to the optimal one but we only use its feasibility and do not prove that it is almost optimal. The weights of specific sizes will be based on the values w, u, v (or just on w, if the others are undefined), and on some of the parameters of the algorithm, namely on the α_{ij} values for the given class.

We define a quantity for each container called the *required weight* of the container, and its goal is to introduce a uniform value such that weights of items are defined based on these values, in order to satisfy all requirements. If the threshold class k (the class containing $1-a$), is a large class, we keep this quantity undefined for that class. For a positive container of volume at least a, the required weight of the container is 1. For a positive container of volume in the interval $(1/2, a)$, the

required weight of the container is denoted as w. This will be a decision variable of the linear program. The required weight of a negative container is 1 if its volume is larger than $1-a$ and otherwise its required weight is $1-w$. We ensure that the required weight of a container depends only on the index of the scenario $(x, y]$ and not the specific value of a in the interval $[1-y, 1-x)$. A related though simplified approach was used for online rectangle packing [49], where weight functions were also used instead of weight systems, and the variable w was set to $\frac{1}{2}$ for all cases.

The weight of a huge item is 1 if its size is at least a and w otherwise. The weight of an item of class $j \leq M$ such that either $j \neq k$ or $j > b$ is the ratio between the average required weight of a container of class j and the average number of items in a container of class j. The weight of a tiny item of size s is s times the ratio between the average required weight of a container of tiny items and the average lower bounds on the total size of items in a container of tiny items. The weight of items of a large class is as follows. An item of this class has weight u if its size is at most $1-a$ and otherwise a weight of v. We find linear inequalities on the variables u, v, w that ensure that the resulting weight function is valid. By solving a linear program we can find such values of u, v, w that minimize the corresponding competitive ratio that can be proven using this weight function. In this linear program the goal is to minimize R that is an upper bound on the total weight of items that can fit into one bin subject to the additional constraints on u, v, w ensuring that the resulting weight function is indeed valid.

In this way we get a table showing for each scenario the set of the values of u, v, w (or only w for scenarios where the threshold class is not large), that define the weight function used for the scenario. When we have these values, we can check the upper bound for that scenario by solving the corresponding knapsack problem using the weight function. This is done by a branch and bound procedure implemented for this purpose. The program code of the analysis can be found on the following link:
http://www.inf.u-szeged.hu/∼bekesi/kutatas/binp_onlinealg_workspace.zip.

1.2.4 The parameters and the analysis of the algorithm

We provide all required data for defining the algorithm and its analysis according to our method of analysis. Recall that we use exact values of parameters and exact calculations. In many cases we write an approximate value in the table in order to provide intuition, but these values were not used in our calculations.

The table in Appendix A contains the values α_{ij} for all j such that $2 \leq j \leq 5$ and $j \geq 166$. The values α_{ij} are only given for i such that $\alpha_{ij} \neq 0$.

For $6 \leq j \leq 165$, $\alpha_{1j} = \frac{22145926}{78181827} \approx 0.2832618$, $\alpha_{2j} = \frac{56035901}{78181827} \approx 0.71673816$. For these values of j, $t_j = 0.35 - \frac{j-5}{9600}$ and $t_{j-1} = 0.35 - \frac{j-6}{9600}$. For class 166, the right endpoint is $0.35 - \frac{160}{9600} = \frac{1}{3}$. There are many boundary points between $\frac{1}{3}$ and 0.35 as AH packs such items carefully, and we would like very similar pairs of such items to be packed together in one bin of the algorithm.

First, consider the case where the scenario is such that k is not a large class, so it is small or tiny. For all scenarios whose interval $(x, y]$ is contained in $(0, \frac{1}{6}]$, we

use $w = 0$, and find $R < \frac{82081796062891}{52009705144320} \approx 1.57820153$. The scenarios contained in $(\frac{3}{10}, \frac{1}{3}]$ also have common features. Every scenario has the form $(\frac{3}{10} + \frac{\ell-1}{4800}, \frac{3}{10} + \frac{\ell}{4800}]$ for $1 \leq \ell \leq 160$, $w = \frac{413913}{524288} \approx 0.7894764$, and

$$R < \frac{10060574276093395247}{6374352691333693440} \approx 1.57828956 \ .$$

The remaining cases are shown in the table in Appendix B. The values in the right column (UB) are the bounds we obtained on the total weight of a bin using the branch and bound procedure.

The case where k is a large class is also given in the table in Appendix B. Here, for each scenario, we present the values of u, v, and w, and the resulting upper bound on the total weight of a bin as computed by our branch and bound procedure. The program code of the analysis can be found on the following link: http://www.inf.u-szeged.hu/∼bekesi/kutatas/binp_onlinealg_workspace.zip.

Using the tables in Appendix B, we completed the proof of the following theorem.

Theorem 1.2.1 ([7]) *The competitive ratio of AH is at most* 1.57828956.

1.2.5 Conclusions

In this section we presented the design and analysis of the first algorithm of asymptotic competitive ratio strictly below 1.58. Specially, we provided an algorithm AH (Advanced Harmonic) whose ACR does not exceed 1.57829. Currently this is the algorithm with the best ACR for the online one-dimensional bin packing problem.

1.3 Lower Bound for 3-Batched Bin Packing

1.3.1 Notations and definitions

In this section we are dealing with a special relaxation of the online problem. The *batched bin packing problem* (BBPP) was defined by Gutin et al. (see [47]): the elements come in batches and one batch is available for packing in a given time. Each batch may contain different sizes of items, and any batch can be empty. If we have $K \geq 2$ batches then we speak about K-BBPP.

A *batched algorithm* packs the batch completely before the next batch arrives. It is clear that if each batch contains one element, then we have the classical online problem, and if only one batch is coming then the problem is the general (offline) one-dimensional BPP.

Let us consider an input sequence L, which is a *batched sequence*, i.e. $L = \{B_1, B_2, \ldots, B_K\}$, where B_j is a set of elements, $1 \leq j \leq K$. The set of all batched sequences exactly with K batches is denoted by $\mathcal{B}(K)$. Let A be a batched algorithm, then for BBPP the ACR is defined as follows.

$$R^{\infty}_{A,K} := \limsup_{N\to\infty} \left\{ \frac{A(L)}{\mathrm{OPT}(L)} : L \in \mathcal{B}(j), \quad j \leq K, \quad \mathrm{OPT}(L) = N \right\}.$$

In [47] the authors gave a 1.3871 . . . lower bound for the ACR of any on-line 2-BBP algorithm. It is clear that

$$R^{\infty}_{A,i} \leq R^{\infty}_{A,j} \leq \ldots, \quad \text{if} \quad 1 \leq i < j < \infty,$$

and such an algorithm A is interesting for which $R^{\infty}_{A,i} < 1.5783$ holds, where 1.5783 is the best known upper bound for the one-dimensional online bin packing algorithms [7]. In the paper [36] Dósa published an upper bound of $\frac{19}{12} = 1.58333\ldots$ for 2-BBPP.

In this section we investigate the case 3-BBPP, and we present a 1.51211 . . . lower bound for its ACR (see [6]). The structure of the section is the following. First we define a concatenated list of batches which we will use to prove a lower bound and we give tight bounds for the optimal packing of the batches of given instances. Subsection 1.3.3 investigates the possible strategies of the 3-batched BBP algorithms. To simplify our later discussions we make some reductions in subsection 1.3.4 on packing patterns of the algorithms under investigation. In subsection 1.3.5 we introduce an LP model to get the desired lower bound for $R^{\infty}_{A,3}$. Based on the LP model we give the lower bound for the ACR of any 3-BBP algorithm in subsection 1.3.6.

We determine this bound as a solution of a linear optimization problem, and we use theoretical analysis.

1.3.2 Construction for K = 3 batches

The construction is the following:

- The first batch B_1 contains $n_1 = 6jn$ pieces of small items – denoted by a_1 – with equal sizes. Let $j \geq 4$ be a fixed integer, then the size of each element in the batch is $s(a_1) = 1/6j = \varepsilon$.
- In the second batch one of the lists $B_{2,k}$ will be given to be packed. List $B_{2,k}$ contains $n_{2,k} = \frac{6j}{j-k}n$ pieces of $a_{2,k}$ items with size $s(a_{2,k}) = \frac{1}{3} + k\varepsilon - \frac{\varepsilon}{3} = \frac{1}{3} + \varepsilon\frac{3k-1}{3}$, and $1 \leq k \leq j-1$.
- The third batch $B_{3,k}$ follows the batch $B_{2,k}$. The number of elements in $B_{3,k}$ depends on the second batch. If we packed the list $B_{2,k}$ in the second step then $B_{3,k}$ contains $n_{3,k} = \frac{6j}{j-k}n$ pieces of a_3 items with sizes $s(a_3) = \frac{1}{2} + \frac{\varepsilon}{3}$.

To understand this structure it is important to see: the kth list among the second batches and the third batch with $n_{3,k}$ elements belonging to $B_{2,k}$ form an inseparable "couple". If the list $B_{2,k}$ has been chosen then $B_{3,k}$ contains $n_{3,k}$ items. To get a lower bound we investigate those three batches which consist of the concatenated lists $(B_1, B_{2,1}, B_{3,1})$, $(B_1, B_{2,2}, B_{3,2}), \ldots, (B_1, B_{2,j-1}, B_{3,j-1})$. In fact, we will prove later, that we do not need to take into account all of these batches. It is enough to consider only a part of them. The following lemmas are true.

Lemma 1.3.1 ([6]) *$OPT(B_1) = n$.*

Proof In the first batch there are items with equal sizes. We get optimal packing, if any bin contains as many items as possible. The maximum number of a_1 items in a bin is $1/\varepsilon = 6j$. Thus $\mathrm{OPT}(B_1) = n$. □

Lemma 1.3.2 ([6]) *For any k, $1 \leq k \leq j-1$, $OPT(B_1, B_{2,k}) = \frac{3j}{j-k}n$.*

Proof Since $\frac{1}{3} < s(a_{2,k}) \leq \frac{1}{2}$, so packing only the items of the second batch we need at least $\frac{n_{2,k}}{2} = \frac{3j}{j-k}n$ bins. Therefore

$$\mathrm{OPT}(B_1, B_{2,k}) \geq \frac{3j}{j-k}n. \tag{1.1}$$

Let us pack the elements of $B_{2,k}$ into bins, 2 in each bin. Then every bin will have

$$1 - 2s(a_{2,k}) = \frac{1}{3} - 2\varepsilon\frac{3k-1}{3}$$

empty space. Since $2(j-k)\varepsilon = \frac{1}{3} - \frac{k}{3j}$ and

$$2\varepsilon\frac{3k-1}{3} = \frac{1}{3j}\frac{3k-1}{3} < \frac{k}{3j}.$$

Therefore we can pack $2(j-k)$ elements from B_1 into each bin which has 2 pieces from the batch $B_{2,k}$. Thus, into the $\frac{3j}{j-k}n$ bins we can put all the $2(j-k)\frac{3j}{j-k}n = 6jn$ items from the batch B_1. Therefore

$$\mathrm{OPT}(B_1, B_{2,k}) \leq \frac{3j}{j-k}n. \tag{1.2}$$

So, (1.1) and (1.2) give together the statement of the lemma. □

Lemma 1.3.3 ([6]) *For any k, $1 \leq k \leq j-1$, $OPT(B_1, B_{2,k}, B_{3,k}) = \frac{6j}{(j-k)}n$.*

Proof The proof is similar to the one in the previous lemma. Any item from the third batch needs its own bin. Therefore

$$\mathrm{OPT}(B_1, B_{2,k}, B_{3,k}) \geq \frac{6j}{j-k}n. \tag{1.3}$$

Into each bin we can pack a further item from the second batch. As $n_{2,k} = n_{3,k}$, all items from the second batch can be packed into these bins. Now in each bin $1 - a_{2,k} - a_{3,k} = 1/6 - k\varepsilon$ empty space remained. Since $(j-k)\varepsilon = \frac{1}{6} - \frac{k}{6j}$, $j-k$ items will fit here from the batch B_1. So, all $(j-k)\frac{6j}{j-k}n = 6jn$ items from the batch B_1 can be packed into the existing bins. Therefore

$$\mathrm{OPT}(B_1, B_{2,k}, B_{3,k}) \leq \frac{6j}{j-k}n. \tag{1.4}$$

From (1.3) and (1.4) the statement of the lemma follows. □

Let us denote by $(i_1, i_2, i_3)_1$, $(i_1, i_2, i_3)_{2,k}$, and $(i_1, i_2, i_3)_{3,k}$ the type of a bin after packing the batch B_1, $B_{2,k}$, and $B_{3,k}$, respectively. If a bin is in status $(i_1, i_2, i_3)_{2,k}$ then we do not started to pack the elements of $B_{3,k}$, therefore it is obvious that for any $(i_1, i_2, i_3)_{2,k}$ of bin $i_3 = 0$. We call a triplet (i_1, i_2, i_3) as *valid packing pattern* (or *feasible packing-pattern*) if

$$i_1 s(a_1) + i_2 s(a_{2,k}) + i_3 s(a_{3,k}) \leq 1.$$

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

$$V_t = \{v \in V \mid \ i_t > 0 \text{ and } i_r = 0, \text{ for } r < t\}, \ t = 1, 2, 3.$$

Clearly, $V_t \cap V_r = \emptyset$ if $t \neq r$.

Let us denote by $V_{2,k}$ the set of all opened bins after having packed the elements of $B_{2,k}$. There will be bins which belong to V_1, and some of them will belong to V_2. Then

$$V_{2,k} = V_1 \cup V_2.$$

We can define similarly $V_{3,k}$, and

$$V_{3,k} = V_1 \cup V_2 \cup V_3.$$

It is important to emphasize that if we open a bin while we pack the elements of the batch B_1, then its type is $(i_1, i_2, i_3)_1$ and it belongs to V_1 i.e. $(i_1, i_2, i_3)_1 \in V_1$. If we put items from $B_{2,k}$ into this bin, then $i_2 > 0$, therefore its type will be changed to $(i_1, i_2, i_3)_{2,k}$, and it will belong to $V_{2,k} \supset V_1$ i.e. its type will be $(i_1, i_2, i_3)_{2,k}$, but this bin henceforward will belong to V_1.

Let $x^k_{i_1,i_2,i_3}$ denote the number of bins which contain exactly i_1, i_2, i_3 pieces from the lists $B_1, B_{2,k}, B_{3,k}$, respectively.

To understand the following proofs we remind the reader that among the batches any batch may be empty, so it is possible that after the first (second) batch the 3-batched algorithm will not receive any element.

1.3.3 Less efficient algorithms and strategies for any 3-BBP algorithm

Lemma 1.3.4 ([6]) *Let A be a batched algorithm. If A packs the elements of B_1 and $B_{2,k}$ batches, and it does not open a new bin while packing the elements of $B_{2,k}$ then $R^\infty_{A,3} \geq 3$.*

Lemma 1.3.5 ([6]) *Let A be a batched algorithm. If A packs the elements of B_1, $B_{2,k}$ and $B_{3,k}$ batches, and it does not open a new bin while packing the elements of $B_{3,k}$ then $R^\infty_{A,3} \geq 2$.*

Because of Lemmas 1.3.4 and 1.3.5 it is enough to investigate only those batched algorithms for which $R^\infty_{A,3} < 2$. On the other hand, from theoretical point of view, we are interested in those algorithms, for which $R^\infty_{A,3} \leq 1.6$.

Let us suppose that the algorithm put into a bin i_1 pieces of a_1 items ($1 \leq i_1 \leq 6j$). Then a bin will be filled at the level $i_1 a_1$. According to the value of i_1, the bins can be classified as follows.

- Class 1.1. $1 - i_1 a_1 < a_{2,k}$.
- Class 1.2. $a_{2,k} \leq 1 - i_1 a_1 < a_{3,k}$.
- Class 1.3. $a_{3,k} \leq 1 - i_1 a_1 < 2a_{2,k}$.
- Class 1.4. $2a_{2,k} \leq 1 - i_1 a_1 < a_{2,k} + a_{3,k}$.
- Class 1.5. $a_{2,k} + a_{3,k} \leq 1 - i_1 a_1 \leq 1$.

Packing the items of the second batch then – according to its strategy – it will pack the items of $B_{2,k}$ as follows.

- Case 2.1. The algorithm can not pack any items into the bins belonging to the Class 1.1.
- Case 2.2. If a bin belongs to the Class 1.2 then this bin can receive only one item from the batch $B_{2,k}$ (and nothing else), so the algorithm will create bins of type $(i_1, 1, 0)_{2,k} \in V_1 \subset V_{2,k}$.
- Case 2.3. If the algorithm tries to put an $a_{2,k}$ item into bins in Class 1.3 then it has several possibilities.
- Case 2.3.1. One item from $B_{2,k}$ will be put into some of these bins, getting $(i_1, 1, 0)_{2,k} \in V_1 \subset V_{2,k}$ type of bins.
- Case 2.3.2. It "reserves", and – waiting an item from the batch $B_{3,k}$ – it does not pack any items in some of such type of bins. This step results in bins of type $(i_1, 0, 0)_{2,k} \in V_1 \subset V_{2,k}$.
- Case 2.4. If the algorithm tries to put $a_{2,k}$ items into bins in the class Class 1.4 then it has again two possibilities.
- Case 2.4.1. It packs at most two items from the batch $B_{2,k}$. If the algorithm packs only a single item into these bins then it does not result in any advance for our batches, so we can suppose that the "good" algorithms pack always two items in this step. Now, we get bins of type $(i_1, 2, 0)_{2,k} \in V_1 \subset V_{2,k}$.
- Case 2.4.2. It holds the space in store and does not pack any item into some of these bins, getting bins of type $(i_1, 0, 0)_{2,k} \in V_1 \subset V_{2,k}$.
- Case 2.5. If – following its strategy – the algorithm puts an $a_{2,k}$ item into bins belong to Class 1.5 then it can do it as follows.
- Case 2.5.1. It puts two items from $B_{2,k}$ into such a bins, resulting bins of type $(i_1, 2, 0)_{2,k} \in V_1 \subset V_{2,k}$.
- Case 2.5.2. It "put this bin on ice" and places only one item into the bin, getting bins $(i_1, 1, 0)_{2,k} \in V_1 \subset V_{2,k}$. Let we remark that those algorithms which do not put any items from $B_{2,k}$ into these bins behave worse while they pack our batches, so we ignore them from our analysis.

- Case 2.6. If the algorithm can not place items from the batch $B_{2,k}$ into opened bins and there remain items unplaced, then the algorithm either put a single item into a bin, so getting bins of type $(0,1,0)_{2,k} \in V_2 \subset V_{2,k}$, or put two items into it, so getting bins of type $(0,2,0)_{2,k} \in V_2 \subset V_{2,k}$.

Finally, the algorithm tries to pack the items of $B_{3,k}$ as follows.

- Case 3.1. If the algorithms follows the strategy given in Case 2.3.2 then in the $(i_1,0,0)_{2,k}$ bin an item from $B_{3,k}$, can be packed, so it produces $(i_1,0,1)_{3,k} \in V_1 \subset V_{3,k}$. type bins.
- Case 3.2. If some bins of type $(i_1,0,0)_{2,k}$ have bin produced in Case 2.4.2, then there is place for an item from the batch $B_{3,k}$, so $(i_1,0,1)_{3,k} \in V_1 \subset V_{3,k}$ type bins arise.
- Case 3.3. At least one bin has been produced in step Case 2.5.2. Then the algorithm can put one item from $B_{3,k}$ into these bins. So, in this step we will get $(i_1,1,1)_{3,k} \in V_1 \subset V_{3,k}$ type bins.
- Case 3.4. If the algorithm put only one single item into an opened bin in Case 2.6.1, then there is enough place for one item from the batch $B_{3,k}$, and the results are bins of type $(0,1,1)_{3,k} \in V_2 \subset V_{3,k}$.
- Case 3.5. Finally, if unplaced items remain then the algorithm opens new bins for each of them, getting bins with type $(0,0,1)_{3,k} \in V_3 \subset V_{3,k}$.

1.3.4 Reduction on packing patterns

Lemma 1.3.6 ([6]) *Let A be a batched algorithm for which $R^{\infty}_{A,3} \leq 1.6$. After packing the elements of $B_{2,k}$ by A, at least one $(0,2,0)_{2,k}$ type bin must be created.*

Lemma 1.3.7 ([6]) *If A is a batched algorithm with $R^{\infty}_{A,3} \leq 1.6$, then always exists such an algorithm $A^{'}$ which does not use more bins than A does, and it does not create bins with types neither $(i_1,1,1)_{3,k} \in V_1$ nor $(i_1,0,1)_{3,k} \in V_1$, where $i_1 > 0$.*

Lemma 1.3.8 ([6]) *Let A be a batched algorithm with $R^{\infty}_{A,3} < 2$. If A packs the items of the batches $B_1, B_{2,k}, B_{3,k}$ in such a way that at least one $(0,1,0)_{2,k}$ type bin will be produced after finishing the packing of all the batches, then always exists an A' online batched algorithm which does not create $(0,1,0)_{2,k}$ type bins and uses less bins than A.*

Corollary 1.3.9 ([6]) *It is enough to investigate those of algorithms which pack the batches $B_1, B_{2,k}, B_{3,k}$ in such a way that they create only bins of types*

$$(i_1,0,0)_1, \quad (i_1,1,0)_{2,k}, \quad (i_1,2,0)_{2,k}, \quad (0,1,1)_{3,k}, \quad (0,2,0)_{2,k}, \quad (0,0,1)_{3,k}.$$

1.3.5 An LP model

Now we will construct a linear program problem to minimize the ACR of any online 3-BBP algorithm. We will give conditions for the number of elements in the batches and we also give lower bounds for the possible values of the ACR of the algorithms. To simplify the notation, instead of $R^{\infty}_{A,3}$ we will use R. The first condition concerns to the number of elements in the first batch B_1.

$$\sum_{(i_1,i_2,i_3)\in V_1} i_1 x^k_{i_1,i_2,i_3} = \sum_{i_1=1}^{6j} i_1 x^k_{i_1,i_2,i_3} = n_1 \tag{1.5}$$

where $x^k_{i_1,i_2,i_3}$ denote the number of (i_1, i_2, i_3) type bins while we pack the batches $B_1, B_{2,k}, B_{3,k}$. Let we remind the reader: $x^r_{i_1,i_2,i_3}$ and $x^s_{i_1,i_2,i_3}$ $(1 \le r, s \le j-1)$ *are the same variables*, their upper indices sign only the type of batches we used. Since the number of items in the first batch is independent from k, so we have here one equality.

We can give conditions for the number of items in the batch $B_{2,k}$. If we put one item from the second batch in a bin, then this bin must contain at least $2j-2k+1$ items from the first batch and it may not contain more items than $4j-k$. Since in this case $i_1 > 0$ – because of the Lemma 1.3.7 – we get that $i_3 = 0$ for these type of bins.

Similarly, if a bin contains 2 pieces from $B_{2,k}$ then the maximum number of the items from the first batch may not be more than $2j-2k$. For these type of bins $i_3 = 0$ is also valid, since we can not put any item from the batch $B_{3,k}$ if a bin contains 2 items from the batch $B_{2,k}$. Therefore

$$\sum_{i_1=2j-2k+1}^{4j-k} x^k_{i_1,1,0} + 2\sum_{i_1=1}^{2j-2k} x^k_{i_1,2,0} + x^k_{0,1,1} + 2x^k_{0,2,0} = n_{2,k}, \quad k = 1, 2, \ldots, j-1. \tag{1.6}$$

The last $j-1$ equations concern to the number of elements in the batch $B_{3,k}$ while we pack the third batch:

$$x^k_{0,1,1} + x^k_{0,0,1} = n_{3,k}, \quad k = 1, 2, \ldots, j-1, \tag{1.7}$$

Now we give three lower bounds for the ACR.

$$\sum_{i_1=1}^{6j} x^k_{i_1,i_2,i_3} \le R \cdot \mathrm{OPT}(B_1) \tag{1.8}$$

$$\sum_{i_1=1}^{6j} x^k_{i_1,i_2,i_3} + x^k_{0,1,1} + x^k_{0,2,0} \le R \cdot \mathrm{OPT}(B_1, B_{2,k}) \tag{1.9}$$

$$\sum_{i_1=1}^{6j} x^k_{i_1,i_2,i_3} + x^k_{0,1,1} + x^k_{0,2,0} + x^k_{0,0,1} \le R \cdot \mathrm{OPT}(B_1, B_{2,k}, B_{3,k}) \tag{1.10}$$

Let us consider the linear programming problem which has the $2(j+1)$ inequalities of (1.5) - (1.10) as conditions and its objective function is to minimize the value of R.

Now, we will eliminate the variables $x^k_{0,1,1}$, $x^k_{0,2,0}$, and $x^k_{0,0,1}$. Therefore we do the following calculations. For $k = 1, 2, \ldots, j-1$ we add the appropriate pairs of (1.9) and (1.10), and substitute the appropriate inequalities (1.6) and (1.7). Then we get the next system of inequalities for every $k = 1, 2, \ldots, j-1$.

$$\sum_{i_1=1}^{6j} i_1 x^k_{i_1,i_2,i_3} = n_1 \tag{1.11}$$

$$\sum_{i_1=1}^{6j} x^k_{i_1,i_2,i_3} \leq R \cdot \mathrm{OPT}(B_1) \tag{1.12}$$

$$2\sum_{i_1=1}^{6j} x^k_{i_1,i_2,i_3} - \sum_{i_1=2j+1-2k}^{4j-k} x^k_{i_1,1,0} - 2\sum_{i_1=1}^{2j-2k} x^k_{i_1,2,0} \leq \tag{1.13}$$
$$\leq R \cdot (\mathrm{OPT}(B_1, B_{2,k}) + \mathrm{OPT}(B_1, B_{2,k}, B_{3,k})) - n_{2,k} - n_{3,k},$$

where the inequality (1.13) is taken for all $k = 1, 2, \ldots, j-1$. We remind the reader that

$$n_1 = 6jn, \quad n_{2,k} = \frac{6j}{j-k}n, \quad \text{and} \quad n_{3,k} = \frac{6j}{j-k}n.$$

Let us substitute the values of $\mathrm{OPT}(B_1)$, $\mathrm{OPT}(B_1, B_{2,k})$, $\mathrm{OPT}(B_1, B_{2,k}, B_{3,k})$, $n_1, n_{2,k}$, and $n_{3,k}$ into the right hand sides. So we get the following $(k+1)$ conditions.

$$\sum_{i_1=1}^{6j} i_1 x^k_{i_1,i_2,i_3} = 6jn \tag{1.14}$$

$$\sum_{i_1=1}^{6j} x^k_{i_1,i_2,i_3} \leq R \cdot n \tag{1.15}$$

$$2\sum_{i_1=1}^{6j} x^k_{i_1,i_2,i_3} - \sum_{i_1=2j-2k+1}^{4j-k} x^k_{i_1,1,0} - 2\sum_{i_1=1}^{2j-2k} x^k_{i_1,2,0} \leq \tag{1.16}$$
$$\leq R \cdot (\frac{6j}{2(j-k)} + \frac{6j}{j-k})n - \frac{12j}{j-k}n, \quad k = 1, 2, \ldots, j-1.$$

If we divide the inequalities by n, and we introduce the following variables for every valid triplets:

$$z^k_{i_1,i_2,i_3} = \frac{x^k_{i_1,i_2,i_3}}{n},$$

then we get the following conditions.

$$\sum_{i_1=1}^{6j} i_1 z^k_{i_1,i_2,i_3} = 6j \tag{1.17}$$

$$\sum_{i_1=1}^{6j} z^k_{i_1,i_2,i_3} \leq R \tag{1.18}$$

$$2\sum_{i_1=1}^{6j} z^k_{i_1,i_2,i_3} - 2\sum_{i_1=1}^{2j-2k} z^k_{i_1,2,0} - \sum_{i_1=2j-2k+1}^{4j-k} z^k_{i_1,1,0} \leq (3R-4)\frac{6j}{2(j-k)}, \tag{1.19}$$

$$k = 1, 2, \ldots, j-1.$$

Making some calculations in (1.19) we get the following conditions

$$\sum_{i_1=1}^{6j} i_1 z^k_{i_1,i_2,i_3} = 6j \tag{1.20}$$

$$\sum_{i_1=1}^{6j} z^k_{i_1,i_2,i_3} \leq R \tag{1.21}$$

$$\sum_{i_1=2j-2k+1}^{4j-k} z^k_{i_1,1,0} + 2\sum_{i_1=4j-k+1}^{6j} z^k_{i_1,i_2,i_3} \leq (3R-4)\frac{6j}{2(j-k)}, \tag{1.22}$$

$$k = 1, 2, \ldots, j-1.$$

1.3.6 The lower bound

As we mentioned earlier, we do not need to consider all conditions from the $k = 1, \ldots, j-1$ possible ones. We assume that d of them is enough, i.e. erasing the conditions for $k = d+1, \ldots, j-1$, the lower bound remains the same. Later we will give the optimal value for d. Moreover, we make some further calculations and simplification. For each value of k, $(k = 1, 2, \ldots, d)$ we multiply equations (1.20) by (-1), inequalities (1.21) by $2(j-d)$. Furthermore, for $k = 1$ we multiply (1.22) by $2j - d + 2$ and all the other ones $(k = 2, \ldots, d)$ by 2.

Lemma 1.3.10 ([6]) *Making the linear combination with the above coefficients, the left hand side of the inequality is non-negative.*

Proof For every k, $k = 1, 2, \ldots, d$, according to the value of i_1, $1 \leq i_1 \leq 6j$, we will distinguish five different cases.

- Case 1. $1 \leq i_1 \leq 2(j-d)$. Since $z^k_{i_1,i_2,i_3}$ does not appear in the last $j-d$ inequalities, now the coefficients are $2(j-d) - i_1 \geq 0$.
- Case 2. $2(j-d)+1 \leq i_1 \leq 2j-2$. In this case for the coefficients we get

  $$2(j-d) - i_1 + 2\lceil (i_1 - 2j + 2d)/2 \rceil \geq 2(j-d) - i_1 + 2(i_1 - 2j + 2d)/2 = 0.$$

- Case 3. $2j-1 \le i_1 \le 4j-d$. In this case for the coefficients we get

$$2(j-d) - i_1 + (2j-d+2) + 2(d-1) = 4j - d - i_1 \ge 0.$$

- Case 4. $4j-d+1 \le i_1 \le 4j-1$. In this case the coefficients are

$$2(j-d) - i_1 + (2j-d+2) + 2\left(d - (i_1 - 4j + d + 1)\right) +$$

$$+4(i_1 - 4j + d) = -4j + d + i_1 \ge 0.$$

- Case 5. $4j \le i_1 \le 6j$ This case the coefficients are

$$2(j-d) - i_1 + 2(2j-d+2) + 4(d-1) = 6j - i_1 \ge 0.$$

□

Because the left hand side of the linear combination with the given coefficients is non-negative, we get the following inequality:

$$6j \le 2(j-d)R + 6j(2j-d+2)\frac{3R-4}{2(j-1)} + (3R-4)\cdot\sum_{k=2}^{d}\frac{6j}{(j-k)}. \tag{1.23}$$

The following inequality is easy to prove, see [47].

$$\sum_{k=2}^{d}\frac{1}{j-k} < \ln\frac{j-2}{j-d-1}.$$

Since $3R-4>0$, using the inequality (1.23), and making some calculations we get

$$\begin{aligned} 6j + \frac{12j(2j-d+2)}{j-1} + 24j\ln\frac{j-2}{j-d-1} \le \\ \le \left(2j - 2d + \frac{9j(2j-d+2)}{j-1} + 18j\ln\frac{j-2}{j-d-1}\right)R. \end{aligned} \tag{1.24}$$

and therefore

$$\begin{aligned} 30j^2 + 18j - 12jd + 24j(j-1)\ln\frac{j-2}{j-d-1} \le \\ \le \left(20j^2 + 16j - 11j\cdot d + 2d + 18j(j-1)\ln\frac{j-2}{j-d-1}\right)R. \end{aligned} \tag{1.25}$$

Ordering this inequality we get

$$f(j,d) = \frac{30j^2 + 18j - 12j\cdot d + 24j(j-1)\ln\frac{j-2}{j-d-1}}{20j^2 + 16j - 11j\cdot d + 2d + 18j(j-1)\ln\frac{j-2}{j-d-1}} \le R. \tag{1.26}$$

Now we have a lower estimation on the asymptotic ratio R for our construction. Formula (1.26) has 2 parameters j and d, $(d = 1, ..., j-2)$. For a fixed j we want

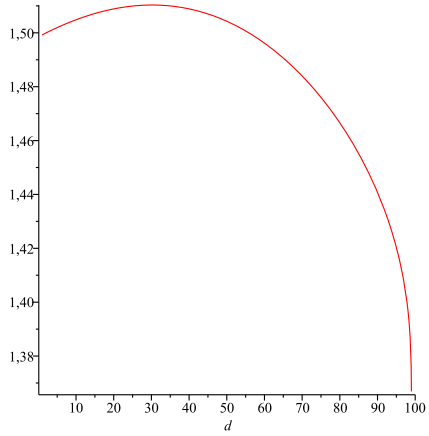

Figure 1.1: Graph of the function $f(d)$ for $j = 100$.

to find such d, which maximizes the left hand side of (1.26). In order to determine this d, we consider the function $f(j,d)$ as a *1-variable* continuous function $f(d)$ for a fixed j on the interval $[1, j-2]$. It is easy to see that $f(d)$ has one maximum in the given interval. To decide the maximum value of d first we need to derivate $f(d)$ by d.

$$f'(d) = \frac{-12\,j + 24\,\frac{j(j-1)}{j-d-1}}{20\,j^2 + 16\,j - 11\,jd + 2\,d + 18\,j\,(j-1)\ln\left(\frac{j-2}{j-d-1}\right)} - \frac{\left(30\,j^2 + 18\,j - 12\,jd + 24\,j\,(j-1)\ln\left(\frac{j-2}{j-d-1}\right)\right)\left(-11\,j + 2 + 18\,\frac{j(j-1)}{j-d-1}\right)}{\left(20\,j^2 + 16\,j - 11\,jd + 2\,d + 18\,j\,(j-1)\ln\left(\frac{j-2}{j-d-1}\right)\right)^2}.$$

Now we can solve the equation $f'(d) = 0$, so we get the value of d_0 where the function takes its maximum.

$$d_0 = \frac{1}{4}\,\frac{9\,j-4}{W\left(-1, -\frac{1}{4}\,\frac{e^{\frac{-1}{8}\,\frac{23\,j-2}{j-1}}(9\,j-4)}{j-2}\right)} + j - 1$$

where $W(-1,x)$ is the negative branch of the Lambert function. Substituting d_0 into $f(d)$ and taking its limit while $j \to \infty$ we get the required formula for R.

$$R \geq \frac{32\,W\left(-1, -\frac{9}{4}e^{-\frac{23}{8}}\right) + 36}{24\,W\left(-1, -\frac{9}{4}e^{-\frac{23}{8}}\right) + 33} \approx 1.51211383.$$

Table 1.1 shows the lower bounds for different values of j and d. So, we have the following theorem.

Theorem 1.3.11 ([6]) *If A is a batched algorithm for 3-BBPP, then*

$$R^{\infty}_{A,3} \geq 1.51211383.$$

j	d	$f(j,d)$
5	1	1.480075901
10	3	1.494928787
20	6	1.503357743
50	15	1.508573181
100	30	1.510335641
200	60	1.511221192
500	152	1.511757013
1000	305	1.511935384

Table 1.1: The values of $f(j,d)$

1.3.7 Conclusions

In this section we investigated the 3-BBP problem, and we gave a lower bound for any on-line 3-BBP algorithm. It is possible to extend the construction to the K-BBP for $K \geq 4$. This was presented in the paper [9] together with some other results on this problem. Using the idea of branching lists the lower bound for the online bin packing problem was improved to 1.54278 in [10].

1.4 Bin Packing with Cardinality Constraints

1.4.1 Definitions and preliminaries

In this section we study a variant of bin packing called bin packing with cardinality constraints (BPCC) . In this problem, the input consists of items, denoted by $1, 2, \ldots, n$, such that item i has a size $s_i > 0$ associated with it, and there is a global parameter $k \geq 2$, called the cardinality constraint. The goal is to partition the input items into subsets, called bins, such that the total size of items of every bin is at most 1, and the number of items packed into each bin does not exceed k. We believe that bounding the number of items as well as their total size provides a more accurate model for packing problems; for example, a data center can usually only store a constant number of files. We study online algorithms and the asymptotic competitive ratio as the measurement of the quality of the algorithms and only lower bounds are considered. The case $k = 2$ is solvable using matching techniques in the offline scenario, but it is not completely resolved in the online scenario. Liang [63] showed a lower bound of $\frac{4}{3}$ on the asymptotic competitive ratio for this case, Babel et al. [4] improved the lower bound to $\sqrt{2} \approx 1.41421$, and designed an algorithm whose asymptotic competitive ratio is at most $1 + \frac{1}{\sqrt{5}} \approx 1.44721$ (improving over the previous bound, which was proved for FF). Fujiwara and Kobayashi [41] improved the lower bound to 1.42764.

Lower bounds for many values of k were given by Fujiwara and Kobayashi in [41], and in particular, they proved lower bounds of 1.5 and $\frac{25}{17} \approx 1.47058$ for $k = 4$ and $k = 5$, respectively. For $6 \leq k \leq 9$, the best lower bound remained 1.5, which was implied by the lower bound of Yao [86], and for $k = 10$ and $k = 11$, lower

bounds of $\frac{80}{53} \approx 1.50943$ and $\frac{44}{29} \approx 1.51724$, respectively, were proved in [41] (see [41] for the lower bounds of other values of k). In this section we provide improved lower bounds on the asymptotic competitive ratio of arbitrary online algorithms for $k = 5, 7, 8, 9, 10, 11$. The values of these lower bounds are 1.5 for $k = 5$, and approximately 1.51748, 1.5238, 1.5242, 1.526, 1.5255, for $k = 7, 8, 9, 10, 11$, respectively. We also provide improved lower bounds for larger values of k in Table 1.3 ([16]).

1.4.2 Lower bounds

In this section we present a method for proving lower bounds on the asymptotic competitive ratio of bin packing problems. This method can be applied for different variants and for bin packing with cardinality constraints it allows us to improve the lower bounds known for relatively small values of k.

Consider an input of the following form for a given bin packing problem Π. Let $\theta \geq 2$ be a fixed positive integer. There are θ lists of items, where the list L_i (for $1 \leq i \leq \theta$) has identical items, each of size s_i, where $s_1 < s_2 < \cdots < s_\theta$. For a large integer $N > 0$, the list L_i has $\alpha_i \cdot N$ items, where $0 < \alpha_i \leq 1$ is a rational parameter for $i = 1, \ldots, \theta$ (N will be selected such that $\alpha_i \cdot N$ is an integer). The items are presented to an online algorithm sorted by non-decreasing sizes (that is, items are presented sorted by non-decreasing indices of their lists) and the input may be stopped after all the items of some list were presented. That is, there are θ possible inputs, and we will examine the behaviour of the algorithm for all possible inputs.

Given a set of values $w_i > 0$ for $i = 1, \ldots, \theta$, w_i is called the *weight* of an item of list L_i, and the weight of a bin is equal to the total weight of its items. For $i = 1, \ldots, \theta + 1$, let W_i denote the maximum weight of any bin that contains only items of lists $i, i + 1, \ldots, \theta$, and no items of earlier lists (so $W_{\theta+1} = 0$), assuming that an unlimited number of items of any size is available. Obviously, the algorithm has no such bins if the input stops after list L_j, for some $j < i$. By definition, $W_i \geq W_{i+1}$ for $i = 1, \ldots, \theta$. For $i = 0, \ldots, \theta$, let OPT_i denote the cost of an optimal solution for the input that consists of the items of lists $L_1, \ldots, L_i$ (thus $OPT_0 = 0$), and denote this input by $\mathcal{L}_i$. We have $OPT_i = \Theta(N)$, as $OPT_i \geq \alpha_i \cdot N \cdot s_i$ (because this is the total size of items of list i) and $OPT_i \leq i \cdot N$ (as this is an upper bound on the number of input items in $\mathcal{L}_i$). Let O_i be an upper bound on $\frac{OPT_i}{N}$ for $1 \leq i \leq \theta$ and let $O_0 = 0$ (these values are constants depending on the input parameters).

Theorem 1.4.1 ([16]) *The asymptotic competitive ratio of any online (deterministic or randomized) algorithm the bin packing problem* Π *is at least*

$$\frac{\sum_{i=1}^{\theta} \alpha_i \cdot w_i}{\sum_{i=1}^{\theta} (O_i - O_{i-1}) \cdot W_i} .$$

Our method of proving lower bounds for BPCC is to use inputs where the numbers of items in the lists are not necessarily equal. Consider the case $k = 5$.

Let $\theta = 4$, $\alpha_1 = \frac{1}{2}$, $\alpha_i = 1$ for $i = 2, 3, 4$. Let $0 < \delta < \frac{1}{2000}$, $s_1 = \frac{1}{42} - \delta > 0$, $s_2 = \frac{1+\delta}{7}$, $s_3 = \frac{1+\delta}{3}$, and $s_4 = \frac{1+\delta}{2}$. In this case, there are less items of size s_1 (than the numbers of other items), since a bin that contains two items of size s_2 and two items of size s_3 can only contain one additional item. We use $O_1 = \frac{1}{10}$, $O_2 = \frac{3}{10}$ (as any five items of sizes at most s_2 can be packed into a bin), $O_3 = \frac{1}{2}$ (as a set of five items, consisting of two items of size s_3, two items of size s_2, and one item of size s_1 can be packed into a bin), and $O_4 = 1$ (as a set of items consisting of one item of each size can be packed into a bin, while only half of the bins will contain an item of size s_1).

Consider the cases $k = 7, 8, \ldots, 11$. Let $\theta = 4$, the item sizes are the same as the case $k = 5$, $\alpha_1 = \frac{k-6}{6}$ (where $\alpha_1 < 1$), $\alpha_i = 1$ for $i = 2, 3, 4$. The motivation for the value α_1 is that a bin that has six items of size s_2 can contain only $k - 6$ additional items. We use $O_1 = \frac{k-6}{6k}$, $O_2 = \frac{1}{6}$ (as a set of six items of size s_2 and $k - 6$ items of size s_1 can be packed into a bin), $O_3 = \frac{1}{2}$ (as a set consisting of at most two items of each one of the three sizes s_1, s_2, s_3 can be packed into a bin), and $O_4 = 1$. Let $w_2 = 1$, $w_3 = w_4 = 2$ for all cases. For $k = 5$, let $w_1 = 2$, for $k = 7, 8$, let $w_1 = 1$, for $k = 9$, let $w_1 = \frac{1}{2}$, and for $k = 10, 11$, let $w_1 = \frac{1}{3}$.

Lemma 1.4.2 ([16])

1. *For all cases* $W_2 \leq 6$, $W_3 \leq 4$, *and* $W_4 \leq 2$.
2. *For* $k = 5$, $W_1 \leq 10$, *for* $k = 7, 8$, $W_1 \leq k + 2$, *for* $k = 9$, $W_1 \leq 8$, *and for* $k = 10, 11$, $W_1 \leq \frac{k}{3} + 4$.

Proof We start with the first part. The claim regarding W_4 holds since $W_4 = w_4$ must hold (as every bin opened for the last list contains exactly one item). In the cases where there may be items of L_3 packed into a bin, any item of size s_4 can be replaced with an item of size s_3 without increasing the total size or number of items, and without changing the total weight (as $w_3 = w_4$ in all cases). Thus, we assume that no such items are packed into the considered bins. Any bin with items of L_3 can contain at most two items and therefore $W_3 \leq 2w_3$. Consider a bin with no items of list L_1. Let y_2 be the number of items of size s_2, and let $y_3 \leq 2$ be the number of items of size s_3. Their weight is $y_2 + 2y_3$, and their total size is above $\frac{y_2+2y_3}{7}$ (and no larger than 1). Thus, $W_2 \leq y_2 + 2y_3 \leq 6$.

Next, we bound W_1 for all values of k considered here. For $k = 5$, since the weight of any item is at most 2, and there are at most five items packed into each bin, $W_1 \leq 10$. Otherwise, consider a packed bin, and let y_2 and $y_3 \leq 2$ have the same meaning as before, while y_1 is the number of items of size s_1. We have that $y_1 + y_2 + y_3 \leq k$ must hold, and by the value of W_2, $y_2 + 2y_3 \leq 6$. The total size of items is at least $y_1(\frac{1}{42} - \delta) + y_2(\frac{1+\delta}{7}) + y_3(\frac{1+\delta}{3}) > \frac{y_1+6y_2+14y_3}{42} - y_1\delta$. Since $y_1\delta < \frac{1}{42}$, we have $y_1 + 6y_2 + 14y_3 \leq 42$. For $k = 7, 8$, the weight of the bin is $y_1 + y_2 + 2y_3$. Since $y_1 + y_2 + y_3 \leq k$ and $y_3 \leq 2$, $W_1 \leq k + 2$.

For $k = 9$, the weight is $y_1/2 + y_2 + 2y_3$. Adding $y_1 + 6y_2 + 14y_3 \leq 42$ to $4(y_1+y_2+y_3) \leq 36$, we get $5y_1+10y_2+18y_3 \leq 78$, or alternatively, $y_1/2+y_2+2y_3 \leq 7.8 + 0.2y_3$. Thus, if $y_3 \leq 1$, $W_1 \leq 8$. If $y_3 = 2$, then by substituting it into the inequalities we get $y_1+6y_2 \leq 14$ and $y_1+y_2 \leq 7$, or alternatively, $y_1+2y_2 \leq 14-4y_2$

and $y_1 + 2y_2 \leq 7 + y_2$. If $y_2 \geq 2$, then the first inequality implies $y_1/2 + y_2 \leq 3$, and if $y_2 \leq 1$, then the second inequality implies $y_1/2 + y_2 \leq 4$. In both cases, $y_1/2 + y_2 + 2y_3 \leq 8$.

For $k = 10, 11$, the weight is $y_1/3 + y_2 + 2y_3$, and we will show $y_1 + 3y_2 + 6y_3 \leq k + 12$. As $y_3 \leq 2$, we consider three cases. If $y_3 = 0$, then we find $y_2 \leq 6$ and $y_1 + y_2 \leq k$. Thus, $y_1 + 3y_2 \leq k + 12$. If $y_3 = 1$, then we find $y_1 + 6y_2 \leq 28$, $y_2 \leq 4$, and $y_1 + y_2 \leq k - 1$. If $y_2 \leq 3$, we get $y_1 + 3y_2 + 6y_3 \leq (k-1) + 2 \cdot 3 + 6 < k + 12$. If $y_2 = 4$, then by the first inequality we find $y_1 \leq 4$, and $y_1 + 3y_2 + 6y_3 \leq 4 + 3 \cdot 4 + 6 = 22 \leq k + 12$ as $k \geq 10$. Finally, if $y_3 = 2$, we get $y_1 + 6y_2 \leq 14$, $y_2 \leq 2$, and $y_1 + y_2 \leq k - 2$. If $y_2 \leq 1$, we get $y_1 + 3y_2 + 6y_3 \leq (k-2) + 2 \cdot 1 + 6 \cdot 2 = k + 12$. If $y_2 = 2$, then by the first inequality we find $y_1 \leq 2$, and $y_1 + 3y_2 + 6y_3 \leq 2 + 3 \cdot 2 + 6 \cdot 2 = 20 \leq k + 12$. □

We apply Theorem 1.4.1 to get the following.

Theorem 1.4.3 ([16]) *The following values are lower bounds on the competitive ratios.*

- $\frac{3}{2} = 1.5$ *for* $k = 5$.
- $\frac{k^2+24k}{k^2+10k+24}$ *for* $k = 7, 8$. *This value is equal to* $217/143 \approx 1.5174825$ *for* $k = 7$ *and to* $\frac{32}{21} \approx 1.5238095$ *for* $k = 8$.
- $\frac{10.5}{62/9} = \frac{189}{124} \approx 1.5241935$, *for* $k = 9$.
- $\frac{k^2+84k}{k^2+48k+36}$ *for* $k = 10, 11$. *This value is equal to* $235/154 \approx 1.525974$ *for* $k = 10$ *and to* $\frac{209}{137} \approx 1.525547$ *for* $k = 11$.

Proof For $k = 5$, $\sum_{i=1}^{4} \alpha_i \cdot w_i = 6$ and $\sum_{i=1}^{4} (O_i - O_{i-1}) \cdot W_i = 4$.
For $k = 7, 8$, $\sum_{i=1}^{4} \alpha_i \cdot w_i = \frac{k+24}{6}$ and $\sum_{i=1}^{4} (O_i - O_{i-1}) \cdot W_i = \frac{7}{3} + \frac{6}{k} + \frac{k^2-4k-12}{6k}$.
For $k = 9$, $\sum_{i=1}^{4} \alpha_i \cdot w_i = 5.25$ and $\sum_{i=1}^{4} (O_i - O_{i-1}) \cdot W_i = \frac{31}{9}$.
For $k = 10, 11$, $\sum_{i=1}^{4} \alpha_i \cdot w_i = \frac{k+84}{18}$ and $\sum_{i=1}^{4} (O_i - O_{i-1}) \cdot W_i = \frac{7}{3} + \frac{6}{k} + \frac{k^2+6k-72}{18k}$. □

Note that in the cases $k = 6$ and $k = 12$, our methods do not produce improved lower bounds, and they give exactly the known lower bound.

Next, consider the cases $14 \leq k \leq 18$. Let $\theta = 4$, $\alpha_i = 1$ for $i = 1, 2, 3, 4$. Let $0 < \delta < \frac{1}{2000}$, $s_1 = \frac{1}{18} - 3\delta > 0$, $s_2 = \frac{1+\delta}{9}$, $s_3 = \frac{1+\delta}{3}$, and $s_4 = \frac{1+\delta}{2}$. We use $O_1 = \frac{1}{k}$, $O_2 = \frac{1}{6}$ (as a set of six items of size s_1 and six items of size s_2 can be packed into a bin), $O_3 = \frac{1}{2}$ (as a set of six items, consisting of two items of each size out of s_1, s_2, s_3 can be packed into a bin), and $O_4 = 1$ (as a set of items consisting of one item of each size can be packed into a bin). Let $w_1 = 1$, $w_2 = 2$, and $w_3 = w_4 = 6$.

Lemma 1.4.4 ([16]) *We have* $W_4 \leq 6$, $W_3 \leq 12$, $W_2 \leq 16$, *and* $W_1 \leq 18$.

Proof As in the proof of Lemma 1.4.2, $W_4 = w_4$ and $W_3 = w_3 + w_4$. We will prove upper bounds on W_1 and W_2 such that the number of packed items is not necessarily bounded by k. This may only increase the bounds.

To prove the bound for W_2, consider a bin with items of sizes above $\frac{1}{9}$. Replace any item of size s_4 or s_3 with three items of size s_2. The total size of items cannot increase while the total weight is unchanged. The bin now contains at most 8 items of size s_2, and therefore its weight is at most 16.

To prove the bound for W_1, consider a bin B. Replace any item of size s_4 or s_3 with six items of size s_1, and any item of size s_2 is replaced with two items of size s_1. The total size of items cannot increase while the total weight is unchanged. The bin now contains at most 18 items of size s_1, and therefore its weight is at most 18. □

We apply Theorem 1.4.1 to get the following.

Theorem 1.4.5 ([16]) *The value* $\frac{45k}{29k+6}$ *is a lower bound on the asymptotic competitive ratio for* k*, where* $14 \leq k \leq 18$*.*

Proof We have $\sum_{i=1}^{4} \alpha_i \cdot w_i = 15$ and $\sum_{i=1}^{4}(O_i - O_{i-1}) \cdot W_i = 18/k + 16(1/6 - 1/k) + 12/3 + 6/2$. □

Note that in the cases $k = 12, 13$, our method does not produce improved lower bounds.

Finally, consider the cases $k = 19, 20, \ldots, 35$. Let $\theta = 5$, $\alpha_1 = \frac{k-18}{18}$, $\alpha_i = 1$ for $i = 2, 3, 4, 5$. Let $0 < \delta < \frac{1}{10000}$, $s_1 = \frac{1}{342} - \delta$, $s_2 = \frac{1+\delta}{19}$, $s_3 = \frac{1+\delta}{9}$, $s_4 = \frac{1+\delta}{3}$, and $s_5 = \frac{1+\delta}{2}$. In this case, there are less items of size s_1, since a bin that contains 18 items of size s_2 can only contain $k - 18$ additional items. We use $O_1 = \frac{k-18}{18k}$ (any bin will contain k items), $O_2 = \frac{1}{18}$ (any bin will contain 18 items of size s_2 and $k - 18$ items of size s_1), $O_3 = \frac{1}{6}$ (any bin will contain six items of size s_3, six items of size s_2, and at most six items of size s_1), $O_4 = \frac{1}{2}$ (any bin will contain two items for each one of the sizes s_2, s_3, s_4, and at most two items of size s_1), and $O_5 = 1$ (any bin will contain one item of each of the sizes of s_2, s_3, s_4, s_5, and at most one item of size s_1). Let $w_2 = 1$, $w_3 = 2$, and $w_4 = w_5 = 6$. The value of w_1 will differ for the different values of k, and we denote it by ρ_k, where $0 < \rho_k < 1$.

Lemma 1.4.6 ([16]) *We have* $W_2 \leq 18$*,* $W_3 \leq 16$*,* $W_4 \leq 12$*, and* $W_5 \leq 6$*.*

Proof As in previous cases, $W_5 = w_5$ and $W_4 = 2 \cdot w_4$. We will prove upper bounds on W_2 and W_3 for bins where the number of packed items is not necessarily bounded by k. This may only increase the bounds. Given a bin with items of sizes in $\{s_3, s_4, s_5\}$, replace each item of size s_4 or s_5 with three items of size s_3. As a result, the total size does not increase, and the total weight is unchanged. Since at most eight items of size s_3 can be packed into a bin, $W_3 \leq 16$. Given a bin with items of sizes in $\{s_2, s_3, s_4, s_5\}$, replace each item of size s_4 or s_5 with six items of size s_2, and each item of size s_3 is replaced with two items of size s_2. As a result, the total size does not increase, and the total weight is unchanged. Since at most 18 items of size s_3 can be packed into a bin, $W_3 \leq 18$. □

Consider a bin that possibly contains items of all lists, where the total weight of items of lists L_2, L_3, L_4, L_5 is exactly 18. Let λ_k denote the maximum number of items of size s_1 that the bin can contain under this condition. Similarly, consider a bin where the total weight of items of lists L_2, L_3, L_4, L_5 is exactly 17. Let ψ_k denote the maximum number of items of size s_1 that the bin can contain under this condition. Let $\rho_k = \frac{1}{\psi_k - \lambda_k}$ (which is well defined, as we will show that $\psi_k > \lambda_k$ for all k). Recall that for any k, the value w_1 is defined by ρ_k. We define an additional parameter, $\phi_k = 18 + \rho_k \cdot \lambda_k$ for all values of k considered here. These values are displayed in Table 1.2.

Lemma 1.4.7 ([16]) *Let $k \in \{19, 20, \ldots, 35\}$. The values λ_k and ψ_k are as in Table 1.2, and $W_1 \leq \phi_k$.*

Proof We start with proving that the values of λ_k and ψ_k given in Table 1.2 correspond to our definition of these values. For a given bin, let y_2, y_3, and y_4, be the numbers of items of sizes s_2, s_3, and s_4 packed into the bin (we replace items of size s_5, if such items exist, with items of size s_4, as they are smaller and have the same weight).

If the total weight of the items of size at least s_2 is 18, then $y_2 + 2y_3 + 6y_4 = 18$, and the total size of items is $(1+\delta)(\frac{y_2}{19} + \frac{y_3}{9} + \frac{y_4}{3}) = \frac{1+\delta}{18}(y_2 + 2y_3 + 6y_4) - \frac{1+\delta}{342}y_2 = 1 + \delta - \frac{1+\delta}{342}y_2$. The bin can still contain items of size s_1 of total size no larger than $\frac{1+\delta}{342}y_2 - \delta$. Let y_1 be the number of such items. We will show $y_1 = \min\{k - y_2 - y_3 - y_4, y_2\}$ by proving that the total size of $y_2 + 1$ items of size s_1 exceeds $\frac{1+\delta}{342}y_2 - \delta$, while the total size of y_2 such items does not exceed this value (and obviously the bin cannot contain more than $k - y_2 - y_3 - y_4$ additional items).

We find that $y_1 \leq y_2$, as the total size of y_2+1 items of size s_1 is $(y_2+1)(\frac{1}{342} - \delta)$, and $(y_2 + 1)(\frac{1}{342} - \delta) > \frac{1+\delta}{342}y_2 - \delta$ is equivalent to $343\delta \cdot y_2 < 1$, which holds as $y_2 \leq 18$ and $\delta < \frac{1}{10000}$. On the other hand, $y_2(\frac{1}{342} - \delta) \leq \frac{1+\delta}{342}y_2 - \delta$ is equivalent to $\frac{343y_2}{342} \geq 1$ (which holds for $y_2 \geq 1$), and therefore if $y_2 \geq 1$, then y_2 items of size s_1 can be packed into the bin (in terms of total size).

For any valid triple (y_2, y_3, y_4) of numbers of items, the maximum value of y_1 is therefore $\min\{k - y_2 - y_3 - y_4, y_2\}$ items. To find the possible triples (y_2, y_3, y_4), we take into account that $y_4 \leq 2$ (as no bin can contain more than two items of sizes above $\frac{1}{3}$) and $y_3 + 3y_4 \leq 8$ (as the total weight of these items is at most $W_3 \leq 16$, and it is equal to $w_3 \cdot y_3 + w_4 \cdot y_4 = 2y_3 + 6y_4$). These patterns are: $(2,2,2)$, $(4,1,2)$, $(6,0,2)$, $(2,5,1)$, $(4,4,1)$, $(6,3,1)$, $(8,2,1)$, $(10,1,1)$, $(12,0,1)$, $(2,8,0)$, $(4,7,0)$, $(6,6,0)$, $(8,5,0)$, $(10,4,0)$, $(12,3,0)$, $(14,2,0)$, $(16,1,0)$, and $(18,0,0)$. Thus,

$$\lambda_k = \max\{\min\{2, k-6\}, \min\{4, k-7\}, \min\{6, k-8\}, \min\{8, k-11\},$$

$$\min\{10, k-12\}, \min\{12, k-13\}, \min\{14, k-16\}, \min\{16, k-17\},$$

$$\min\{18, k-18\}\} \ .$$

The last bound was computed by considering each pattern separately, and computing $\min\{k - y_2 - y_3 - y_4, y_2\}$, then removing any entry that is dominated by

another entry, for example, $\min\{2, k-8\}$ that results from the fourth triple is dominated by $\min\{2, k-6\}$ resulting from the first triple. The values in the table are with accordance to this calculation.

If the total weight of the items of sizes above $\frac{1}{19}$ is 17, then $y_2+2y_3+6y_4 = 17$, and the total size of items is $(1+\delta)(\frac{y_2}{19}+\frac{y_3}{9}+\frac{y_4}{3}) = \frac{1+\delta}{18}(y_2+2y_3+6y_4) - \frac{1+\delta}{342}y_2 = \frac{17(1+\delta)}{18} - \frac{1+\delta}{342}y_2$. The bin can contain items of size s_1 of total size no larger than $\frac{1+\delta}{18}+\frac{1+\delta}{342}y_2-\delta$. Let y_1 be this number. In this case we show that $y_1 = \min\{y_2+19, k-y_2-y_3-y_4\}$. To show $y_1 \leq y_2+19$, we prove that the total size of y_2+20 items of size s_1 exceeds $\frac{1+\delta}{18}+\frac{1+\delta}{342}y_2-\delta$. Indeed, $(y_2+20)(\frac{1}{342}-\delta) > \frac{1+\delta}{342}y_2-\delta+\frac{1+\delta}{18}$ is equivalent to $\frac{1}{342} > \frac{343y_2\delta}{342}+\frac{343\delta}{18}$, which holds as $y_2 \leq 18$ and $\delta < \frac{1}{10000}$. On the other hand, it is possible to pack y_2+19 items of size s_1 (in terms of total size) as the empty space is $\frac{1+\delta}{342}y_2-\delta+\frac{1+\delta}{18}$, we saw that y_2 items can be packed into a space of $\frac{1+\delta}{342}y_2-\delta$, and since $s_1 < \frac{1}{342}$, 19 items of size s_1 can be packed into a space of $\frac{1+\delta}{18}$. Thus, for any triple (y_2, y_3, y_4), it is possible to pack $\min\{k-y_2-y_3-y_4, y_2+19\}$ items. The possible triples are: $(1,2,2)$, $(3,1,2)$, $(5,0,2)$, $(1,5,1)$, $(3,4,1)$, $(5,3,1)$, $(7,2,1)$, $(9,1,1)$, $(11,0,1)$, $(1,8,0)$, $(3,7,0)$, $(5,6,0)$, $(7,5,0)$, $(9,4,0)$, $(11,3,0)$, $(13,2,0)$, $(15,1,0)$, and $(17,0,0)$. Thus,

$$\psi_k = \max\{\min\{20, k-5\}, \min\{22, k-6\}, \min\{24, k-7\}, \min\{26, k-10\},$$
$$\min\{28, k-11\}, \min\{30, k-12\}, \min\{32, k-15\},$$
$$\min\{34, k-16\}, \min\{36, k-17\}\} \ .$$

The calculation is similar to the one for λ_k, and the values for ψ_k in the table are deduced from this calculation.

Consider a bin B, and let X denote the total weight of items that do not belong to list L_1 that are packed into B. We have $X \leq 18$, as $W_2 \leq 18$.

If $X = 18$, then the total weight of items is at most $X+\rho_k \cdot y_1 \leq 18+\rho_k \cdot \lambda_k = \phi_k$. If $X = 17$, then the total weight of items is at most $17+\rho_k \cdot \psi_k = 17+\rho_k \cdot (\psi_k - \lambda_k)+\rho_k \cdot \lambda_k \leq 18+\rho_k \cdot \lambda_k = \phi_k$, by the definitions of ρ_k and ϕ_k.

We claim that otherwise (if $X \leq 16$), the total weight of items is no larger than $X+\rho_k \cdot k$. For $k \leq 32$, $(k-\lambda_k)\rho_k \leq 2$, and therefore $X+k \cdot \rho_k \leq 16+2+\lambda_k\rho_k = \phi_k$. If $k \geq 33$, and $X \leq 15$, since $(k-\lambda_k)\rho_k \leq 3$, we also find $X+k \cdot \rho_k \leq 15+3+\lambda_k\rho_k = \phi_k$. In the case $X = 16$, there must be at least four items of lists L_2, L_3, L_4, L_5 packed into the bin, as the total weight of three items is at most 14 (there can be at most two items of weight 6 as their sizes exceed $\frac{1}{3}$). Thus, there are at most $k-4$ items whose weights are ρ_k. We have that $(k-4-\lambda_k)\rho_k < 2$ for $k \in \{33, 34, 35\}$, and $X+(k-4) \cdot \rho_k \leq 16+2+\lambda_k\rho_k = \psi_k$ in this case as well. □

Theorem 1.4.8 ([16]) *The values stated in Table 1.3 are lower bounds on the competitive ratios for* $k = 19, 20, \ldots, 35$.

Proof Recall that $\alpha_1 = \frac{k-18}{18}$, and $\alpha_i = 1$ for $i = 2, 3, 4, 5$. Thus, $\sum_{i=1}^{\theta} \alpha_i \cdot w_i = \frac{k-18}{18} \cdot \rho_k + w_2 + w_3 + w_4 + w_5 = \frac{k-18}{18} \cdot \rho_k + 15$.

value of k	λ_k	ψ_k	ρ_k	$(k-\lambda_k)\rho_k$	$\phi_k - 18 = \rho_k\lambda_k$
19	8	14	1/6	11/6	4/3
20	8	15	1/7	12/7	8/7
21	9	16	1/7	12/7	9/7
22	10	17	1/7	12/7	10/7
23	10	18	1/8	13/8	5/4
24	11	19	1/8	13/8	11/8
25	12	20	1/8	13/8	3/2
26	12	20	1/8	14/8	3/2
27	12	21	1/9	15/9	4/3
28	12	22	1/10	8/5	6/5
29	13	22	1/9	16/9	13/9
30	14	23	1/9	16/9	14/9
31	14	24	1/10	17/10	7/5
32	15	24	1/9	17/9	5/3
33	16	24	1/8	17/8	2
34	16	24	1/8	18/8	2
35	17	25	1/8	18/8	17/8

Table 1.2: Auxiliary variables for the analysis of lower bounds for $k = 19, 20, \ldots, 35$.

We have $O_1 - O_0 = \frac{k-18}{18k}$, $O_2 - O_1 = \frac{1}{k}$, $O_3 - O_2 = \frac{1}{9}$, $O_4 - O_3 = \frac{1}{3}$, and $O_5 - O_4 = \frac{1}{2}$. Thus, $\sum_{i=1}^{\theta}(O_i - O_{i-1})W_i = \frac{k-18}{18k} \cdot (18 + \rho_k \cdot \lambda_k) + \frac{18}{k} + \frac{16}{9} + \frac{12}{3} + \frac{6}{2} = 1 - \frac{18}{k} + \frac{\rho_k \cdot \lambda_k}{18} - \frac{\rho_k \cdot \lambda_k}{k} + \frac{18}{k} + \frac{79}{9} = \frac{88}{9} + \rho_k \cdot \lambda_k(\frac{1}{18} - \frac{1}{k})$.

Therefore, using Theorem 1.4.1 we find a lower bound of $\frac{15+(k-18)\rho_k/18}{88/9+\rho_k\lambda_k(1/18-1/k)}$ on the asymptotic competitive ratio for $k = 19, 20, \ldots, 35$. □

The construction that was used for $k = 19, \ldots, 35$ can be used for $k = 36$, but the resulting lower bound is lower than the known lower bound [41]. It is possible, however, to prove improved bounds for larger values of k. Consider, for example, the cases $k = 43, 44, 45$. Let $s_1 = \frac{1}{1806} - \delta$, $s_2 = \frac{1+\delta}{43}$, $s_3 = \frac{1+\delta}{7}$, $s_4 = \frac{1+\delta}{3}$, and $s_5 = \frac{1+\delta}{2}$; $\alpha_1 = \frac{k-42}{42}$, and $\alpha_i = 1$ for $i = 2, 3, 4, 5$. It can be verified that using the weights $w_1 = \rho_k$, $w_2 = 1$, $w_3 = 6$, and $w_4 = w_5 = 12$, where $\rho_{43} = \frac{1}{14}$, $\rho_{44} = \frac{1}{15}$, and $\rho_{45} = \frac{1}{16}$, gives $W_1 = 42 + \rho_k\lambda_k$, $W_2 = 42$, $W_3 = 36$, $W_4 = 24$, and $W_5 = 12$. This gives lower bound of approximately 1.53903, 1.53906, and 1.53909 on the asymptotic competitive ratios for $k = 43, 44, 45$, respectively. This slightly improves the previously known lower bound of approximately 1.53900 [85] mentioned in [41].

value of k	previous asymptotic LB	new LB	
5	1.47058 [41]	3/2	= 1.5
7	1.5 [86]	217/143	≈ 1.51748
8	1.5 [86]	32/21	≈ 1.52380
9	1.5 [86]	189/124	≈ 1.52419
10	1.50943 [41]	235/154	≈ 1.52597
11	1.51724 [41]	209/137	≈ 1.52554
14	1.52595 [41]	315/206	≈ 1.52912
15	1.52912 [41]	75/49	≈ 1.53061
16	1.52567 [41]	72/47	≈ 1.53191
17	1.52312 [41]	765/499	≈ 1.53306
18	1.52459 [41]	135/88	≈ 1.53409
19	1.52678 [41]	30799/20072	≈ 1.53442
20	1.52912 [41]	2365/1541	≈ 1.53471
21	1.52941 [41]	13251/8633	≈ 1.53492
22	1.52914 [41]	10417/6786	≈ 1.53507
23	1.53004 [41]	49795/32434	≈ 1.53527
24	1.53086 [41]	152/99	≈ 1.53535
25	1.53162 [41]	54175/32284	≈ 1.53539
26	1.53231 [41]	3523/2294	≈ 1.53574
27	1.53296 [41]	2439/1588	≈ 1.53589
28	1.53356 [41]	1897/1235	≈ 1.53603
29	1.53412 [41]	70789/46079	≈ 1.53625
30	1.53465 [41]	6105/3974	≈ 1.53623
31	1.53514 [41]	84103/54742	≈ 1.53635
32	1.53560 [41]	39104/25449	≈ 1.53656
33	1.53603 [41]	23925/15568	≈ 1.53680
34	1.53644 [41]	289/188	≈ 1.53723
35	1.53682 [41]	76195/49569	≈ 1.53715

Table 1.3: New lower bounds on the asymptotic competitive ratio. The second column contains the previously known bounds and the third column contains our improved lower bounds.

1.4.3 Conclusions

In this section we proved some lower bounds for different values of k for the cardinality constrained version of the online bin packing problem. Using fully adaptive construction a general online lower bound of 2 was proved in [8] together with improvements on specific values of k.

1.5 NF-based Bounded-Space Bin Packing Algorithm

1.5.1 Definitions and preliminaries

In [89] Zheng et al. considered the following surgery problem. In each day there is a uniform time interval available for an operating room to process surgical operations. Each request with a planned operation time is temporarily stored in a waiting pool. In each day, a surgery scheduler selects a subset of requests to be executed the next day. The total planned operation time of the selected requests cannot exceed the time available for the day. They modelled this problem by the one-dimensional bin packing, and developed a semi-online algorithm to give an efficient feasible solution. In their algorithm they used a buffer to temporarily store items, having a possibility to lookahead in the list. Because of the considered practical problem they investigated the 2-parametric problem, i.e. $\max_{a\in L} s(a) \leq 1/2$. In each iteration step their algorithm puts the largest items of the buffer into a new opened bin using a Next Fit-based rule (NF): packing the actual preprocessed contents of the buffer the algorithm opens a new, empty bin and – following some simple rules – puts iteratively the items of the buffer into this bin until they fit and closes the bin. This means that at most 1 bin can be open during the packing. Those algorithms that use constant number of open bins are called bounded-space algorithms. If there is at most 1 open bin, the algorithm is called NF-based online algorithm. Analysing the performance of NF-based algorithms they proved an ACR of $\frac{13}{9}$ for any given buffer size not less than 1. They also gave a lower bound of $\frac{4}{3}$ for those bounded-space algorithms that use NF-based rules.

Later, Zhang et al. also investigated the problem (see [88]). They presented two algorithms. The first one used a buffer with capacity of 2, and they proved that the ACR of the algorithm is 1.4375. Using a buffer size 3 they gave further improvement on the upper bound. Their algorithm has an ACR of 1.4243. Finally, they gave a lower bound 1.4230, which is also better than the one previously proved in [89].

In this section we deal with NF-based semi-online algorithms that use buffer with constant size based on the paper [20]. Firstly, instead of the 2-parametric problem, we investigate the parametric problem in general. It means that we consider the lists L where $\max_{a\in L} s(a) \leq \frac{1}{r}$ for a given integer $r \geq 1$. We prove lower bounds for any NF-based online algorithm with constant buffer size for the r-parametric case. Our lower bounds for the first few values of r are the following: $h_\infty(1) = 1.69103$, $h_\infty(2) = 1.42312$, $h_\infty(3) = 1.30238$. The special case $r = 2$ gives an improvement for the earlier lower bounds.

On the positive side, we present an NF-based online algorithm that considers the r-parametric problem, and uses a buffer with capacity of 3. We prove that this algorithm has ACRs that are equal to the lower bounds, so we also improve the upper bound for the case $r = 2$ to 1.42312.

We will use a sequence that was first introduced by Sylvester in [82] (1880) for the case $r = 1$, therefore, we refer to this sequence as *generalized Sylvester*

sequence. This sequence was commonly used in the bin packing area, see e.g. [12, 42, 85].
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 1.4: The first few items 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

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

Similarly, we will use the following notations.

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

The first few values of $h_\infty(r)$: $h_\infty(1) \approx 1.69103$, $h_\infty(2) \approx 1.42312$, $h_\infty(3) \approx 1.30238$. To avoid the plenty of indices – where it is not confusing – we will denote m_j^r by m_j.

1.5.2 An improved lower bound

First, we give an improvement of the lower bound of any NF-based online algorithm with given buffer size $S \geq 1$. The buffer can store arbitrary items with total size less than the size of the buffer.

Theorem 1.5.1 ([20]) *Let us consider the r-parametric case. If a buffer is given with size of $|B| = S$, then for any A – NF-based online – algorithm $R_\infty(A) \geq h_\infty(r)$.*

Proof We will construct the following instance. Let $n > 0$ be a large integer and let $k > 4$ be an integer. Then we will consider the concatenated list $L = (L_1, L_2, \ldots, L_k)$, where

- L_1 contains $n(m_k - 1)(m_1 - 1)$ items with size $\frac{1}{m_1} + \varepsilon$.
- L_i contains $n(m_k - 1)$ items with size $\frac{1}{m_i} + \varepsilon$, for $2 \leq i \leq k-1$,
- L_k contains $n(m_k - 1)$ items with size $\frac{1}{m_k - 1} - k\varepsilon$,

where ε is arbitrary small, i.e. $\frac{1}{m_k - 1} - (m_1 + k - 3)\varepsilon > \frac{1}{m_k}$.

After packing the items of the list L_1 there are at most $\lfloor \frac{S}{1/m_1 + \varepsilon} \rfloor < Sm_1$ items in the buffer. Therefore, the algorithm has to pack $n(m_k - 1)(m_1 - 1) - Sm_1$ items from the list L_1 into bins. So A needs at least

$$\frac{n(m_k - 1)(m_1 - 1) - Sm_1}{m_1 - 1} = n(m_k - 1) - \frac{Sm_1}{m_1 - 1}$$

bins to pack the elements of L_1.

Let us pack the items of L_2. We have $n(m_k - 1)$ pieces. Having packed the elements of L_2 the buffer contains $\lfloor \frac{S}{1/m_2 + \varepsilon} \rfloor < Sm_2$ elements. So, the algorithm has to pack $n(m_k - 1) - Sm_2$ items, $m_2 - 1$ pieces in each bin. So, the algorithm uses at least

$$\frac{n(m_k - 1) - Sm_2}{m_2 - 1} = \frac{n(m_k - 1)}{m_2 - 1} - \frac{Sm_2}{m_2 - 1}$$

bins. Since at most one active bin exists, therefore at least $\frac{n(m_k - 1)}{m_2 - 1} - \frac{Sm_2}{m_2 - 1} - 1$ new bins were opened while the algorithm packed the elements of L_2.

Following this train of thought while the algorithm packs the items of L_i, $3 \leq i \leq k-1$, it will open $\frac{n(m_k - 1)}{m_2 - 1} - \frac{Sm_i}{m_i - 1} - 1$ new bin for each list.

In the end the algorithm packs the items of L_k. Each bin contains $(m_k - 1)$ items from this list, so the algorithm opens $(n-1)$ new bins. Therefore, the number of bins used by the algorithm A is at least

$$A(L) \geq n(m_k - 1) + n \sum_{i=2}^{k} \frac{m_k - 1}{m_i - 1} - S \sum_{i=1}^{k} \frac{m_i}{m_i - 1} - (k-1).$$

It is easy to check that $(m_1 - 1)$ pieces of L_1, and one item from each L_i, $1 \leq i \leq k$ can be packed into one bin, therefore $\textsc{opt}(L) \leq n(m_k - 1)$. So,

$$\frac{A(L)}{\textsc{opt}(L)} \geq 1 + \sum_{i=2}^{k} \frac{1}{m_i - 1} - \frac{S \sum_{i=1}^{k} \frac{m_i}{m_i - 1} - (k-1)}{n(m_k - 1)}$$

and so,

$$R_\infty(A) \geq \lim_{k \to \infty} \lim_{n \to \infty} \frac{A(L)}{\textsc{opt}(L)} = 1 + \sum_{i=2}^{\infty} \frac{1}{m_i - 1} = h_\infty(r).$$

□

For the problem considered in [88] and [89] the given best lower bound is 1.4230, and since $h_\infty(2) = 1.423117\ldots$ our lower bound gives an improvement for the case $r = 2$.

1.5.3 The Algorithm NFFD-B3

Investigating online bounded-space algorithms in [43] a weighting function was defined for the items. Generalizing the idea we define the following weighting function.

$$W(x) = \begin{cases} x + \dfrac{1}{m_i(m_i-1)}, & \text{if } \frac{1}{m_i} < x \le \frac{1}{m_i-1} \\ \dfrac{m_i+1}{m_i}x, & \text{if } \frac{1}{m_{i+1}-1} < x \le \frac{1}{m_i} \end{cases}$$

The weight of a bin is defined as the weight of all elements in it, and generally, the weight of a set is the weight of all items in the set. It is easy to see that the following statements are true.

Fact 1.5.2

(i) $W(x)$ *is non-decreasing in* $(0, 1]$.

(ii) For $i \ge 1$, $\frac{W(x)}{x} \le \frac{m_i+1}{m_i}$ *if* $x \le \frac{1}{m_i}$,

(iii) For $i \ge 1$, $\frac{W(x)}{x} \ge \frac{m_i+1}{m_i}$ *if* $x \ge \frac{1}{m_{i+1}-1}$.

Lemma 1.5.3 ([20]) *Let us consider the* r*-parametric problem. Then any packing of a list* L*, the weight of any bin is at most* $h_\infty(r)$.

Proof Let us suppose that a bin contains the items $x_1, x_2, \ldots, x_t$, where $x_1 \ge x_2 \ge \ldots \ge x_t$. **Case A.** First we suppose that there are exactly r items from the interval $(\frac{1}{m_1}, \frac{1}{m_1-1}]$ in an arbitrary bin $\mathcal{B}$. We denote the remaining items by $p_1, \ldots, p_{t-r}$. **Case A.1.** Now, we suppose that each p_i is in the interval $(\frac{1}{m_{i+1}}, \frac{1}{m_{i+1}-1}]$ for $i = 1, \ldots, (t-r)$. Taking into account that $r = m_1 - 1$, $m_1 = m_2 - 1$, and $\frac{1}{m_i(m_i-1)} = \frac{1}{m_{i+1}-1}$ we get

$$\begin{aligned} W(\mathcal{B}) &= \textstyle\sum_{i=1}^{r} W(x_i) + \sum_{i=1}^{t-r} W(p_i) \\ &= \textstyle\sum_{i=1}^{r} x_i + \frac{r}{m_1(m_1-1)} + \sum_{i=1}^{t-r} p_i + \sum_{i=1}^{t-r} \frac{1}{m_{i+1}(m_{i+1}-1)} \\ &\le 1 + \frac{1}{m_2-1} + \textstyle\sum_{i=3}^{t-r+2} \frac{1}{m_i-1} \\ &= 1 + \textstyle\sum_{i=2}^{t-r+2} \frac{1}{m_i-1} < h_\infty(r). \end{aligned}$$

Case A.2. Let us suppose that $\mathcal{B}$ contains $s < t - r$ pieces of p_i items, each of them in the interval $(\frac{1}{m_i}, \frac{1}{m_i-1}]$, $i = 2, 3, \ldots, s+1$.

Let us denote the remaining $(t-r-s)$ items by q_l, $l = 1, 2, \ldots, t-r-s$. Then Q $= \sum_{i=1}^{t-r-s} q(i) \le \frac{1}{m_{s+2}-1}$, and $\sum_{i=1}^{r} x_i + \sum_{i=2}^{s+1} p_i \le 1 - Q$. Because of the Fact 1.5.2 (ii), we get

$$\sum_{i=1}^{t-r-s} W(q(i)) \le Q\frac{m_{s+2}+1}{m_{s+2}}.$$

Therefore

$$\begin{aligned} W(\mathcal{B}) &= \sum_{i=1}^{r} W(x_i) + \sum_{i=2}^{s+1} W(p_i) + \sum_{i=1}^{t-r-s} W(q_i) \\ &= \sum_{i=1}^{r} x_i + \frac{r}{m_1(m_1-1)} + \sum_{i=2}^{s+1} p_i + \sum_{i=2}^{s+1} \frac{1}{m_i(m_i-1)} + Q\frac{m_{s+2}+1}{m_{s+2}} \\ &\leq 1 - Q + Q\frac{m_{s+2}+1}{m_{s+2}} + \sum_{i=2}^{s+1} \frac{1}{m_i-1} \\ &= 1 + \sum_{i=2}^{s+3} \frac{1}{m_i-1} < h_\infty(r). \end{aligned}$$

Case B. Let us suppose that the bin $\mathcal{B}$ contains $q \leq r-1$ items belonging to the interval $(\frac{1}{m_1}, \frac{1}{m_1-1}]$. Since $W(x)$ is a monotone increasing function, so the weighting function is maximal if these items have maximal sizes i.e.

$$\sum_{i=1}^{q} x_i = \frac{q}{m_1 - 1}.$$

Then the remaining place in the bin is $1 - \frac{q}{m_1-1}$. We know that for any item x for which $x \leq \frac{1}{m_1}$ the weight is $W(x) \leq x\frac{m_1+1}{m_1}$.

$$\begin{aligned} W(\mathcal{B}) &= \sum_{i=1}^{q} W(x_i) + \sum_{i=q+1}^{t} W(x_i) \\ &\leq \frac{q}{m_1-1} + \frac{q}{m_1(m_1-1)} + \left(1 - \frac{q}{m_1-1}\right)\frac{m_1+1}{m_1} \\ &= 1 + \frac{1}{m_1} = 1 + \frac{1}{m_2-1} \\ &< h_\infty(r). \end{aligned}$$

□

Theorem 1.5.4 ([20]) *For any list L, $W(L) \leq h_\infty(r)\mathrm{OPT}(L)$.*

In the sequel we will call a bin *good bin* if the sum of the weights of the items in the bin is at least one, and a set of items is *good subset* if the sum of the weights of the items is greater than or equal to one and the sum of the sizes is at most one. Of course a good bin contains a good subset. We consider a buffer with capacity 3 and we will apply three virtual bins – with capacity one – for preprocessing the items in the buffer before we pack them into bin. The algorithm Next Fit with First Fit Decreasing in Buffer-length 3 (*NFFD-B3*) is the following:

(1) Fill up the buffer with the subsequent elements of the list until the next item cannot fit into the buffer.

(2) Order the items in the buffer in non increasing order, and put the items in three virtual bins – denoted by $VBIN_i$, $i = 1, 2, 3$ – each of them with capacity 1 using the FFD rule. The items that do not fit in any of the virtual bins, remain in the buffer.

(3) Check the contents of the virtual bins. For all those virtual bins that are good bins, open a new empty bin, put the items from the good bin into this new-opened bin, and close the bin. Go to step (5).

(4) Find a good subset in the contents of $VBIN_i$, $i = 1, 2, 3$, open a new empty bin, put the items from the virtual bins into this new-opened bin, and close the bin.

(5) If there is unplaced item then go to (1),

(6) Empty the contents of the virtual bins into new-opened bins. Close the bins, and quit.

We remark that we speak about *virtual bins* since after ordering the items we do not move them from the buffer into bins, but they get two indices, where the first one denotes which of the virtual bin belongs to the item, and the second signs its position within the virtual bin. (Items could not be assigned to any virtual bins that have index values 0.) The position within the virtual bin depends on the size of the item: the larger an item the smaller its position.

Let us divide the interval $(0, \frac{1}{r}]$ into subintervals as follows.

$$\begin{array}{lll} A & = (1/m_1, 1/(m_1-1)] & \\ B_i & = (1/m_i, 1/(m_i-1)] & \text{for } i \geq 2. \\ C_i & = (1/(m_i+1), 1/m_i] & \text{for } i \geq 2. \\ D_i & = (1/(m_{i+1}-1), 1/(m_i+1)] & \text{for } i \geq 2. \end{array}$$

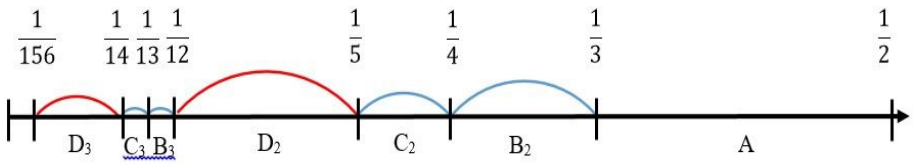

Figure 1.2: Splitting the interval $(0, \frac{1}{r}]$ into subintervals for $r = 2$.

We will call an item *X-item* if it is in the interval X, $X \in \{A, B_i, C_i, D_i\}$, $i \geq 2$. A bin is *X-homogeneous*, if it only contains X-items, and there is no space for further X-items in the bin.

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

Table 1.5: The weighting function $W(x)$ for $r = 1$.

Lemma 1.5.5 ([20]) *Any X-homogeneous bin is a good bin.*

Proof Case B_i. We remind the reader that the interval A is a B_1 interval. So, if $\mathcal{B}$ is a B_i-homogeneous bin ($i \geq 1$) then we can put $m_i - 1$ pieces of items into this bin, so the total size of the items is larger than $\frac{m_i-1}{m_i}$. Therefore

$$W(\mathcal{B}) = \sum_{a \in \mathcal{B}} s(a) + (m_i - 1)\frac{1}{m_i(m_i - 1)} > \frac{m_i - 1}{m_i} + \frac{1}{m_i} = 1.$$

Case C_i. If $\mathcal{B}$ is a C_i-homogeneous bin ($i \geq 2$) then we can put m_i pieces into the bin, so the total size of the items is larger than $\frac{m_i}{m_i+1}$. Therefore

$$W(\mathcal{B}) = \frac{m_i + 1}{m_i} \sum_{a \in \mathcal{B}} s(a) > \frac{m_i + 1}{m_i} \frac{m_i}{m_i + 1} = 1.$$

Case D_i. If $\mathcal{B}$ is a D_i-homogeneous bin ($i \geq 2$) then we can put at least m_i+1 pieces into the bin. Since we can not put further D_i-item into the bin, so the total size of the items is larger than $1 - \frac{1}{m_i+1} = \frac{m_i}{m_i+1}$. Therefore

$$W(\mathcal{B}) \geq \frac{m_i + 1}{m_i} \sum_{a \in \mathcal{B}} s(a) > \frac{m_i + 1}{m_i} \frac{m_i}{m_i + 1} = 1.$$

□

Type	Interval	$W(x)$	Type	Interval	$W(x)$
A	$(1/3, 1/2]$	$x + 1/3$			
B_2	$(1/4, 1/3]$	$x + 1/12$	B_3	$(1/13, 1/12]$	$x + 1/156$
C_2	$(1/5, 1/4]$	$5x/4$	C_3	$(1/14, 1/13]$	$13x/12$
D_2	$(1/12, 1/5]$	$5x/4$	D_3	$(1/156, 1/14]$	$13x/12$

Table 1.6: The weighting function $W(x)$ for $r = 2$.

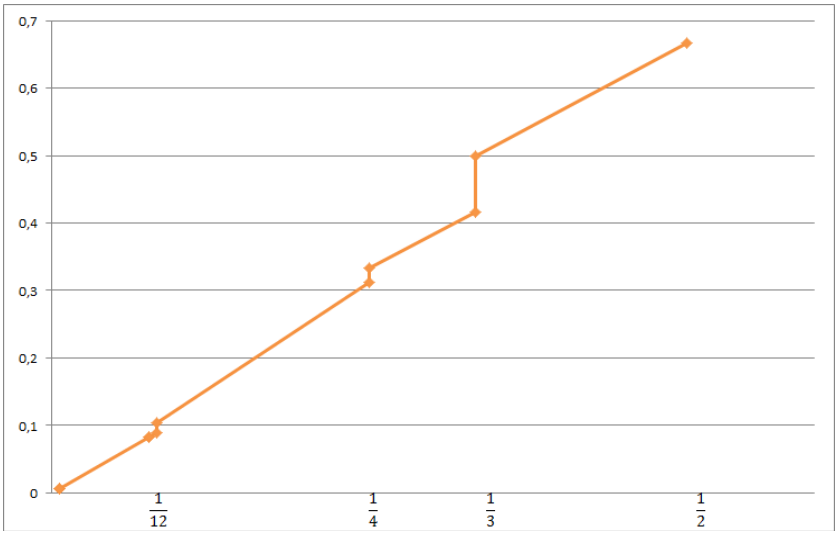

Figure 1.3: The weighting function $W(x)$ for $r = 2$.

Our main theorem is the following.

Theorem 1.5.6 ([20]) *If we pack the items of any list by the algorithm NFFD-B3 then in Step (3) we either find at least one good bin, or we find a good subset in the contents of the three virtual bins.*

Proof We will assume the contrapositive. We suppose that a list L_0 exists for which after the Step (2) the algorithm *NFFD-B3* neither produces at least one virtual bin nor good subsets can be found. We can suppose that the list L_0 has the following properties.

- The total size of the items in L_0 is $\sum_{a \in L_0} s(a) > 3 - \frac{1}{r}$.
- If $\min_{a \in L_0} s(a)$ is an X-item, then in Step (2) no further X-items can be put into any of the virtual bins.
- After Step (2) none of the virtual bins are empty.

During our proof we will distinguish different cases according to which interval the smallest item belongs to.

Lemma 1.5.7 ([20]) *If* $\min_{a \in L_0} s(a)$ *is an A-item, then* $VBIN_1$ *(and* $VBIN_2$*) is a good bin.*

Proof Let us suppose that we have at least one A-item in $VBIN_2$ or $VBIN_3$. Then $VBIN_1$ is an A-homogeneous bin. □

Corollary 1.5.8 ([20]) *If* $r = 1$ *then* $\max_{a \in L_0} s(a) \leq \frac{1}{m_1}$.

Corollary 1.5.9 ([20]) *After Step (2) neither* $VBIN_2$ *nor* $VBIN_3$ *contains A-items, and* $VBIN_1$ *contains at most* $r - 1 = m_1 - 2$ *pieces of A-item.*

Lemma 1.5.10 ([20]) *If* $\min_{a\in L_0} s(a)$ *is a* B_2*-item, then* $VBIN_1$ *is a good bin.*

Proof First we consider the case $r = 1$. Since $VBIN_1$ does not contain A-item, therefore, B_2 is the largest item in L_0. If $VBIN_2$ or $VBIN_3$ contains B_2 item then $VBIN_1$ must be a B_2-homogeneous bin. Therefore, if $r = 1$ then neither $VBIN_2$ nor $VBIN_3$ contains B_2 item.

Consider the case $r \geq 2$. Since $VBIN_2$ does not contain A-item, $VBIN_2$ is a B_2-homogeneous bin, and therefore it is a good bin. □

Corollary 1.5.11 ([20]) *If* $r = 1$ *then neither* $VBIN_2$ *nor* $VBIN_3$ *contains* B_2*-item.*

Lemma 1.5.12 ([20]) *If* $\min_{a\in L_0} s(a)$ *is a* C_2*-item, then either at least one of* $VBIN_1$ *and* $VBIN_2$ *is a good bin, or the algorithm can collect a good subset from the items in the virtual bins.*

Proof First we consider the case $r = 1$. From the Corollary 1.5.11 it follows that neither $VBIN_2$ nor $VBIN_3$ can contain A- and B_2-items. Therefore, if there is a C_2-item that does not fit in $VBIN_1$ then $VBIN_2$ must be a C_2-homogeneous bin.

Now, we can suppose that $r \geq 2$. Because of the Lemma 1.5.7 and Lemma 1.5.10, $VBIN_3$ contains only C_2-items. Since $\sum_{a\in VBIN_1} s(a)+\sum_{a\in VBIN_2} s(a) \leq 2$, and the buffer was at least to the level $\frac{3r-1}{r}$ full after Step (1), in the $VBIN_3$ there is at least $\frac{3r-1}{r} - 2 = \frac{r-1}{r}$ place which contains C_2-items only. Therefore, there are at least r pieces of C_2-items in the $VBIN_3$.

Case A. Let us suppose that we have $r - 1$ A-items in the $VBIN_1$, and we denote the total sum of the sizes of A-items and the B_2-items in $VBIN_1$ by x_A and x_{B_2}, respectively. Then

$$W(VBIN_1) = x_A + \frac{r-1}{r(r+1)} + x_{B_2} + \frac{1}{(r+1)(r+2)}.$$

Since $x_A + x_{B_2} > \frac{r+1}{r+2}$, therefore

$$W(VBIN_1) > (x_A + x_{B_2}) + \frac{r-1}{r(r+1)}\frac{1}{(r+1)(r+2)} = 1 + \frac{r-2}{r^3+3r^2+2r}.$$

Since $r \geq 2$, the right hand side is greater than 1, so $VBIN_1$ is a good bin.

Case B. Now, we suppose that there are at most $(r - 2)$ A-items in the bin $VBIN_1$. In this case at least 2 pieces of B_2-items are in the $VBIN_1$. From the Corollary 1.5.9 there is no A-item in $VBIN_2$.

Case B.1. If $VBIN_2$ contains $r-1$ pieces of B_2-items then these items together with the 2 pieces of B_2-items in $VBIN_1$ give a good subset.

Case B.2. Since $VBIN_2$ is full to at least level $\frac{r+1}{r+2}$, if it contains at most $(r - 2)$ pieces of B_2-items then the bin must contain at least 2 pieces of C_2-items. So, there are at least $r+2$ pieces of C_2-items together in $VBIN_2$ and $VBIN_3$, and so, they can form a good subset. □

Corollary 1.5.13 ([20]) *If $r = 1$ then neither $VBIN_2$ nor $VBIN_3$ contains C_2-item.*

Lemma 1.5.14 ([20]) *If $\min_{a\in L_0} s(a)$ is a D_i-item, $i \geq 2$, then $VBIN_1$ is a good bin.*

Proof When the algorithm *NFFD-B3* puts the first D_i item into $VBIN_j$, $j = 2, 3$, then $VBIN_1$ is at least $\frac{m_i}{m_i+1}$ full. By the fact 1.5.2 (iii) $W(x) \geq x(m_i + 1)/m_i$, so the total weight in the $VBIN_1$ is at least 1, so $VBIN_1$ is a good bin. □

Corollary 1.5.15 ([20]) *If $r = 1$ then neither $VBIN_2$ nor $VBIN_3$ contains D_i-item, $i \geq 2$.*

We introduce the following notations. Let $S(X^+, j)$ denote the sum of the sizes of all items in the virtual bin $VBIN_j$, $j = 1, 2, 3$, that are larger than the X-elements, where $X \in \{B_i, C_i\}$, $i \geq 3$. Furthermore, let $N(X, j)$ and $S(X, j)$ denote the number and the overall sizes of X-elements in $VBIN_j$, respectively.

Lemma 1.5.16 ([20]) *Let $\min_{a\in L_0} s(a)$ be a B_i-item, $i \geq 3$. Then the number of B_i-items in the $VBIN_1$ is*

$$N(B_i, 1) > m_i - \frac{2m_i}{m_i - m_{i-1}}.$$

Proof By the assumption L_0 is a counterexample, so – using the Fact 1.5.2 (iii) – we get

$$W(VBIN_1) = \frac{m_{i-1} + 1}{m_{i-1}} S(B_i^+, 1) + S(B_i, 1) + \frac{N(B_i, 1)}{m_{i+1} - 1} < 1. \tag{1.27}$$

Let x be the first B_i-item that has not fit into the bin $VBIN_1$. Since x did not fit into $VBIN_1$ and $x \leq \frac{1}{m_i-1}$ the total sizes of the items in $VBIN_1$ is

$$S(B_i^+, 1) + S(B_i, 1) > \frac{m_i - 2}{m_i - 1} \tag{1.28}$$

Let us eliminate $S(B_i^+, 1)$ from the inequalities (1.27)and (1.28) multiplying (1.27) by m_{i-1} and (1.28) by $-(m_{i-1} + 1)$. Adding these two inequalities we get

$$N(B_i, 1)\frac{m_{i-1}}{m_{i+1} - 1} - S(B_i, 1) < m_{i-1} - \frac{m_i - 2}{m_i - 1}(m_{i-1} + 1). \tag{1.29}$$

Since every B_i-item has size at most $\frac{1}{m_i-1}$ and there are $N(B_i, 1)$ items in $VBIN_1$, therefore $S(B_i, 1) \leq \frac{N(B_i,1)}{m_i-1}$. Substituting this into the inequality (1.29) we get

$$N(B_i, 1)\Big(\frac{1}{m_i - 1} - \frac{m_i - 1}{m_{i+1} - 1}\Big) > \frac{m_i - 2}{m_i - 1}(m_{i-1} + 1) - m_{i-1}, \tag{1.30}$$

and so

$$N(B_i, 1) > \frac{m_i - m_{i-1} - 2}{m_i - 1} \cdot \frac{m_i(m_i - 1)}{m_i - m_{i-1}} = \frac{m_i(m_i - m_{i-1} - 2)}{m_i - m_{i-1}}, \tag{1.31}$$

which yields the desired result. □

Lemma 1.5.17 ([20]) *Let* $\min_{a\in L_0} s(a)$ *be a* B_i*-item,* $i \geq 3$*, and let* $r = 1$*. If* $i = 3$ *then* $VBIN_3$ *does not contain any* B_3 *item. Furthermore, if* $i \geq 4$ *then neither* $VBIN_2$ *nor* $VBIN_3$ *contains any* B_i *item.*

Proof Let x be the first B_i-item that has not fit into the bin $VBIN_1$.

Case $i = 3$. The Lemma 1.5.16 states that $N(B_3, 1) > \frac{7}{2}$. Therefore $N(B_3, 1) \geq 4$. If there are at least two B_3 items in $VBIN_2$ and $VBIN_3$, we have at least $6 = m_3 - 1$ pieces of B_3 elements. These items can fit into one bin, so their total weight is at least 1. So, they form a good subset, which is a contradiction. Therefore, there is at most one B_3 item in $VBIN_2$ and $VBIN_3$ together. If $r = 1$ then from the Corollaries 1.5.8, 1.5.11, 1.5.13, 1.5.15 follows that in $VBIN_2$ and $VBIN_3$ a B_3-item is the largest item. So, the B_3-item must be placed in $VBIN_2$.

Case $i \geq 4$. We know that $\frac{2m_i}{m_i - m_{i-1}} \leq 3$. Therefore $N(B_i, 1) \geq m_i - 2$. Together with x these $m_i - 1$ pieces of B_i items form a good subset, which is a contradiction. □

From the Lemma 1.5.17 it follows that if $\min_{a\in L_0} s(a)$ be a B_i-item, $i \geq 3$, then $VBIN_3$ is empty. Since the total size of the items in L_0 is $\sum_{a\in L_0} s(a) > 3 - \frac{1}{r}$, so this results in a contradiction again.

Lemma 1.5.18 ([20]) *Let* $\min_{a\in L_0} s(a)$ *be a* B_i*-item,* $i \geq 3$*, and let* $r \geq 2$*. Then neither* $VBIN_2$ *nor* $VBIN_3$ *contains* B_i *item.*

Proof Let x be the first B_i-item that has not fit into the bin $VBIN_1$. Now, for every $i \geq 3$ we get $\frac{2m_i}{m_i - m_{i-1}} < 3$, and therefore $N(B_i) \geq m_i - 2$. So, we can use the proof of Case $i \geq 4$ of the Lemma 1.5.17. □

From the Lemma 1.5.17 and Lemma 1.5.18 follows that if $\min_{a\in L_0} s(a)$ be a B_i-item, $i \geq 3$, then $VBIN_3$ is empty. Since the total size of the items in L_0 is $\sum_{a\in L_0} s(a) > 3 - \frac{1}{r}$, so this results in a contradiction again. Therefore it is not possible that $\min_{a\in L_0} s(a)$ is a B_i-item, where $i \geq 3$.

Lemma 1.5.19 ([20]) *If* $\min_{a\in L_0} s(a)$ *is a* C_i*-item,* $i \geq 3$*, then either* $VBIN_2$ *is a good bin, or the algorithm can collect a good subset from the* C_i*-items in the virtual bins.*

Proof Let x be the first C_i-item in $VBIN_3$. If there is no larger item in L_0 then x then both virtual bins, $VBIN_1$ and $VBIN_2$, are C_i-homogeneous bins. So, we can suppose that there are larger items in $VBIN_1$ and $VBIN_2$.

Since L_0 is a counterexample, using the Fact 1.5.2 (iii) for the total weight of the items in the $VBIN_2$ we get

$$\frac{m_{i-1} + 1}{m_{i-1}} S(C_i^+, 2) + S(C_i, 2) \frac{m_i + 1}{m_i} < 1. \tag{1.32}$$

Since x did not fit into $VBIN_2$ and $x \leq \frac{1}{m_i}$ the total size of the items in $VBIN_2$

$$S(C_i^+, 2) + S(C_i, 2) > \frac{m_i - 1}{m_i} \tag{1.33}$$

Let us eliminate $S(C_i^+, 2)$ from the inequality system (1.33) and (1.32) by multiplying 1.33 by $-(m_{i-1}+1)$ and 1.32 by m_{i-1}. Adding these two inequalities we get

$$S(C_i, 2) > \frac{m_i - m_{i-1} - 1}{m_i - m_{i-1}} = 1 - \frac{1}{m_i - m_{i-1}}. \tag{1.34}$$

Since every C_i-item has size at most $\frac{1}{m_i}$ and there are $N(C_i, 2)$ items in $VBIN_2$, therefore $N(C_i, 2) > m_i - \frac{m_i}{m_i - m_{i-1}} = m_i - 1 - \frac{m_{i-1}}{m_i - m_{i-1}} > m_i - 2$. So, there are at least $m_i - 1$ pieces of C_i items in $VBIN_2$, and at least one C_i-item in $VBIN_3$. The sum of the sizes of these items is at most one, and the total weight of these items is at least one. So, we can construct a good subset again. □

With the help of the above Lemmas we have shown that if L_0 is a counterexample, i.e. packing the items of L_0 by the algorithm *NFFD-B3* there is neither a good bin among the virtual bins nor a good subset, then the smallest item of L_0 may not be in any of the interval X, where $X \in \{A, B_i, C_i, D_i\}$, $i \geq 2$. Therefore L_0 must be an empty list, which completes the proof of the Theorem 1.5.6. □

The algorithm *NFFD-B3* opens at least one bin in each iteration step, fills up the new opened bin(s) with items with total weight of at least one, and closes it. At the end of the list the content of the last three virtual bins will be added to the number of used bins. So the following theorem is valid.

Theorem 1.5.20 ([20]) *Let r be a positive integer, and L_r be an arbitrary list with items $s(a_i) \leq \frac{1}{r}$. Let us pack the items of L by the algorithm NFFD-B3. Then*

$$NFFD\text{-}B3(L) \leq W(L) + 3.$$

Theorem 1.5.1 and Theorem 1.5.20 and Corollary 1.5.4 together yield that

$$R_\infty(NFFD\text{-}B3) = h_\infty(r).$$

Our algorithm needs maximum $O(n \log n)$ operations to order the contents of the buffer, and $O(n)$ operations to find a good subset among the items in the buffer in each iteration step. Since the number of iterations is maximum $O(n)$, the time-complexity of the algorithm is $O(n^2 \log n)$.

1.5.4 Conclusions

In two earlier papers a bounded-space semi-online bin packing problem was considered. In papers [88] and [89] lower and upper bounds were given for the online algorithms for this problem. Both of the papers investigated the case $r = 2$. The best lower- and upper-bounds were 1.4230 and 1.4243, respectively.

In this section we defined an algorithm with asymptotic competitive ratio of $h_\infty(r)$ for any $r \geq 1$ integer. We also proved that these upper bounds are tight for every r. Especially, for $r = 2$ this value is $h_\infty(2) = 1.423117\ldots$.

Chapter 2

Scheduling Problems

2.1 Introduction

In case of scheduling problems, we need to plan the execution of specific jobs on machines. In the general model, jobs can consist of several activities and for them we need to give the machines and time intervals in which each operation is to be performed. Scheduling models can be applied in such diverse areas as economics, transportation, industrial production and many others. More formally the problem can be defined as follows: there are m machines that are used to process n jobs. For each machine $i(i = 1, 2, \ldots, m)$ and each job $j(j = 1, 2, \ldots, n)$, a schedule specifies one or more time intervals throughout which processing is performed on j by i. A schedule is considered feasible if there is no overlapping of time intervals corresponding to the same job and there is no overlapping of time intervals corresponding to the same machine. More requirements can be given related to the problem type. The type can be specified by the machine environment, the job characteristics and the optimality criterion. In the simplest case each job requires one operation. Considering the machine environment single and parallel machine problem versions are known. If there are more machines, the processing times can be independent of the machine (identical parallel machines). The machines can operate at different speeds (uniformly related parallel machines) or the processing time of a job can depend completely on the machine (unrelated parallel machines). In the more complicated variants the jobs may have more phases which can be executed on one or more machines based on several criteria. In addition to the processing requirements the jobs may have further characteristics, like availability for processing, precedence constraints or interruption conditions. The objectives can be also varied, an important measure is some function of the completion times, but flow time, lateness, earliness or other measures can be considered in different models. The classical objective is to minimize the makespan, where the makespan is defined as the maximum load over all machines. In this chapter first we present lower bounds for online makespan minimization for a small number of uniformly related machines. In the next two sections we investigate the unit execution time version of the coupled task scheduling problem.

2.2 Lower Bounds for Related Machines

2.2.1 Definitions and preliminaries

The instance of this problem consists of a sequence of machines with possibly different speeds and a sequence of jobs specified by their processing times. A *schedule* assigns each job to one of the machines; the time needed to process a job is equal to its processing time divided by the speed of the machine where it is assigned. The objective is to minimize the *makespan* (also called the length of the schedule, or the maximal completion time). Usually a schedule also needs to specify the timing of each job (its starting and completion times) so that the jobs on each machine do not overlap. Due to the simplicity of the problem we consider, this is not necessary and it is sufficient to specify the assignment to the machines, silently assuming that each job is started as soon as all the previous jobs on its machine are processed. Instead of calculating the completion times individually for each job, we can calculate the completion time of each machine as the total processing time of the jobs allocated to it divided by the speed of the machine; the makespan is then the maximum of the completion times over all machines.

In the online version of the problem, jobs appear online *one-by-one*. When a job appears, an online algorithm has to make an irrevocable decision and assign the job to a machine. This decision is permanent and made without the knowledge of the future jobs; the algorithm is not even aware of whether any future jobs exist or not. An online algorithm is *R-competitive* if for each instance it produces a schedule with makespan at most R times the optimal makespan.

First we present lower bounds of 2.141391 for $m = 4$ and 2.314595 for $m = 5$. The construction is based on an instance where the processing times are a geometric sequence, similarly as in previous works [24, 39, 37]. The speeds are chosen so that any online algorithm can use only two fastest machines, called the active machines. The bound is then obtained by carefully analyzing the possible patterns of scheduling the jobs on these machines.

Generalizing this to larger values of m, we use computer search for elimination of possible patterns and give instances with up to 5 active machines ([53]).

It is known that for two and three machines, the tight bounds are 1.618 and 2 respectively. Two other lower bounds for a small number of machines were 2.2880 for $m = 6$ and 2.4380 for $m = 9$ by [24]. For an arbitrary (large) number of machines, the current lower bound is 2.5648 [37].

Naturally, the lower bounds need to be compared to the existing algorithms. For a small number of machines the best currently known algorithm is the greedy List Scheduling (LS). Here List Scheduling is defined so that the next job is always scheduled so that it will finish as early as possible. Its competitive ratio for $m = 2$ is exactly $\phi \approx 1.618$, the golden ratio, and for $m \geq 3$ it is at most $1+\sqrt{(m-1)/2}$; this bound is tight for $3 \leq m \leq 6$ [30]. Moreover, for $m = 2, 3$ it can be checked easily that there is no better deterministic algorithm. For $m = 2$ it is possible even to give the exact optimal ratio for any speed combination and it is always achieved by greedy List Scheduling [38]. Already for three machines, it is not known exactly for which speed combinations List Scheduling is optimal, even though we know it

is optimal in the worst case. Some recent progress is reported in [27, 48]. Another special case when some partial results about optimality of List Scheduling are known is the case when $m-1$ machines have the same speed, see, e.g., [48, 65].

For an arbitrary (large) number of machines, the greedy algorithm is far from optimal: its competitive ratio is $\Theta(\log m)$ [3]. The first constant-competitive algorithm for non-preemptive scheduling on related machines was developed in [3]. The currently best algorithms are $3+\sqrt{8} \approx 5.828$ competitive deterministic and 4.311 competitive randomized one [24]. For an alternative very nice presentation see [13]. All these algorithms use doubling, i.e., strategies that work with some estimate of the optimal makespan and when it turns out that the estimate is too low, it is multiplied by 2 or some other constant. While this is a standard technique for obtaining a constant competitive ratio, it would be surprising if it led to optimal algorithms. Designing better algorithms both for small and large number of machines remains one of the central open problems in this area.

2.2.2 Combinatorial lower bounds

We number the machines as well as the jobs from 0 (to obtain simpler formulas). Thus we have machines $M_0, M_1, \ldots, M_{m-1}$ and jobs $J_0, J_1, \ldots, J_{n-1}$. The speed of machine M_i is denoted s_i; we order the machines so that their speeds are non-increasing. The processing time of job J_j is denoted p_j; thus the job takes time p_j/s_i to be processed on M_i.

For a given sequence of jobs $\mathcal{J}$, let $\mathcal{J}_i$ be the set of indices of jobs scheduled on machine M_i. The completion time of the machine is then simply the sum of processing times of the jobs scheduled to the machine divided by its speed: $C_i = \frac{1}{s_i}\sum_{j\in\mathcal{J}_i} p_j$. We compare the maximum completion time in the output of the algorithm with the maximum completion time of the optimal schedule.

In all our lower bounds, we let $\mathcal{J}$ denote an infinite sequence of jobs $j_0, j_1, \ldots$, where j_i has processing time $p_i = \alpha^i$ for some $\alpha > 1$. We shall consider the algorithm's assignment of jobs for the prefixes of $\mathcal{J}$; we denote the prefix of $\mathcal{J}$ consisting of the first $t+1$ jobs $j_0, j_1, \ldots, j_t$ by $\mathcal{J}[t]$.

We first briefly review the proof of the tight lower bound of 2 for $m = 3$, to introduce the main ideas.

The machine M_0 has speed 1, the machines M_1 and M_2 have speed 1/2. We set $\alpha = 2$, i.e., $p_i = 2^i$. We observe that the optimal makespan for $\mathcal{J}[t]$ is $p_t = 2^t$: j_t is scheduled on M_0, j_{t-1} on M_1 and all the remaining jobs on M_2. Assume we have an algorithm with a competitive ratio smaller than 2. If a job j_t is scheduled on one of the slow machines, the makespan on $\mathcal{J}[t]$ is $2p_t$, twice the optimum. Thus the algorithm schedules all jobs on M_0. But then the makespan on $\mathcal{J}[t]$ is $2p_t - 1$, by taking t sufficiently high we get a contradiction again.

To obtain a lower bound of R for more machines, we arrange the instance so that an R-competitive algorithm is forced to schedule all jobs on the k fastest machines (instead of one for $m = 3$), for some k. With $k = 2$ we obtain combinatorial bounds for $m = 4, 5$ while for larger values of k we use computer search and obtain bounds for larger k. The value of α will be chosen later separately for each case.

The machines $M_0, M_1, \ldots, M_{k-1}$ are called *active* machines; we set the speed of the active machine M_i to $s_i = \alpha^{-i}$. All the remaining machines, i.e., M_k, M_{k+1}, $\ldots$, M_{m-1} are called *inactive* and have the same speed s_k. This speed is chosen so that:
(A) $s_k \leq 1/R$ and
(B) for any t, the optimal makespan for $\mathcal{J}[t]$ equals p_t.

The condition (A) limits the possible lower bound on R and needs to be verified separately in each case. As we shall see later, for some values of α, k and m it gives the tightest bound on R, while for other values not. In any case, it is desirable to choose s_k as small possible, so that we have a chance of proving a large bound on R.

Thus we investigate what the smallest possible speed s_k is such that (B) holds, and it turns out that a simple argument gives an exact bound. Due to our choice of speeds of active machines, the k largest jobs of $\mathcal{J}[t]$ fit on the k active machines so that they all complete exactly at time p_t. Thus it is necessary that all the remaining jobs together do not overflow the capacity of all the inactive machines. With this in mind, we set the speeds

$$\begin{aligned} s_k &= s_{k+1} = \cdots = s_{m-1} \\ &= \alpha^{-k} \cdot \max\left\{1, \frac{1}{m-k} \sum_{i \geq 0} \alpha^{-i}\right\} \\ &= \alpha^{-k} \cdot \max\left\{1, \frac{\alpha}{(m-k)(\alpha-1)}\right\} \quad , \end{aligned} \tag{2.1}$$

and in Lemma 2.2.1 we prove that this choice of s_k indeed guarantees that (B) holds.

Lemma 2.2.1 ([53]) *The optimum makespan for $\mathcal{J}[t]$ is $p_t = \alpha^t$.*

The lower bound proofs in each case continue by examining the possible patterns of scheduling jobs on the active machines. Intuitively it is clear that the best algorithm uses an eventually periodic pattern, which then matches the lower bound. For the actual proof we proceed by gradually excluding more and more patterns. For $k = 2$ active machines this requires only a few steps and thus can be performed by hand. For larger k we use computer search.

We note that our lower bounds, both combinatorial and computer-assisted ones, are optimized in the following sense: For every choice of α, assuming the speeds and the jobs in the instance are as above, and assuming that R equals our lower bound, there either exists a periodic pattern so that the sequence can be scheduled on the active machines with makespan at most R times the optimum or the speed of the inactive machines is larger than $1/R$ and thus the algorithm can use them.

We give improved lower bounds for $m = 4$ and $m = 5$ machines. In both cases we use only $k = 2$ active machines. However, having three inactive machines instead of only two allows us to use smaller α and consequently obtain stronger lower bound for five machines.

Theorem 2.2.2 ([53]) *Let $\alpha \approx 1.72208$ be the largest real root of $z^4-z^3-z^2-z+1$. For makespan minimization on $m = 4$ uniformly related machines there exists no online algorithm with competitive ratio smaller than*

$$R = \frac{\alpha^4}{\alpha^3 - 1} = 1 + \frac{\alpha(\alpha+1)}{\alpha^3 - 1} \approx 2.141391 \ . \tag{2.2}$$

Theorem 2.2.3 ([53]) *Let $\alpha \approx 1.52138$ be the only real root of $z^3 - z - 2$. For makespan minimization on $m = 5$ uniformly related machines there exists no online algorithm with competitive ratio smaller than $R = \alpha^2 \approx 2.314595$.*

2.2.3 Computer search based lower bounds

The lower bounds described in [24] were found via search through possible assignments of jobs to $k = 3$ active machines. We present a generalization of their computer search technique in order to find the exact pattern of allocations that maximizes the lower bound obtainable for $k = 3, 4, 5$ active machines. It also confirms the previous section's results for $k = 2$ active machines.

First, we describe our technique. Fix some competitive ratio R that we want to achieve. We maintain a vector $S = (S_0, \ldots, S_{k-1})$ that tracks the relative load on each of the active machines. Now, we build a graph with states representing all such vectors such that $S_i \leq R\alpha^{-i}$ and edges being the possible vectors after a single job is scheduled. Our initial state is $(0, \ldots, 0)$. To get the relative load after a job scheduled on machine i on a given state S, we first divide each entry of our relative load vector S by α and then add 1 to the i^{th} entry of our vector. So, we get up to k edges out of each vertex.

This is the same infinite graph as the one considered by [24], though they only considered $k = 3$ active machines. We also consider all possible competitive ratios R, whereas they always set R to equal α^k. Clearly, there is an R-competitive deterministic scheduling algorithm if and only if there is an infinite path in the graph starting from the initial state. In order to make this graph computer searchable, Berman et al. made the graph finite by discretizing the S vector. This, however, led to large rounding errors except for large choices of n, the factor of discretization. So, we use a different method for which it is sufficient to search only a part of the graph.

Rather than building the entire graph, we only build the tree generated by scheduling r jobs for a small choice of r. Normally, this would generate a tree with k^r states, but since we do not include states with $S_i > R\alpha^{-i}$, many branches of the tree are pruned. From this graph, we can determine if there is a path of length r in the infinite graph. Thus, for a given choice of r, α, and k, if there is no path of length r, then there is no deterministic scheduling algorithm with competitive ratio R, giving us a lower bound.

As we are proving a lower bound of R, the common speed of the inactive machines has to be at most $1/R$, or else it would be possible to schedule at least one job on the inactive machines. Hence, by Lemma 2.2.1, we have that the number of inactive machines, $m - k$, is no less than $(R/\alpha^k) \sum_{i \geq 0} \alpha^{-i} = R \cdot \alpha^{1-k}(\alpha - 1)^{-1}$.

So, this limits our choice of α to a certain range for a given combination of k and m. We maximize the choice of R over this range to obtain our lower bound.

We can also calculate the pattern that the online algorithm can follow to achieve a competitive ratio close to our lower bound. We select an R slightly larger than the maximum R for which the graph was finite. Then, any sufficiently long path is guaranteed to follow an optimal pattern; in principle there could be several optimal patterns, actually we did not even prove that there exists one. However, it is sufficient for us to find a single one matching the lower bound, since any such pattern shows that our lower bound cannot be improved further—and we have done this for each m for the optimal value of α.

As we saw in the analysis of $m = 4$ and $m = 5$, such pattern does not necessarily make the completion times of all machines equal, i.e., for some i the i-th entry might be much smaller than α^{k-i}. While for our choice of α all paths in the tree will follow the pattern after some point, there is some flexibility in the allocation of the first few jobs, as they are very small. As we are using DFS to find a single long path in the tree in order to find the optimal pattern, our search is likely to find the pattern after inspecting only a small number out of many feasible initial allocations. After finding a cyclic pattern of length ℓ, it is simple to check if it attains the desired ratio by inspecting the relative load vector that it yields. Our method works for values of k up to 5. We are unable to find lower bounds and matching patterns for $k \geq 6$, as the tree becomes too large.

As described, we are able to verify for a given k, m, α and R if there is an R-competitive algorithm for our sequences. For the optimal values of α, it is feasible to find R by binary search. To speed up computations, we have not searched the whole tree, but only a random sample of the initial branches. This works in practice, since if there is an infinite branch and R is not too close to the real bound, most initial branches can be extended. If an infinite branch is found, it is proven that the bound is smaller. If none is found, we first verify the result by extending the random sample and then we have verified the results by a complete search for selected values of α.

Our computer search results are presented in Table 2.1 and, in more detail, in Figures 2.1, 2.2 and 2.3. The data in Table 2.1 is presented as follows. First, the number of active machines, k, then the optimal number of machines, m, then the approximation to the optimal value of α. This is followed by the optimal pattern, presented as follows. The integer i denotes M_i, the i^{th} fastest machine (counting from 0), and the pattern repeatedly assigns jobs to the machines in the order given. The next column gives the number h of the machine M_h that attains the highest load for this pattern and this value of α. Finally, the last column gives the value of the lower bound ratio R given by the computer search and matched by the pattern found.

Figure 2.1 shows the bounds for all values of m and k. For each value of α, the largest lower bound R is displayed. The values of m and k are implicit, as given by the condition (A) and setting of the speeds. Most important, for a given α and R, k is the smallest number such that $\alpha^k \geq R$, since $s_k \geq \alpha^{-k}$ and (A) would be violated otherwise. Similarly, since $(m-k)s_k \geq \alpha^{1-k}/(\alpha-1)$, the number of

k	m	α	Pattern	h	R
2	4	1.722081	001	1	2.141391
2	5	1.521380	001	0	2.314595
3	6	1.450217	001021001020100102010010201	2	2.347312
3	7	1.346256	0010201012	1	2.439957
4	8	1.346256	0010201012	1	2.439957
4	9	1.255564	010210301201021030120 1023	2	2.462775
5	10	1.222412	010321041230012013021041023012	0	2.483120
5	11	1.209132	010213020140312010230412010321 ... ...040120310210340120132010423 010210341020130120412031020134	4	2.502672

Table 2.1: The results of the computer search for lower bounds, together with cyclic patterns that attain upper bounds slightly larger than these lower bounds ([53]).

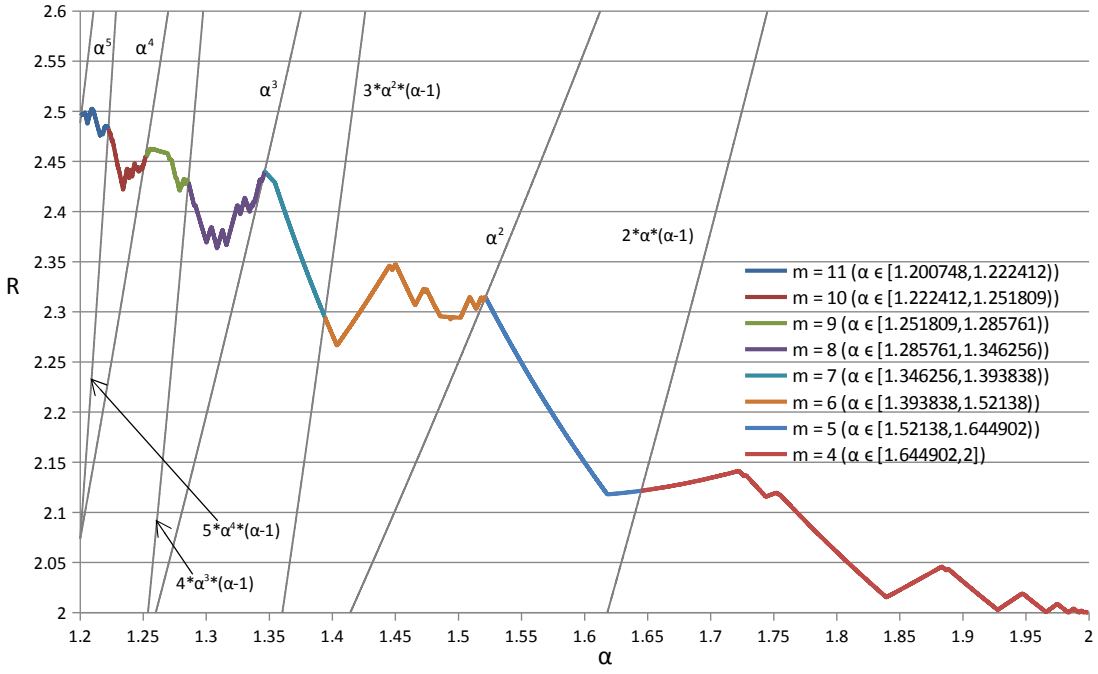

Figure 2.1: The graph presents what lower bounds our strategy yields for $\alpha \in (1.2, 2.0)$.

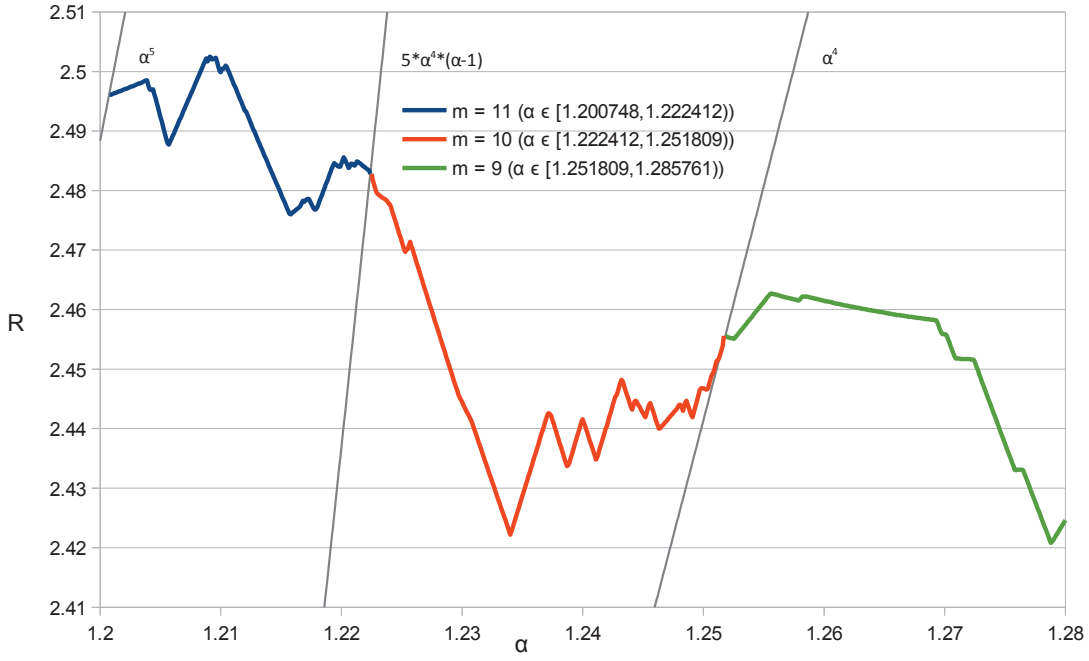

Figure 2.2: A zoom-in on the left part of Figure 2.1, i.e., $\alpha \in (1.20, 1.28)$.

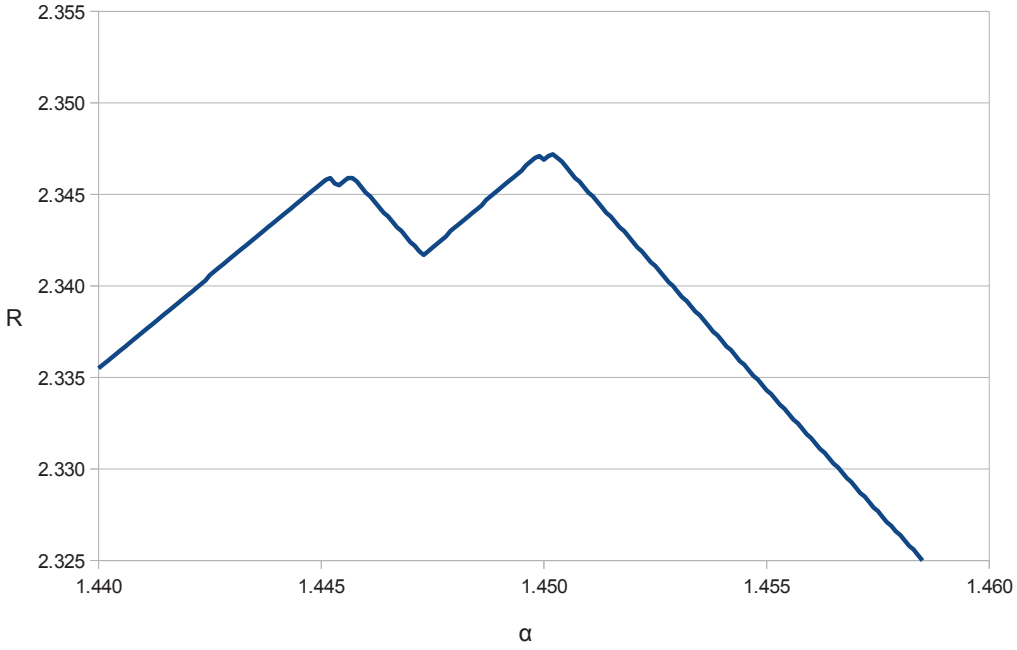

Figure 2.3: A zoom-in on the center part of Figure 2.1, i.e., $\alpha \in (1.44, 1.46)$.

inactive machines $(m-k)$ needs to satisfy $(m-k)\alpha^{k-1}(\alpha-1) \geq R$. These bounds on R are drawn as thin, fast increasing functions. As α decreases, the number of active or inactive machines increases once one of these curves is crossed. The largest values of α correspond to $m = 4$ (and $k = 2$), but as α decreases, the curve for R crosses $1/s_2 = 2\alpha(\alpha - 1)$, so an additional inactive machine is required for smaller α. Hence at that point the region corresponding to $m = 5$ and $k = 2$ starts, and extends to the left until the R curve crosses $1/s_2 = \alpha^2$; at that point an active machine is added and the region corresponding to $m = 6$ and $k = 3$ starts, and so on.

The patterns found for $k = 2$ match the patterns found in our analysis of $m = 4, 5$. The lower bound we found for $k = 3$ is slightly better than the previous lower bound found by computer search for 3 active machines (2.438).

In general, the lower bounds increase with increasing number of machines. However, as the figures show, the exact dependence on α is complicated.

While the optimal R is at most α^k in general, sometimes it is strictly smaller: This is the case for all even $m \leq 10$, including $m = 4$, for which we gave a combinatorial proof. Thus, trying all possible values of R, rather than fixing it at α^k (as [24] did) allowed us to obtain new bounds for even k. Additionally, it is interesting that for $m = 9$ and $m = 11$, the optimal R is strictly smaller than α^k, leading to improved bounds for odd m as well. Another anomaly is that for $m = 8$, the best bound matches the bound for $m = 7$. Thus, it does not always help to add an active machine without an accompanying inactive machine.

2.2.4 Conclusions

We have presented new lower bounds for online makespan scheduling on a small number of uniformly related machines. For small m we were able to take an advantage of the combinatorial structure of the problem and of the fact that even for the optimal online algorithm (pattern), the completion times of the active machines will be uneven.

2.3 Improved Analysis of an Algorithm for the CTP

2.3.1 Definitions and preliminaries

The coupled task problem (CTP) is defined as follows: we are given n jobs each of them consisting of two sub-tasks. The two sub-tasks have to be executed in a given sequence and there is an exact delay time (gap) to be observed between their execution. We specify job i, $i = 1, \ldots, n$, by a triple (a_i, l_i, b_i) of positive integers 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 and other jobs can be processed in this time interval. The aim is to schedule the n jobs on a single machine in such a way that no two tasks overlap and the latest completion time of a job

(makespan) is as small as possible. Preemptions are not allowed. In this general version the problem is strongly NP-hard. For the general case, we will use the standard three-field notation – introduced by Graham et al. [44] – as follows: $1|\mathit{Coup\text{-}Task}, \mathit{exact}\ l_i|C_{\max}$.

The general problem was first studied by Shapiro [80]. He discussed practical applications and gave three simple heuristics. The heuristics were not analyzed, but experimental results showed their effectiveness. Orman and Potts [67] examined the complexity of several special cases of the problem defined by additional conditions on the numbers a_i, l_i and b_i. In [2] a dynamic programming algorithm for the special case where all n jobs have identical specifications is presented. This problem is of special interest since its complexity is still unknown.

Yu et al. [87] proved that the problem remains NP-hard even if the jobs have unit execution time (UET problem). So, research turned on the investigation of approximation algorithms.

Let $S(n)$ be a given list of n job pairs and let A be an approximation algorithm. We will denote by $C_{\max}(\mathrm{A}, S(n))$ and $C_{\max}(\mathrm{OPT}, S(n))$ the makespan produced by algorithm A and of the optimal schedule for $S(n)$, respectively. Algorithm A is called *ρ-approximation algorithm* ($\rho \geq 1$) if

$$C_{\max}(\mathrm{A}, S(n)) \leq \rho\, C_{\max}(\mathrm{OPT}, S(n))$$

for every list of jobs. The smallest possible such value ρ is called *worst-case (performance) ratio* of algorithm A and is denoted by ρ_{A}.

Ageev and Baburin [1] defined the following algorithm for the case of unit execution times, i.e., for the case $a_i = b_i = 1$, for $i = 1, \ldots, n$.

Algorithm DNF

(1) Sort the jobs according to non-decreasing idle times, i.e., $l_1 \leq l_2 \leq \ldots \leq l_n$.

(2) Construct schedule σ_1 as follows:

- **(2.1)** Start job 1 at position 0.
- **(2.2)** For $i = 2, \ldots, n$: start job i at the earliest possible position after the first task of the previously scheduled job is finished.

(3) Let k be the number of jobs which terminate completely before the start of job n.

(4) Construct schedule σ_2 as follows (only if $k > 0$):

- **(4.1)** Start job $k+1$ at position 0.
- **(4.2)** For $i = k+2, \ldots, n, 1, \ldots, k$: start job i at the earliest possible position after the first task of the previously scheduled job is finished.

(5) Output schedule σ which is the better one of σ_1 and σ_2.

So basically, this algorithm consists of performing two times a greedy algorithm according to a next fit rule for the first sub-tasks, once for the list $(1, \ldots, n)$ and once for the list $(k+1, \ldots, n, 1, \ldots, k)$. Therefore we call it *Double Next Fit (DNF)*.

The following theorem is proven in [1].

Theorem 2.3.1 (Ageev, Baburin, [1]) *The worst-case performance ratio of DNF satisfies*

$$\rho_{\mathrm{DNF}} \leq \frac{7}{4}.$$

For analyzing the tightness of this competitive ratio, Ageev and Baburin constructed the following problem instances. Let $n \geq 3$ be odd and $t = \frac{n-1}{2}$. Define the unit execution time problem series $S_{\mathrm{AB}}(n)$ by setting

$$l_i = n-2, \quad i = 1, \ldots, n-t,$$
$$l_{n-t+r} = \frac{3n-7}{2} + r, \quad r = 1, \ldots, t.$$

It is easy to see that for this problem

$$C_{\max}(\mathrm{DNF}, S_{\mathrm{AB}}(n)) = C_{\max}(\sigma_1, S_{\mathrm{AB}}(n)) = C_{\max}(\sigma_2, S_{\mathrm{AB}}(n)) = \frac{1}{2}(7n-5),$$

where $C_{\max}(\sigma_1, S_{\mathrm{AB}}(n))$ and $C_{\max}(\sigma_2, S_{\mathrm{AB}}(n))$ denote the makespans of schedules σ_1 and σ_2. For any list $S(n)$ a trivial lower bound on the optimum makespan of a UET problem is

$$C_{\max}(\mathrm{OPT}, S(n)) \geq \max\bigl\{2n, 2 + \max\{l_i \mid i = 1 \ldots, n\}\bigr\}.$$

In [1] the authors used $2n$ as lower bound and thus came to the conclusion that problem series $S_{\mathrm{AB}}(n)$ would prove the $\rho_{\mathrm{DNF}} = \frac{7}{4}$.

2.3.2 An improved lower bound

We will now derive a better lower bound for UET problems and thus show that the above series does not yield the tightness of the bound $\frac{7}{4}$ (see [23]).

Theorem 2.3.2 ([23]) *Suppose that* $\sum_{i=1}^{n} l_i > n(n-1)$. *Then, for any list* $S(n)$

$$C_{\max}(\mathrm{OPT}, S(n)) \geq 2n + \Bigl\lceil \frac{1}{n}\bigl(\sum_{i=1}^{n} l_i - n(n-1)\bigr) \Bigr\rceil.$$

Proof Consider an arbitrary schedule of n jobs. For each position of the schedule count the number of jobs for which this position falls into their gap. Let O_T denote the total sum of the overlap counts over all positions where the machine is busy and O_I denote the sum over all idle positions.

We get an upper bound on O_T if we simply consider a schedule of length $2n$, i.e., without idle positions. The first position of the schedule cannot be contained in any gap. The second position can be contained in at most one gap. Continuing this idea we get that the m-th position of the schedule can be contained in at most $m-1$ gaps, for $m = 3, \ldots, n$. We get the same for the second n positions by scanning the schedule from right to left. So, the overlap sum O_T is at most $n(n-1)$.
Obviously $O_T + O_I = \sum_{i=1}^{n} l_i$, and therefore $O_I \geq \sum_{i=1}^{n} l_i - n(n-1)$.

If $\sum_{i=1}^{n} l_i > n(n-1)$, then the schedule must have p idle positions, $p > 0$. Since every idle position can contribute at most n to I_s, we get that

$$p \geq \left\lceil \frac{1}{n} \left(\sum_{i=1}^{n} l_i - n(n-1) \right) \right\rceil.$$

□

Lemma 2.3.3 ([23]) *For problem series* $S_{\mathrm{AB}}(n)$

$$\lim_{n\to\infty} \frac{C_{\max}(\mathrm{DNF}, S_{\mathrm{AB}}(n))}{C_{\max}(\mathrm{OPT}, S_{\mathrm{AB}}(n))} \leq \frac{28}{19}.$$

Proof By definition we obtain

$$\begin{aligned} \sum_{i=1}^{n} l_i &= (n-t)(n-2) + t\,\frac{3n-7}{2} + \frac{t(t+1)}{2} \\ &= \frac{11n^2 - 24n + 5}{8} \end{aligned}$$

and therefore

$$\sum_{i=1}^{n} l_i - n(n-1) = \frac{3n^2 - 16n + 5}{8}.$$

For $n \geq 5$ the condition of Lemma 2.3.2 holds and we get that

$$C_{\max}(\mathrm{OPT}, S_{\mathrm{AB}}(n)) \geq 2n + \left\lceil \frac{3n^2 - 16n + 5}{8n} \right\rceil \geq \frac{19n^2 - 16n + 5}{8n}.$$

From these bounds we obtain for problem series $S_{\mathrm{AB}}(n)$

$$\lim_{n\to\infty} \frac{C_{\max}(\mathrm{DNF}, S_{\mathrm{AB}}(n))}{C_{\max}(\mathrm{OPT}, S_{\mathrm{AB}}(n))} \leq \lim_{n\to\infty} \frac{28n^2 - 20n}{19n^2 - 16n + 5} = \frac{28}{19}.$$

□

So, the result in [1] actually is $\frac{28}{19} \leq \rho_{\mathrm{DNF}} \leq \frac{7}{4}$.

In fact, using a time-indexed integer programming approach [22] we were able to compute the true optimum solutions for odd $n \leq 57$. In every case the optimum solution was equal to our improved lower bound.

2.3.3 Tightness of the upper-bound

However, by constructing a different example we will show that the worst-case bound $\frac{7}{4}$ for algorithm DNF is indeed tight.
Let $n = 4m + 2$, for $m \geq 2$. We define the problem series $S_I(n)$ by setting

$$\begin{aligned} l_i &= 2m - 1, \quad i = 1, \ldots, 2m - 1, \\ l_{2m} &= 4m, \\ l_i &= 4m + 1, \quad i = 2m + 1, \ldots, 4m, \\ l_{4m+2} &= 8m + 1. \end{aligned}$$

Lemma 2.3.4 ([23]) *For problem series $S_I(n)$*

$$C_{\max}(\mathrm{DNF}, S_I(n)) = \frac{7n - 6}{2}.$$

Proof It is easy to see that

$$C_{\max}(\sigma_1, S_I(n)) = n + m + l_n + 1 = n + \frac{n - 2}{2} + 2n - 3 + 1 = \frac{7n - 6}{2}$$

and that $k = 2m$.
For computing $C_{\max}(\sigma_2, S_I(n))$ note the following facts. Since $l_{k+1} = ... = l_{2k} = l_1 + n - k$, second operation of job 1 will collide with all the second operations of the jobs $k + 1, ..., 2k$. By definition of the algorithm the machine will be idle in σ_2 in the interval $[n - k + 1, n]$ and job 1 will be scheduled after the second operation of job $2k = n - 2$, i.e., at position $n + k + 1$. The second operation of job $n - 1$ will scheduled after the second operation of job 1, because $n - k - 1 + n + k + 1 = 2n$ and $n + k + 1 + k - 1 + 1 + 1 = 2n$.
Consequently, the machine will be idle in position $n+k+2$, and the first operation of job k can be scheduled only after the second operation of job $k - 1$, i.e., at position.

$$n - k + k + k + 1 + k - 1 + 1 = n + 3k + 1 = \frac{5n - 4}{2}.$$

From this

$$C_{\max}(\sigma_2, S_I(n)) = \frac{5n - 4}{2} + n - 2 + 1 = \frac{7n - 6}{2},$$

and the statement of the lemma follows. □

Figure 2.4 displays the schedules σ_1 and σ_2 and the optimum schedule for $n = 10$, i.e., $m = 2$. (Second tasks are grayed.) The structure of this optimum solution will be generalized in the following.

Lemma 2.3.5 ([23]) *For problem series $S_I(n)$*

$$C_{\max}(\mathrm{OPT}, S_I(n)) = 2n.$$

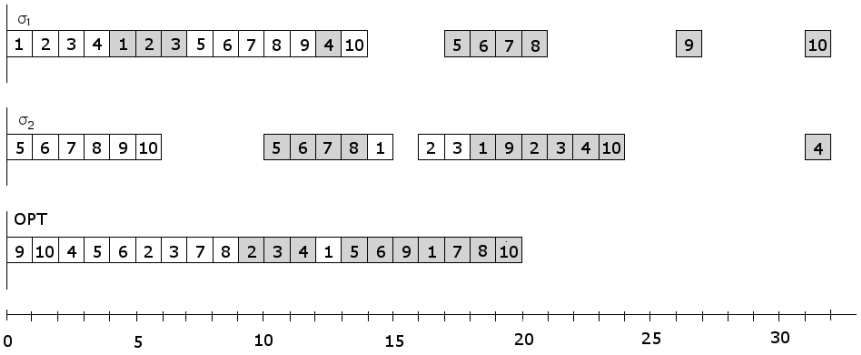

Figure 2.4: Schedules σ_1, σ_2 and the optimum schedule for $n = 10$.

We can now show the tightness of the worst-case bound.

Theorem 2.3.6 ([23]) *The worst-case performance ratio of algorithm DNF is* $\frac{7}{4}$.

Proof. Using the previous lemmas we now immediately obtain

$$\rho_{\mathrm{DNF}} \geq \lim_{n\to\infty} \frac{C_{\max}(\mathrm{DNF}, S_I(n))}{C_{\max}(\mathrm{OPT}, S_I(n))} = \lim_{n\to\infty} \frac{7n-6}{4n} = \frac{7}{4}$$

which proves the tightness of the upper bound for algorithm DNF.

2.3.4 Conclusions

Ageev and Baburin [1] gave an approximation algorithm for the unit execution time version of the coupled task scheduling problem. By deriving a new lower bound on the minimum makespan and by giving a new example we proved that their worst-case bound $\frac{7}{4}$ is tight.

2.4 A First Fit Type Algorithm for the CTP

2.4.1 Definitions and preliminaries

In this subsection we will consider again UET coupled task problems based on [18]. Let us consider an arbitrary feasible schedule σ. We will say that in σ J_i and J_j are *consecutive* if $S(b_i) < S(a_j)$, the jobs are *nested* if $S(a_i) < S(a_j) < S(b_j) < S(b_i)$, and the jobs are *interleaved* if $S(a_i) < S(a_j) < S(b_i) < S(b_j)$.

A similar definition can be done for the set of jobs: e.g. let S_1 and S_2 be sets of jobs. In schedule σ S_2 is *nested* in S_1 if $\forall J_i \in S_1$ and $\forall J_j \in S_2$, $S(a_i) < S(a_j) < S(b_j) < S(b_i)$. If a task is performed in the time-period $[i-1, i]$, then we say that the task occupies the position i.

From the literature, there are some lower bounds known to estimate the optimal schedule for the CTP.

$$\mathrm{OPT}(I_n) \geq LB1 = 2n. \tag{2.3}$$

The next one gives an improvement for those cases where $\sum l_i \geq n(n-1)$ (see [23]).

$$\mathrm{OPT}(I_n) \geq LB2 = 2n + \left\lceil \frac{1}{n}\Big(\sum_{i=1}^{n} l_i - n(n-1)\Big) \right\rceil. \tag{2.4}$$

If we apply the lower bound (2.4) for inputs with two different delays we can use the following estimation.

$$\mathrm{OPT}(I_n) \geq LB3 = n_1 + n_2 + \frac{n_1 L_1 + n_2 L_2}{n_1 + n_2} + 1, \tag{2.5}$$

where n_1 and n_2, mean the number of jobs with longer and shorter delays, respectively. At the end, if we have inputs with two different delays then the first part of the input – with n_1 jobs and L_1 delays – gives also a lower bound.

$$\mathrm{OPT}(I_n) \geq LB4 = n_1 + L_1 + 1. \tag{2.6}$$

2.4.2 Algorithm First-Fit Decreasing

We introduce the *First-Fit Decreasing* (*FFD*) approximation algorithm for the problem

$$1|\mathit{Coup\text{-}Task}, \mathit{exact}\ l_i \in \{L_1, L_2\},\ a_i = b_i = 1|C_{\max}.$$

The FFD algorithm was originally defined by D. S. Johnson [54] as a bin packing approximation algorithm. We define the appropriate version of this algorithm to our problem in consideration. Naturally, the algorithm is designed to solve problems with arbitrary number of different delays but we investigate it only for the UET problem with two distinct delays among tasks.

Algorithm First-Fit Decreasing

Step 1. Sort the jobs in non-increasing order according to the idle times. After ordering we suppose that $l_1 \geq l_2 \geq \ldots \geq l_n$. Let $i = 1$.

Step 2. Schedule job J_i from the earliest time which results in a feasible schedule.

Step 3. $i = i + 1$. If $i \leq n$ then goto Step 2. Otherwise END.

2.4.3 Theorems for scheduling jobs with equal delays

Let I_n be an instance with jobs of equal delay times, and let us denote the common delay time by L. Such instance will be called *L-uniform* instance. We will say that the subset $\{J_i, \ldots, J_m\}$ of jobs are *continuously* scheduled if $\forall j$, $i \leq j < m$, $S(a_{j+1}) = S(a_j) + 1$. It is clear that for an L-uniform instance *FFD* schedules the items in a block continuously. Furthermore, let J_k and J_{k+1} be the last item of block B, and the first item of block $B+1$, respectively. Then if $S(b_k)+1 = S(a_{k+1})$, then the two blocks are also continuously scheduled.

Let I_n be an L-uniform instance. Let $k = \lfloor n/(L+1) \rfloor$, and $n = n_c + n_r$, where $n_c = k(L+1)$, and $n_r = n - k(L+1) \leq L$. In [67] an optimal algorithm has

been defined for the problem $1|CTP, a_i = b_i = p, l_i = L|C_{\max}$. If we apply this algorithm for the case $a_i = b_i = 1$, then we get the following algorithm.

Algorithm Greedy

Step 1. Compute $k = \lfloor n/(L+1) \rfloor$.

Step 2. Form k (complete) blocks of jobs, where each block contains $L+1$ jobs.

Step 3. If $n_r > 0$, then form an incomplete block containing all the remaining jobs where the first tasks are scheduled continuously.

Step 4. Schedule the complete blocks and the incomplete blocks – if any – continuously.

This algorithm has been analyzed in [67] and it was proven to provide an optimal schedule. The value of $\mathrm{OPT}(I_n)$ was not given. Here, we give a simpler – but more formalized – proof, and we give $\mathrm{OPT}(I_n)$ explicitly.

Theorem 2.4.1 ([18]) *Algorithm Greedy generates an optimal schedule for the* $1|Coup - Task, a_i = b_i = 1, l_i = L|C_{\max}$ *problem in* $O(n)$ *time, and*

$$\mathrm{OPT}(I_n) = \begin{cases} k(L+1)+n, & \text{if } n_r = 0; \\ (k+1)(L+1)+n, & \text{otherwise.} \end{cases}$$

Proof We already know that Greedy generates an optimal schedule, so we prove only the validity of the formula.
First consider the case when all blocks are complete, i.e. $n_r = 0$. Then $\mathrm{OPT}(I_n) = 2k(L+1)$ and $n = k(L+1)$ thus the claim holds.
Now suppose that the last block is not complete. The contribution of k complete blocks to the makespan is $2(L+1)k$. The length of the last block is $L+2+(n_r-1)$ where $n_r = n - k(L+1)$. Thus we get

$$\begin{aligned} \mathrm{OPT}(I_n) &= 2(L+1)k + L + 2 + n - k(L+1) - 1 \\ &= 2(L+1)k + L + 1 - k(L+1) + n \\ &= (k+1)(L+1) + n. \end{aligned}$$

□

Since for L-uniform instances an *FFD* schedule is identical to *Greedy*, the following corollary is a consequence of Theorem 2.4.1.
Corollary For L-uniform instances

$$C_{\max}^{FFD}(I_n) = \begin{cases} k(L+1)+n, & \text{if } n_r = 0; \\ (k+1)(L+1)+n, & \text{otherwise.} \end{cases} \tag{2.7}$$

We realize that $C_{\max}^{FFD}(I_n)$ is a function of n, and L. Let us denote this function by

$$C_{\max}^{FFD}(I_n) = f(n, L). \tag{2.8}$$

Considering the formula of (2.7), we see that moving from n to $(n+1)$, $f(n, L)$ grows by 1 if the last block is not complete (for n), otherwise (if the last block is complete for n) the value of (2.7) is growing by $L+2$. In both cases, by $n \to (n+1)$, the value of (2.7) is growing by at least 1. Thus, if we increase n by $d \geq 1$, the value of (2.7) is growing by at least d. Thus we have

$$f(n-d, L) \leq f(n, L) - d \tag{2.9}$$

for any $1 \leq d \leq n$.

2.4.4 Theorems for two different delays

Hereinafter, we suppose that instances contain only jobs with two different delay times L_1 and L_2, where $L_1 > L_2$. Jobs with delay time L_1, or L_2 are called *long*, or *short*, resp. Let $I_n = (I^1, I^2)$ be the concatenation of instances I^1 and I^2, with n_1 long and n_2 short jobs ($n = n_1 + n_2$), resp. To avoid the complicated notations sometimes we write instead of I_n simply I.

If we have two different delay times, then Step 1 in algorithm *FFD* can be performed in $O(n)$ time creating two "heaps" for the long and short jobs, respectively.

In subsection 2.4.1 we defined the set of interleaved jobs. If we have jobs with two different gaps, then a (set) of short jobs can be in interleaved position either with a (set) of long jobs or with a (set) of short jobs. In these cases, we speak about *long-interleaved* or *short-interleaved* jobs, resp. If for a short job J_i, $S(a_i) > S(b_{n_1})$ $(i > n_1,)$ then we call the job *long-consecutive*.

Figure 2.5: An example for two different types of interleaved short jobs. Boxes denote the long jobs and circles correspond to short jobs.

We define the algorithm Separate, denoted by *SEP*. This algorithm applies the *FFD* rule independently for the sub-instances I^1, and I^2, and concatenates the two schedules.

Algorithm Separate

Step 1. $t = 1$. Schedule all jobs in I^1 from the position t using *FFD*.

Step 2. $t = C_{\max}^{FFD}(I^1) + 1$. Schedule all jobs in I^2 starting at position t by the algorithm *FFD*.

First, we prove a simple Claim that we use several times thereinafter.

Claim 2.4.2 ([18]) *If the FFD schedule of input I_n contains long-interleaved short jobs, and there is at least one empty position just after the second task of the last long job, then the idle time (denoted by τ) in the FFD schedule is at most L_2.*

Proof During the proof we use the following – easily provable – facts. If the conditions are true then
(a) there is at least one nested complete short block,
(b) there is no incomplete nested short block,
(c) the first long-interleaved short job starts just after the second task of the last nested short job.

Because of the conditions, there is at least one empty position just after the second task of the last long job. Let us denote the number of complete nested blocks of short jobs by $n_{2,cn}$. There are two cases.

Case A. Suppose that $n_2 \geq (L_1 - n_1 + 1) - 2(L_2 + 1)n_{2,cn}$.

In this case, the gap within the incomplete long block will be filled with nested jobs and the first tasks of interleaved short jobs. Therefore, there is no gap within the incomplete block of the long jobs. So, idle time can occur only after the incomplete block of the long jobs. By this reason, the sum of idle times in the *FFD* schedule – independently whether or not long-consecutive short jobs exist – is at most L_2.

Case B. Suppose that $n_2 < (L_1 - n_1 + 1) - 2(L_2 + 1)n_{2,cn}$.

Now, there is no long-consecutive short job. A gap remains within the incomplete block of the long jobs, and there is also an idle time after the second task of the last long job. Let us denote these gaps by τ_1, and τ_2, resp. Both gaps start up within the two tasks of the first interleaved short job (see Figure 2.6), thus $\tau = \tau_1 + \tau_2 < L_2$.

□

Figure 2.6: Schedule of short interleaved jobs if second tasks are not consecutive.

Lemma 2.4.3 ([18]) *For any input I with two different delays $C_{\max}^{SEP}(I) \geq C_{\max}^{FFD}(I)$.*

Proof For scheduling the long jobs *FFD* and *SEP* are identical, so $C_{\max}^{FFD}(I^1) = C_{\max}^{SEP}(I^1)$. Furthermore, the two algorithms remain identical if there are only complete blocks from long jobs. So we can suppose that an incomplete block from long jobs exists. Let us apply the equality (2.8) for the instances I^1 and I^2 with parameters (n_1, L_1), and (n_2, L_2) We get

$$C_{\max}^{SEP}(I) = C_{\max}^{FFD}(I^1) + C_{\max}^{FFD}(I^2) = f(n_1, L_1) + f(n_2, L_2) = t + f(n_2, L_2).$$

Denote the number of nested short jobs by $n_{2,ne} \geq 0$. Now we distinguish several cases.

Case A. There is no (long-)interleaved short job. Then *FFD* schedules $n_2 - n_{2,ne}$ jobs from time t, so

$$C_{\max}^{FFD}(I) = t + f(n_2 - n_{2,ne}, L_2) \leq t + f(n_2, L_2) - n_{2,ne} \leq C_{\max}^{SEP}(I).$$

Case B. There is at least one long-interleaved short job. Suppose the last interleaved short job ends at position $p > t$. It follows that positions $j = t, ..., p-1$ are either idles or they are occupied by the second tasks of some interleaved short jobs.

Case B.1. There is no empty position between t and $p-1$. This means that there are $n_{2,i} = p - t$ interleaved short jobs.

Let $n_{2,lc} \geq 0$ be the number of long-consecutive short jobs. Since there are also $n_{2,ne}$ nested small jobs, $n_{2,lc} = n_2 - n_{2,ne} - n_{2,i}$. The long-consecutive short jobs are scheduled by *FFD* from position p. Then

$$\begin{aligned} C_{\max}^{FFD}(I) &= p + f(n_2 - n_{2,ne} - (p-t), L_2) \leq p + f(n_2, L_2) - n_{2,ne} - (p-t) \\ &= t + f(n_2, L_2) - n_{2,ne} \leq SEP(I). \end{aligned}$$

Case B.2. There is at least one empty position between t and $p-1$. Now, we can use the Claim 2.4.2, and so, $\tau \leq L_2$.

In this case the first interleaved short job ends strictly later than $t+1$, and the incomplete long block will be filled with nested jobs and the first tasks of interleaved short jobs. By this reason, the sum of idle times in the *FFD* schedule is at most L_2.

On the other hand, in the *SEP* schedule all positions in the gap of the incomplete long block remain idle. Since in the *FFD* schedule there is at least one complete block of nested jobs, the length of the gap is at least $2(L_2 + 1) > L_2$. Thus the total length of idle intervals in the *SEP* schedule is bigger than L_2. Then clearly $C_{\max}^{FFD}(I) < C_{\max}^{SEP}(I)$. □

At this point we are able to easily determine the time complexity of algorithm *FFD* . We assume that L_1 and L_2 are fixed.

Claim 2.4.4 ([18]) *The time complexity of the FFD algorithm is* $O(n)$.

Proof The ordering in Step 1 needs only $O(n)$ time, since there are only two different types of jobs. In Step 2, *FFD* first schedules the long jobs in $O(n)$ time. Then come the short jobs. From this point we store (in a vector) that what time-slots are occupied and what time slots are free. Note, that any time slot will be checked at most once: if the time slot will be occupied by the current job, this time slot will be never checked again. Otherwise, if the time slot is free but cannot be occupied by the current (short) job, it never will be able to be occupied by some other short job.

We know from Lemma 2.4.3 that $C_{\max}^{FFD}(I) \leq C_{\max}^{SEP}(I) \leq 2n + L_1 + L_2$ holds, it means that at most so many time slots are tried (each one is tried at most once). We conclude that the complexity is $O(n)$. □

2.4.5 Lower bound for *FFD*

In this subsection, we give lower bounds for the performance ratio of *FFD*.

Theorem 2.4.5 ([18]) *Let $I(n,k)$ be an instance which contains $n = n_1+n_2 = 9k$ jobs, where $n_1 = 3k$ and $n_2 = 6k$ pieces of long and short jobs, respectively. Let us suppose that $L_1 = 12k-2$ and $L_2 = 9k-2$. Then*

$$\limsup_{k\to\infty} \frac{C_{\max}^{FFD}(I(n,k))}{\mathrm{OPT}(I(n,k))} = \frac{30}{19}.$$

Proof After having scheduled long jobs there is a gap of $9k-1$ among the two tasks of the items. Figure 2.7 shows the status when every long job of $I(n,k)$ has been scheduled.

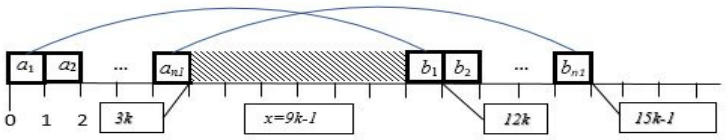

Figure 2.7: Status when all long jobs are scheduled by *FFD*.

Since a short job needs $9k$ positions, *FFD* can not schedule any short job as nested job in the gap of long jobs. So, *FFD* will start to schedule interleaved short jobs.

FFD will schedule the first item in such a way that its second task occupies the first empty position after the last task of the long jobs. It occupies the positions $6k+2$ and $15k$ positions for the first and the second tasks, respectively. (The tasks occupy the units $[6k+1, 6k+2]$ and $[15k-1, 15k]$. It is clear that the position $15k$ was free. Similarly, the position $6k+2$ is also free. So, the first interleaved short job can be scheduled in this position. Following this idea, *FFD* can schedule $(12k-1)-(6k) = 6k-1$ interleaved short jobs. The second task of the last interleaved short job occupies the position $21k-2$.

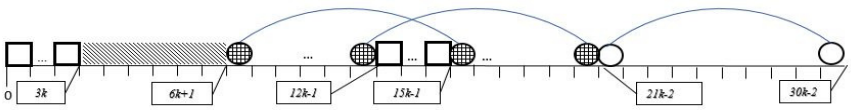

Figure 2.8: Status when all the jobs of $I(n,k)$ are scheduled by *FFD*.

The remaining short job needs L_2+2 units to be completed. Therefore, the makespan of the instance $I(n,k)$ produced by the algorithm *FFD* is $C_{\max}^{FFD}(I(n,k)) = 21k-2+L_2+2 = 30k-2$. For the instance $I(n,k)$

$$\left\lceil \frac{1}{n}\Big(\sum_{i=1}^{n} l_i - n(n-1)\Big) \right\rceil = k-1 \geq 0,$$

therefore, we can apply LB2, and so $C^*(I(n,k)) \geq 19k-1$. Therefore,

$$\limsup_{k\to\infty} \frac{C_{\max}^{FFD}(I(n,k))}{\mathrm{OPT}(I(n,k))} \leq \frac{30k-2}{19k-1} \tag{2.10}$$

To prove that the right hand side of the inequality (2.10) is tight, we consider the following feasible schedule of the instance $I(n,k)$. We schedule three times k pieces of long jobs and $2k$ pieces of short jobs at the earliest possible time. We illustrate the schedule on the Figure 2.9 for the case $k=2$.

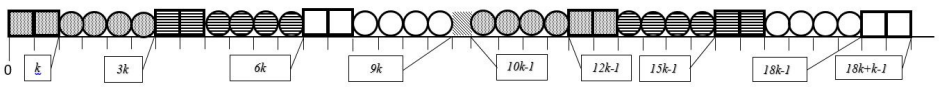

Figure 2.9: An optimal schedule of the instance $I(n,k)$ if $k=2$.

It is easy to check that all the positions are occupied except those ones which are between the first task of the last interleaved short job and the second task of the first nested short job. So the gap in this schedule is:

$$(12k-2) - [(k-1) + 2k + 3k + 3k + 2k] = k-1.$$

So,

$$C^*(I(n,k)) \leq 2n + (k-1) = 19k-1. \tag{2.11}$$

Therefore

$$\limsup_{k\to\infty} \frac{C_{\max}^{FFD}(I(n,k))}{\mathrm{OPT}(I(n,k))} \geq \frac{30k-2}{19k-1} \tag{2.12}$$

Taking the inequalities (2.10) and (2.12), we get the desired result.

□

2.4.6 Upper bound for *FFD*

In this section, we give an upper bound for the performance ratio of *FFD*. This ratio depends strongly on the structure of an instance i.e. does the *FFD* schedule contain complete blocks, are there nested short jobs, or interleaved short jobs, etc.? We will investigate *FFD* schedules with different structures. In the following, if for two instances I and I', $C_{\max}^{FFD}(I)/\mathrm{OPT}(I) \leq C_{\max}^{FFD}(I')/\mathrm{OPT}(I')$, then we say that $C_{\max}^{FFD}(I')$ *dominates* $C_{\max}^{FFD}(I)$. Our upper bound proof will be divided into three parts.

- First, we will consider instances for which $C_{\max}^{FFD}(I)/\mathrm{OPT}(I) \leq \frac{3}{2}$.
- Secondly, we investigate inputs for which there is a dominant instance with a given structure.

- Finally, we give an upper bound for the set of dominant instances.

In the sequel, we will use the following notations. Let k_b, k_s, r_b, and r_s be such integers, for which $n_1 = k_b(L_1+1)+r_b$ where $1 \leq r_b \leq L_1$ and $n_2 = k_s(L_2+1)+r_s$ where $0 \leq r_s \leq L_2$. Then

$$k_b(L_1+1) = n_1 - r_b \quad \text{and} \quad k_s(L_2+1) = n_2 - r_s. \tag{2.13}$$

Inputs with performance ratio at most $\frac{3}{2}$.

Lemma 2.4.6 ([18]) *If I is such an input for which there is at least one complete block of long jobs in the FFD schedule, then $\frac{C^{FFD}_{\max}(I)}{\mathrm{OPT}(I)} \leq 3/2$.*

Proof Since there is a complete block of long jobs, we have

$$L_1 + 1 \leq n_1. \tag{2.14}$$

Applying (2.8) for the instances I^1 and I^2 separately, and taking into account that for L-uniform instances $C^{FFD}_{\max}(I) = C^{SEP}_{\max}(I)$, we get that

$$C^{SEP}_{\max}(I) = f(n_1, L_1) + f(n_2, L_2). \tag{2.15}$$

On the other hand, applying Theorem 2.4.1 for I^1, we have

$$\mathrm{OPT}(I) \geq \mathrm{OPT}(I^1) = f(n_1, L_1).$$

Case A. If $k_s = 0$ (i.e. there is no complete block of short jobs in the SEP schedule), then we first apply the lower bound (2.6). Then using the inequality (2.14) and the lower bound (2.3), we have

$$\begin{aligned} C^{SEP}_{\max}(I) - \mathrm{OPT}(I) &\leq f(n_2, L_2) = L_2 + 1 + n_2 \\ &\leq L_1 + n_2 < n_1 + n_2 = n \leq \mathrm{OPT}(I)/2, \end{aligned}$$

thus $C^{SEP}_{\max}(I) \leq (3/2)\mathrm{OPT}(I)$ and we are done.
Case B. If $k_s > 0$ (i.e. there is at least one complete block also from the short jobs in the SEP schedule), then

$$L_2 + 1 \leq n_2. \tag{2.16}$$

Let us start with (2.15) and apply (2.13)- (2.16). Then we get

$$\begin{aligned} C^{SEP}_{\max}(I) &= f(n_1, L_1) + f(n_2, L_2) \\ &\leq (k_b+1)(L_1+1) + n_1 + (k_s+1)(L_2+1) + n_2 \\ &= n_1 - r_b + (L_1+1) + n_1 + n_2 - r_s + (L_2+1) + n_2 \\ &\leq 2n_1 + (L_1+1) + 2n_2 + (L_2+1) < 3n_1 + 3n_2 = 3n \\ &\leq (3/2)\mathrm{OPT}(I) \end{aligned}$$

where the last inequality follows from the lower bound (2.3).

□

Lemma 2.4.7 ([18]) *Let us suppose that $k_b = 0$, i.e. there is no complete block from the big jobs. If there is at least one complete block of short jobs, then $\frac{C_{\max}^{FFD}(I)}{\mathrm{OPT}(I)} \leq 3/2$.*

Proof Let us denote the number of short jobs in complete blocks by $n_{2,c}$. Then

$$n_2 \geq n_{2,c} \geq L_2 + 1. \tag{2.17}$$

If all of the short jobs are nested jobs then the *FFD* schedule is optimal. So, we can suppose that there is at least one short job i with $S(b_i) \geq S(b_{n_1}) + 1$. Let us denote the sum of idle times in the *FFD* schedule by τ. If $\tau \leq n$ then $C_{\max}^{FFD}(I) \leq 3n$. Thus it is enough to show that $\tau \leq n$.
Case A. There is at least one long-interleaved short job, and these jobs are scheduled so that the second tasks are *not* consecutive to the last task of the long jobs (see Figure 2.6). Now, we apply Lemma 2.4.2, which results that $\tau \leq L_2$. From (2.17) we get $L_2 < n_2 < n$. Therefore $\tau < n$, and we are done.
Case B. There is at least one long-interleaved short job, and these jobs are scheduled so that the second tasks are consecutive to the last task of the long jobs. This schedule of the interleaved jobs does not result in gap just after the second task of the last long job (see Figure 2.10).

Let us consider the tasks which are scheduled in the gap of the last long-interleaved short job. These tasks are the second tasks of the last incomplete block of nested jobs (if any), the second tasks of the long jobs and second tasks of the long-interleaved jobs except the last one. Since there is no gap within this time interval, here there are L_2 tasks. Note that we also have at least one complete block of $L_2 + 1$ short jobs (which are nested jobs or long-consecutive jobs), and these jobs belong to another subset of I^2. This means, that the number of jobs is at least $n \geq 2L_2 + 1$.

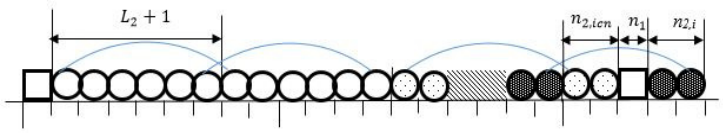

Figure 2.10: Schedule of short interleaved jobs if first tasks are consecutive.

On the other hand, gap (idle time interval) can happen only in the incomplete block of long-consecutive jobs and within the first tasks and second tasks of the long jobs. The size of both gaps is at most L_2. Thus, the total gap is at most $\tau \leq 2L_2 \leq n$ and we are done.
Case C. There is no interleaved short job in the FFD schedule at all. Similar to the Case B, we get that the total gap is at most $\tau \leq 2L_2$.

Now, we can suppose that the number of long-consecutive jobs is $n_{2,lc} > 0$, otherwise the schedule is optimal. Any long-consecutive job could not be scheduled as long-interleaved job, thus $n_{2,icn} + n_1 > L_2$, where $n_{2,icn}$ is the number of jobs in the incomplete block of the nested short jobs (see again Figure 2.10). Since in

the schedule there are further short jobs which form at least one complete block, $n \geq n_{2,icn} + n_1 + (L_2 + 1) > 2L_2 + 1 > \tau$.

□

Lemma 2.4.8 ([18]) *If there are neither nested nor interleaved short jobs in the input I, then $C_{\max}^{FFD}(I)/\mathrm{OPT}(I) \leq 3/2$.*

Proof If the conditions are true then all short jobs are long-consecutive and $C_{\max}^{FFD}(I) = C_{\max}^{SEP}(I)$. Since there is no interleaved job, therefore $L_2 \leq n_1 - 1$. So, we get

$$C_{\max}^{FFD}(I) = (n_1 + L_1 + 1) + (n_2 + L_2 + 1) = n + L_1 + L_2 + 2$$
$$\leq n + (L_1 + n_1 + 1) = \frac{1}{2}LB1 + LB4 \leq (3/2)\mathrm{OPT}(L).$$

□

In the next lemma, we exclude the case when there is no long-consecutive short job at all.

Lemma 2.4.9 ([18]) *If there is no long-consecutive short job in the input I, then $C_{\max}^{FFD}(I)/\mathrm{OPT}(I) \leq 3/2$.*

Proof We claim that $C_{\max}^{FFD}(I) \leq L_1 + n + 2$. Let us consider the status when all the long jobs are just scheduled. Now, the first n_1 time slots are occupied, and the time slots are busy from $L_1 + 1$ to $L_1 + 2 + (n_1 - 1) = L_1 + n_1 + 1$.

Now let us see how the short jobs are scheduled.

First, suppose that there is no nested short job. Then all short jobs are interleaved jobs and since $L_2 < L_1$, the second tasks of the short jobs can be assigned continuously from the time $L_1 + n_1 + 1$, and so we have $C_{\max}^{FFD}(I) = (L_1 + n_1 + 1) + n_2$.

Otherwise, there are several nested jobs. Let us denote by $n_{2,ne}$ and $n_{2,i}$ the number of nested short jobs and the number of interleaved jobs, resp. Then $n_2 = n_{2,ne} + n_{2,i}$ and so $n_{2,i} < n_2$. Therefore,

$$C_{\max}^{FFD}(I) = (L_1 + n_1 + 1) + n_{2,i} < (L_1 + n_1 + 1) + n_2 = L_1 + n + 1,$$

and so

$$\frac{3}{2}\mathrm{OPT}(I) \geq LB4 + LB1/2 = (L_1 + n_1 + 1) + n \geq C_{\max}^{FFD}(I).$$

□

Inputs with dominant instances

From Lemma 2.4.8 we know that if there are neither nested nor interleaved short jobs then $C_{\max}^{FFD}(I)/\mathrm{OPT}(I) \leq \frac{3}{2}$. In this part we will investigate those cases when one of them occurs. Let

$$LB = \max\{LB1, LB3, LB4\}.$$

Lemma 2.4.10 ([18]) *Suppose that I is an input that contains nested short jobs in its FFD schedule. Then there exists a dominant instance I' for which the FFD schedule does not contain nested jobs.*

Proof Let $z > 0$, $n_{2,i}$ denote the number of nested and the number of interleaved short jobs, respectively. By Lemma 2.4.6, we can suppose that the *FFD* schedule of I does not contain complete blocks of long jobs, and by Lemma 2.4.7, there is no complete block of nested short jobs. So, the nested short jobs form an incomplete block, and their second tasks occupy z positions just before the second tasks of the long jobs (see Figure 2.11).

Now, in I' let $L'_1 = L_1 - z$, and we consider the changes in the *FFD* schedule of our modified input. We remark that $L_2 = L'_1 - n_1$.

Claim 2.4.11 $LB(I') \leq LB(I)$

Proof Since $LB1$ depends on only the number of jobs therefore $LB1(I) = LB1(I')$. In I' the sum of the gaps are smaller than in I. Therefore, $\text{LB3}(I') \leq \text{LB3}(I)$, and $\text{LB4}(I') \leq \text{LB4}(I)$.

□

Case A. First, we suppose that there is at least one interleaved short job in I. Then

$$L_2 = n_1 + z + (n_{2,i} - 1). \tag{2.18}$$

If we schedule the items of I', then each earlier nested short job will be interleaved and the interleaved short jobs remain interleaved, i.e. $z' = 0$, and $n'_{2,i} = z + n_{2,i}$. It is easy to check that after having scheduled all long jobs and all interleaved jobs, the makespan is the same and the number of already scheduled short jobs is also the same, i.e. $n_1 + z + n_{2,i} = n_1 + z' + n'_{2,i}$. As a consequence of (2.18), there is no complete block in the modified schedule.

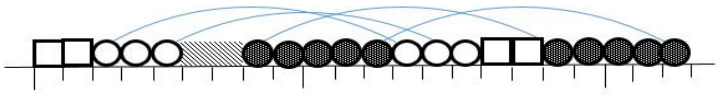

Figure 2.11: An instance with jobs, $L_1 = 14$, $n_1 = 2$, $L_2 = 9$, $n_{2,i} = 5$, and $z = 3$.

Figure 2.12: The modified instance with $L'_1 = 11$, $n_1 = 2$, $L_2 = 9$, $n'_{2,i} = 8$.

By finishing the schedule with the remained short jobs (that all will be long-consecutive jobs), the makespan of the *FFD* schedule does not change. So, applying the Claim 2.4.11 we get $C_{\max}^{FFD}(I')/LB(I') > C_{\max}^{FFD}(I)/LB(I)$.
Case B. Now assume that there is no interleaved short job while *FFD* schedules items of I. Let us denote the number of long-consecutive jobs by n_{lc}. It is clear that the *FFD* schedule of I' does not contain complete block from the long-consecutive short jobs. Now, after the scheduling of the long jobs, the last occupied time slot in I' will be z unit earlier, and the number of nested jobs is $z' = 0$. The schedule will contain $n'_{2,i} = z$ new interleaved short jobs, therefore – before the *FFD* schedules the long-consecutive short jobs – the makespan of the two schedules is equal. Since the number of long-consecutive short jobs have not changed, $C_{\max}^{FFD}(I') = C_{\max}^{FFD}(I)$, and – using again Claim 2.4.11 we get the desired inequality.

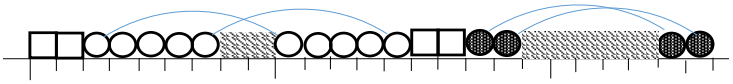

Figure 2.13: An instance without interleaved jobs $L_1 = 13$, $n_1 = 2$, $L_2 = 6$, $n_2 = 7$, $z = 5$.

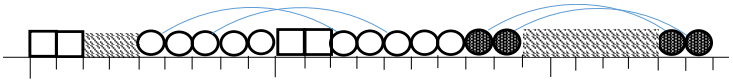

Figure 2.14: The modified instance with $L'_1 = 8$, $n_1 = 2$, $n_{2,i} = 5$, $L_2 = 6$, $n_2 = 7$.

□

Lemma 2.4.12 ([18]) *Let I be an instance for which the FFD schedule contains interleaved short jobs. Then there exists a dominant instance I' for which in the FFD schedule the first short job is an interleaved short job, and $L_2 = L_1 - n_1$.*

Proof There are no nested jobs, therefore $L_2 \geq L_1 - n_1$. Then I' must contain at least one interleaved job. If $L_2 = L_1 - n_1$ then we are ready, $I' = I$. Let $L_2 = L_1 - n_1 + z$, where $z > 0$, integer.

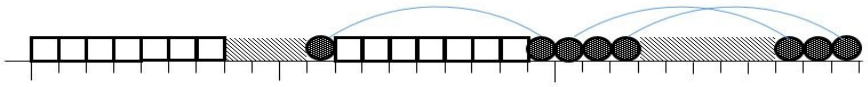

Figure 2.15: An instance with $L_1 = 10$, $n_1 = 7$, $n_2 = 4$, $L_2 = 7$, and $z = 4$

Figure 2.16: The modified instance with $L_1 = 10$, $n'_1 = 3$, $n'_2 = 8$, and $L_2 = 7$

Let us make a modified input I' where $n'_1 = n_1 - z$ and $n'_2 = n_2 + z$. Note that the number of jobs does not change. Now, having scheduled the long jobs and the interleaved short jobs by *FFD* the makespan is the same as earlier, and the number of already scheduled short jobs is increasing by z. So, finishing the schedule with the remained short jobs (that all will be long-consecutive jobs), the value of the *FFD* schedule remains the same.

Let us see how the lower bound changes. $LB1$ does not change, $LB3$ will be smaller, and $LB4$ decreases by z. Therefore, LB will not increase, thus for I' $C^{FFD}_{\max}(I')/LB(I') > C^{FFD}_{\max}(I)/LB(I)$ still holds.

□

Now, we will investigate the structure of the long-consecutive short jobs.

Lemma 2.4.13 ([18]) *Let I be an instance that its FFD schedule does not contain nested short jobs, contains interleaved and long-consecutive short jobs. Then there exists such a dominant instance I' which contains exactly one long-consecutive short job.*

Proof Suppose there are at least $z+1$ long-consecutive short jobs, where $z \leq L_2$. We derive the following instance: let $L'_1 = L_1 + z$ and $L'_2 = L_2 + z$. Note that the number of jobs does not change.

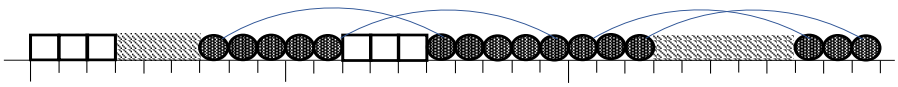

Figure 2.17: Instance with $z = 2$ short jobs in long-consecutive block.

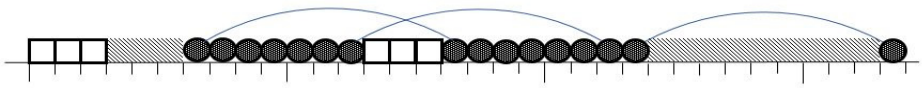

Figure 2.18: Modified input with $z = 0$ short jobs in long-consecutive block.

Now after scheduling the long jobs, the last occupied time slot will be z units later. Since $L'_2 > L_2$, the number of interleaved jobs increases by z, and just after scheduling the interleaved jobs, the corresponding makespan increased by $2z$. So, if it was previously t, now it is $t + 2z$. The number of the remained short jobs is decreased by z, but since L_2 is also increased by z, the increment comparing to

$t + 2z$ is the same, as the increment was comparing to t in the previous input. It means that the makespan of the *FFD* schedule increases by $2z$.

Let us see how the lower bound changes. LB_1 does not change, LB_3 will be increased by z, and LB_4 also grows by z. This means that LB increases by at most z. Therefore, for I' it still holds that $C_{\max}^{FFD}(I')/LB(I') > C_{\max}^{FFD}(I)/LB(I)$, as the numerator increased by $2z$, while the denominator increased by at most z. □

The upper bound

In the previous subsections we have found some instances with performance ratio $\leq 3/2$, and also some instances have been analyzed for which the *FFD* schedules have dominant examples with given structures. Therefore the following proposition is true.

Proposition 2.4.14 *If we want to find an instance I with $C_{\max}^{FFD}(I)/LB(I) > 3/2$ then we have to analyze those cases for which the FFD schedule*
(a) does not contain complete blocks (Lemma 2.4.6, Lemma 2.4.7),
(b) does not contain nested short jobs (Lemma 2.4.8, Lemma 2.4.10),
(c) contains interleaved short jobs (Lemma 2.4.8, Lemma 2.4.12),
(d) the idle time of the short jobs is $L_2 = L_1 - n_1$ (Lemma 2.4.12),
(e) contains exactly one long-consecutive short job (Lemma 2.4.9, 2.4.13).

Let I be an example, which satisfies the conditions (a)-(e). From Proposition 2.4.14 the following simple assumptions follow.

- $n_1 < L_1 + 1$, results in a schedule with one – incomplete – block of long jobs, and after having scheduled the long jobs an idle time starts up with size $L_1 - n_1 + 1 > 0$.

- $L_2 + 2 > L_1 - n_1 + 1$, to avoid nested short jobs. We will investigate those special inputs where $L_2 = L_1 - n_1$, which results in the longest short jobs in the instance.

- $L_2 \geq n_1$, results in several interleaved jobs.

- $n_2 = L_2 - n_1 + 2$. Using this condition, *FFD* schedules as much interleaved jobs as possible and just one short job remains. This lonely short job will be a long-consecutive job.

As consequence of the assumptions above, if an instance I disposes of the characteristics above, the following facts are true.

Fact 2.4.15 *From the conditions it follows that $L_2 = L_1 - n_1 \geq n_1$, and so*

$$\frac{L_1}{n_1} \geq 2. \tag{2.19}$$

Fact 2.4.16 *After scheduling the long jobs we have*

$$C_{\max}^{FFD}(I^1) = L_1 + 2 + n_1 - 1 = L_1 + n_1 + 1. \tag{2.20}$$

Fact 2.4.17 *After scheduling the interleaved jobs we have*

$$C_{\max}^{FFD}(I) = (L_1 + n_1 + 1) + (L_2 - n_1 + 1) = L_1 + L_2 + 2. \tag{2.21}$$

Fact 2.4.18 *After scheduling the last short job we have*

$$C_{\max}^{FFD}(I) = (L_1 + L_2 + 2) + L_2 + 2 = L_1 + 2L_2 + 4. \tag{2.22}$$

We compare $C_{\max}^{FFD}(I)$ and $\mathrm{OPT}(I)$. Estimating the optimal solution, we use the lower bound $LB3$. To do that, let us express L_2 and n_2 as the variables of L_1 and n_1. Applying $L_2 = L_1 - n_1$ and $n_2 = L_2 - n_1 + 2 = L_1 - 2n_1 + 2$, we get

$$C_{\max}^{FFD}(I) = L_1 + 2L_2 + 4 = 3L_1 - 2n_1 + 4.$$

and

$$\begin{aligned}\mathrm{OPT}(I) &\geq n_1 + (L_1 - 2n_1 + 2) + \frac{n_1 L_1 + (L_1 - 2n_1 + 2)(L_1 - n_1)}{n_1 + L_1 - 2n_1 + 2} + 1 \\ &= L_1 - n_1 + 3 + \frac{n_1 L_1 + (L_1 - 2n_1 + 2)(L_1 - n_1)}{L_1 - n_1 + 2}\end{aligned}$$

Let us introduce the new variable $x = L_1/n_1$. Then we can express the previous values as $C_{\max}^{FFD}(I)/n_1 = 3x - 2 + 4/n_1$ and

$$\frac{\mathrm{OPT}(I)}{n_1} \geq x - 1 + \frac{3}{n_1} + \frac{x + (x - 2 + \frac{2}{n_1})(x - 1)}{x - 1 + \frac{2}{n_1}}.$$

For the sake of simpler notation we introduce the substitution $1/n_1 = a$. Then we get

$$\begin{aligned}\frac{C_{\max}^{FFD}(I)}{\mathrm{OPT}(I)} &\leq \frac{3x - 2 + 4a}{x - 1 + 3a + \frac{x + (x-2+2a)(x-1)}{x-1+2a}} \\ &= \frac{(3x - 2 + 4a)(x - 1 + 2a)}{(x - 1 + 3a)(x - 1 + 2a) + x + (x - 2 + 2a)(x - 1)} \\ &= \frac{(3x - 2 + 4a)(x - 1 + 2a)}{6a^2 - 7a + 7ax + 2x^2 - 4x + 3}\end{aligned} \tag{2.23}$$

and we are interested in the maximum of the right hand side. Because of the inequality (2.19), we can suppose that $2 \leq x$.

Lemma 2.4.19 ([18]) *If for an instance I, $2 \leq x \leq 4$ then*

$$\frac{C_{\max}^{FFD}(I)}{\mathrm{OPT}(I)} \leq \frac{30}{19}$$

Proof Applying the equality (2.23), our claim is equivalent with

$$3x^2 + (20a - 25)x + (28a^2 - 58a + 52) \geq 0 \tag{2.24}$$

Here, the discriminant is

$$D(a) = (20a - 25)^2 - 4 \cdot 3 \cdot (28a^2 - 58a + 52) = 64a^2 - 304a + 1.$$

Recall that $a = 1/n_1$ where n_1 is integer, thus $0 < a \leq 1$. It is easy to see that $D'(a) = 128a - 304 < 0$ in the whole $(0; 1]$ interval. Thus, $D(a)$ is decreasing. The unique solution of $D(a) = 0$ in the considered interval is $\frac{19}{8} - \frac{3}{4}\sqrt{10} \approx 0.0032918$ which means that $D(a) < 0$ for $\frac{19}{8} - \frac{3}{4}\sqrt{10} < a \leq 1$, or in equivalent form, $n_1 < 1/(\frac{19}{8} - \frac{3}{4}\sqrt{10}) \approx 303.79$.

Now let us suppose that $0 < a \leq \frac{19}{8} - \frac{3}{4}\sqrt{10}$ which means that $n_1 \geq 304$. Then, the equation (2.24) has two solutions, which are $x_{1,2} = \frac{25}{6} - \frac{10}{3}a \pm \frac{1}{6}\sqrt{64a^2 - 304a + 1}$. We state that both solutions are strictly bigger than 4. It suffices to see that the smaller root is bigger than 4, i.e.

$$\frac{1}{6} - \frac{10}{3}a - \frac{1}{6}\sqrt{64a^2 - 304a + 1} > 0,$$

By simple calculation, we get $24a\,(14a + 11) > 0$, which holds. This means that the function in the left hand side in (2.24) is positive, if $x \leq 4$.

□

Lemma 2.4.20 ([18]) *If for an instance I, $4 < x$, then*

$$\frac{C_{\max}^{FFD}(I)}{\mathrm{OPT}(I)} \leq \frac{(3x - 2)(x - 1)}{2x^2 - 4x + 3} \leq \frac{\sqrt{11} + 3}{4} \approx 1.579156.$$

Proof Applying (2.23), we have to prove the following inequality.

$$\frac{(3x - 2 + 4a)(x - 1 + 2a)}{6a^2 - 7a + 7ax + 2x^2 - 4x + 3} \leq \frac{(3x - 2)(x - 1)}{2x^2 - 4x + 3}, \tag{2.25}$$

which is equivalent to the following inequality.

$$a(2x^2 + 2x - 12) + x^3 - 13x + 10 \geq 0.$$

In the left hand side $2x^2 + 2x - 12 = 2\,(x + 3)\,(x - 2) \geq 0$ because $x \geq 2$. Moreover, it is easy to see that for $x \geq 4$

$$x^3 - 13x + 10 > x^3 - 13x - 12 = (x + 3)\,(x - 4)\,(x + 1) \geq 0.$$

Now we are interested in the biggest possible value of $\frac{(3x-2)(x-1)}{2x^2-4x+3}$, considering that $x \geq 4$. To prove this, it is enough to see that

$$\frac{(3x - 2)\,(x - 1)}{2x^2 - 4x + 3} - \frac{\sqrt{11} + 3}{4} \leq 0. \tag{2.26}$$

The left hand side can be transformed as follows.

$$\frac{(3x-2)(x-1)}{2x^2-4x+3} - \frac{\sqrt{11}+3}{4} = \frac{3-\sqrt{11}}{4}\frac{\left(x-\frac{1}{2}\sqrt{11}-\frac{5}{2}\right)^2}{x^2-2x+\frac{3}{2}}.$$

Since $3-\sqrt{11}<0$ and $x^2-2x+\frac{3}{2}>0$ if $x>4$, (2.26) is true.

□

Combining the results in the Lemma 2.4.19, Lemma 2.4.20, and Theorem 2.4.5 we get the following theorem.

Theorem 2.4.21 ([18]) *For those problems where we only have jobs with two different idle times, the worst-case ratio of the FFD algorithm is*

$$1.57894\ldots = \frac{30}{19} \leq \rho_{FFD} \leq \frac{\sqrt{11}+3}{4} = 1.579156\ldots,$$

and the lower bound is tight if $L_1/n_1 \leq 4$.

2.4.7 Conclusions

In this section, we investigated a special case of the CTP where the tasks have unit length and there are only two different gaps. We considered the First Fit Decreasing (*FFD*) algorithm where the jobs are scheduled in greedy way according to their delay time: the larger the delay time, the sooner the schedule. We proved that the worst-case ratio of *FFD* is between $\frac{30}{19}$ and $\frac{\sqrt{11}+3}{4}$.

Chapter 3

Matrix Transpose Problem

3.1 Introduction

The rapidly increasing computational demands of the applied sciences pushed the computer systems progressively towards the higher computational capacity. In the same time they required more effective algorithms. Therefore, recently high performance computing is in the focus of computer science. To make computers more efficient, developments were needed both on the fields of hardware and software. Hardware developments resulted in multicore processors and connected computers with different architectures, while the algorithms became more sophisticated step by step and they have been analysed deeper than ever before. A good architecture or a more efficient algorithm may decrease the processing time strongly in a parallel computational environment.

On the hardware side the effectiveness of any parallel computation effort vigorously depends on how fast we can send data from a source processor to a destination one. Therefore different architectures were developed in the last two decades. Hypercubes, tori and meshes are the architectures that have been intensively studied. See e.g. [51], [66], [71], and [81].

Routing, sorting, merging, and matrix transpose are the problems that were investigated already in the early ages of parallel computation (see e.g. [25], [58], [60], [75]). These problems are among the basic ones, that often appear in numerical computations. For example matrix transpose is one of the basic operations in linear algebra. The speed of such computations can be critical in some real time practical applications, like digital signal processing, image processing, radar systems, etc. (see [28]). To exploit the increased computational capacity on the software side parallel algorithms have been developed, so in the last decade parallel processing has been further improved by leaps and bounds. All of the investigated algorithms were accommodated to a given architecture. The effectiveness of certain algorithms were investigated extensively, see e.g. [29], [40], [52], [62], [72] and [78].

The effectiveness of a parallel computation effort strongly depends on how fast we can send data from a source processor to a destination one. Meshes are the architectures that are flexible, the processors can be connected in different

ways, and they are suitable to implement different algorithms in an efficient way. Therefore among the different architectures the most extensively studied ones are the mesh architectures. In the simplest, one dimensional (1D) case a mesh is a linear array where the elements are the processors and each processor is connected by a full duplex line with its neighbours. In higher dimensions (2D, 3D) processors form an array, and they are connected by communication links. Figure 3.1 shows some mesh architectures.

Execution of any algorithm is performed in *steps*. In one step two connected processors can change data. Normally, only the connected processors can communicate with each other. There are different communication modes which influence the speed of data transfer. MIMD, SPMD, SIMD and the Weak SIMD are some examples for communication. (More details see in [45] and [68]).

Here we suppose MIMD communication among the processors, i.e. processors choose their communication directions independently, and they can communicate with all their neighbours in one step.

The efficiency of an algorithm is measured by the number of steps needed to fulfill the given task. While routing from a source processor to a destination one, data may pile up at a processor. This may cause a bottleneck effect if we do not have enough memory for storing these data. We assume that all processors have sufficiently large memory to store the waiting data – sometimes called as *messages* or *items* – in separate queues.

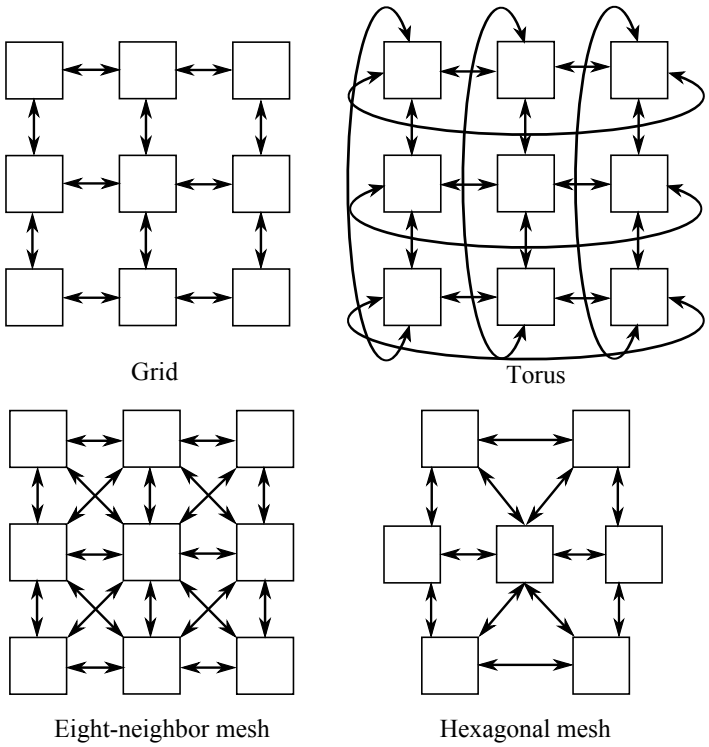

Figure 3.1: Some mesh architectures.

If the processors can communicate with only their neighbours, then sending data from a processor to a far one may take many steps. There are different

ways to avoid this situation. In [83] the so-called *wormhole switched meshes* are considered. In case of wormhole routing the data transfer has two steps. In the first one a circuit is established between the source and destination processors facilitating a quick data transfer between the nodes, and in the second step packets are sent over different paths independently from each other. The advantage of this communication lies in the first step: although it takes more time than the second one, it builds up a direct connection between the processors, and the second step allows to send packages between the processors saving much more time.

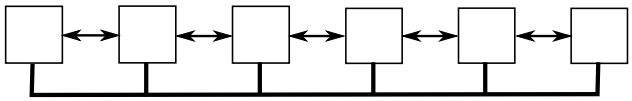

Figure 3.2: 1D mesh with bus.

To speed up the communication between two far processors, it is possible to use buses. Buses can be used in 1D meshes (see Figure 3.2.) and for 2D meshes as well. We show different bus-configurations in Figure 3.3. Row and column buses were used in [21]. If we use a bus then the processors connected to the bus can communicate not only with their neighbours but also with the ones that are connected to the same bus. In one step only one processor can send data to a bus and one of the others can accept it in the same step. In case of 2D meshes we can use row and column buses, and all processors in the same row or column are connected to one bus. To a 2D mesh which has both, row and column buses we will refer as *2RCB-mesh*.

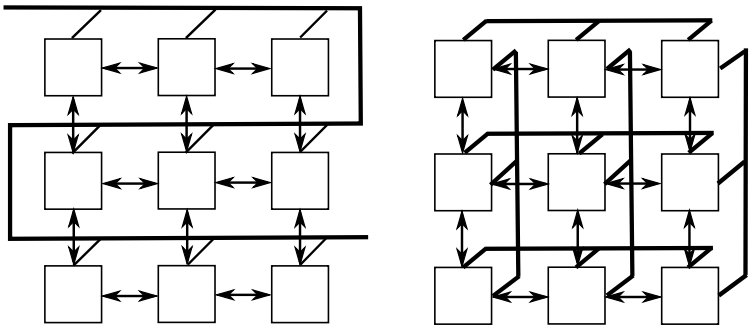

Figure 3.3: Single (snake-like) bus, row and column buses.

3.2 Definitions and preliminaries

A special permutation routing problem is the *matrix transpose problem* (*MTP*) on a 2D mesh [19]. In this case a message originally contained by the processor (i, j) should be routed to the processor (j, i) for all i, j where $1 \leq i, j \leq n$. We will call those two processors *pairs*. For 2D meshes with MIMD processors and without buses Ding, Ho and Tsai [35] analyzed the *MTP*. They denoted by $T_A(k, n)$ the

number of steps needed to transpose k pieces of $n \times n$ matrices by the algorithm A. For this $k - k$ version their main result is the following lower bound. For any *MTP* algorithm A,

$$T_A(k, n) \geq (1 - 1/\sqrt{2})kn \approx 0.293kn.$$

Later Kaufmann, Meyer and Sibeyn [57] gave an algorithm which requires $0.301kn + O(n/k)$ steps. So, for any constant k the additive term is proportional to n, therefore the gap is large between the upper and the lower bounds.

A natural generalization of the (2D) matrix transpose problem to d-dimensions ($d \geq 2$) is to consider the permutations

$$(a_1, \ldots, a_d) \to (a_i, a_{i+1}, \ldots, a_d, a_1, a_2, \ldots, a_{i-1})$$

for some $i, 1 \leq i \leq d$. These permutations are also called *transposes* [57]. There are d transposes, one of them is the identity permutation. Especially in 3D the two non–trivial transposes are $(i, j, k) \to (j, k, i)$ and $(i, j, k) \to (k, i, j)$. An architecture of a 3D mesh with buses in each direction will be denoted by *3RCB-mesh*.

First we will show that in case of a *2RCB-mesh*, we can improve the efficiency of the matrix transpose algorithms. More precisely, if we denote by $T_A^B(1, n)$ the number of steps needed to transpose a matrix by the algorithm A on a *2RCB-mesh*, then in Section 3.3 we prove that $\lim_{n\to\infty}(T_A^B(1, n)/n) > 0.4508\ldots$, and we will define an algorithm – denoted by *MTB* – for which $T_{MTB}^B(1, n) \leq \frac{n}{2} + 9$. We also investigate the 3D case, and we prove that any solution of a matrix transpose problem with a *3RCB-mesh* architecture needs at least $0.45n$ steps. In our analysis we consider only the $1 - 1$ version of the problem.

3.3 Lower Bound in 2D

Let us consider the *MTP* on a *2RCB-mesh* architecture, and let n be the number of processors both in the rows and the columns. In this case the following theorem is true.

Theorem 3.3.1 ([19]) *Let A be an arbitrary algorithm and let $T_A^B(1, n)$ be the number of steps needed to solve the* MTP *on a* 2RCB-mesh *with $n \times n$ processors. Then*

$$\lim_{n\to\infty} \frac{T_A^B(1, n)}{n} \geq 2 - \frac{2}{5}\sqrt{15} \approx 0.450806\ldots \tag{3.1}$$

Proof The idea of the proof is that we compare the walking distances to the total number of bus operations required to send data to far processors and calculate the optimal number of steps. Let us divide the $n \times n$ processors into 5 diagonal regions as shown on Figure 3.4. The regions denoted by the same letters ($\mathcal{A}$ and $\mathcal{B}$) contain equal number of processors in the corresponding rows and in the columns as well. Furthermore, let us choose x so that $\frac{n}{2} < x < n$, and let $y = 2x - n + 1$. So we get that $0 < y \leq x$. Let $P(\mathcal{A}_i)$ and $P(\mathcal{B}_i)$ be the number of processors in

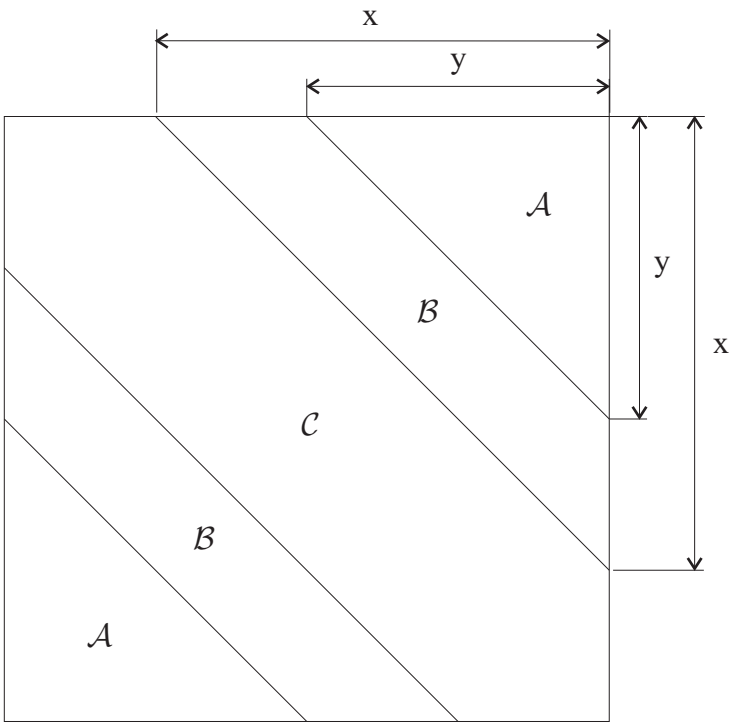

Figure 3.4: Division of the $n \times n$ mesh.

the i-th row in a region $\mathcal{A}$ and $\mathcal{B}$, respectively ($1 \leq i \leq n$). The regions are chosen so that $x = P(\mathcal{B}_1) + P(\mathcal{A}_1)$, and $y = P(\mathcal{A}_1)$.
Let us denote the distance of a pair by $|p_{i,j}|$, which means the minimum number of steps while a message moves to a destination processor through the connection lines. We will call a route that uses only communication lines to reach the destination processor as *walk.* Then $|p_{i,j}| = 2|i - j|$. We will investigate the regions $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ separately.

In the region $\mathcal{C}$ processors are close to each other, and

$$\max_{p_{i,j} \in C} |p_{i,j}| = |p_{(n-x),1}| = 2(n - x - 1) \tag{3.2}$$

The distance between any pair in the regions $\mathcal{A}$ and $\mathcal{B}$ is always longer than $2(n-x-1)$, which means that if we want to get a better result, then each message must use a bus at least once in these regions. For the pairs in regions $\mathcal{A}$ we get that

$$\min_{p_{i,j} \in A} |p_{i,j}| = |p_{n,y}| = 2(n - y),$$

so we get that

$$2(n - y) = 2(2n - 2x - 1) > 4(n - x - 1). \tag{3.3}$$

From (3.3) it follows that the messages in the region $\mathcal{A}$ must use bus twice to reach their destinations in at most $2(n - x - 1)$ steps. This means that the total number of steps that use bus operations while routing all the messages is at least

$$4P(\mathcal{A}) + 2P(\mathcal{B}).$$

We get that

$$P(\mathcal{A}) = \sum_{i=1}^{y} P(\mathcal{A}_i) = 1 + 2 + ... + y = \frac{(2x - n + 1)(2x - n + 2)}{2}.$$

Similarly,

$$\begin{aligned} P(\mathcal{B}) &= \sum_{i=y+1}^{x} P(\mathcal{B}_i) = 1 + 2 + ... + x - P(\mathcal{A}) = \\ &= \frac{x(x+1) - (2x - n + 1)(2x - n + 2)}{2}. \end{aligned}$$

Since the total number of buses is $2n$, routing of all the $2(P(\mathcal{A}) + P(\mathcal{B}))$ messages requires at least

$$T_A^B(1, n, x) = \frac{4P(\mathcal{A}) + 2P(\mathcal{B})}{2n} = \frac{x(x+1) + (2x - n + 1)(2x - n + 2)}{2n}.$$

steps. Since $T_A^B(1, n, x)$ is an increasing function, while $2(n-x-1)$ is a decreasing function of x, we get the best possible choice for x by solving the equation

$$T_A^B(1, n, x) = 2(n - x - 1). \tag{3.4}$$

The solution is

$$x = -\frac{7}{10} + \frac{1}{10}\sqrt{60n^2 - 20n + 9}.$$

Substituting this into (3.2) and using equation (3.4) we get the desired result.

□

3.4 Upper Bound in 2D

Now, we define an algorithm for solving the matrix transpose problem in 2–dimensions. During the construction we will follow the ideas used in the proof of the lower bound: those messages which are close to their destinations will walk. We call these messages *W-messages.* For a longer distance a message will be routed using one bus transfer and some walk. These are the *BW-messages.* Those messages which are very far from their destinations will be scheduled to use buses twice. We call them *BB-messages.* There are two basic problems:

- How can we determine the regions which define the message-type during the execution?
- Those pairs which are very far from each other are placed in the lower left and upper right corners. This induce that they must be defined as BB-messages. But, in this case message transfers for these pairs require a heavy usage of the outer buses, while the center buses remain idle. So, to make our algorithm more efficient we change the schedule of some parts of these

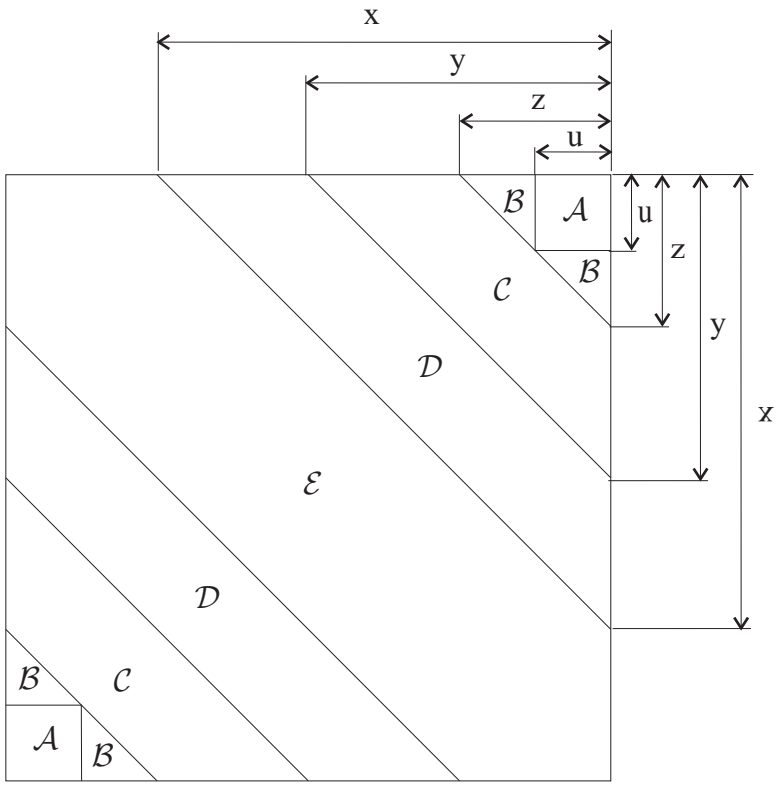

Figure 3.5: Division of the $n \times n$ mesh by algorithm LBA.

messages: first they will walk to reach one of the buses which are closer to the center, and some steps later they will "catch" one of the center buses. So, they become BW-messages. This modification results in load-balancing among the buses.

We will call our algorithm the Load-Balancing Algorithm (*LBA*).

3.5 Construction of Load Balancing Algorithm

Preparation step: Let

$$x = n - \left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor - 1, \quad y = \left\lceil \frac{n}{2} \right\rceil,$$
$$z = \left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor + 1, \quad u = \left\lceil \frac{\left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor + 1}{2} \right\rceil$$

and divide the processors into five diagonally symmetric disjoint sets $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}$ as seen on Figure 3.5.

Step 1: Label the processors in the regions $\mathcal{C}$ and $\mathcal{D}$ by R and C as seen on Figure 3.6. During the execution, processors labelled by R and C will use a row bus or a column bus first, respectively. A message originated from a *C-labeled* or

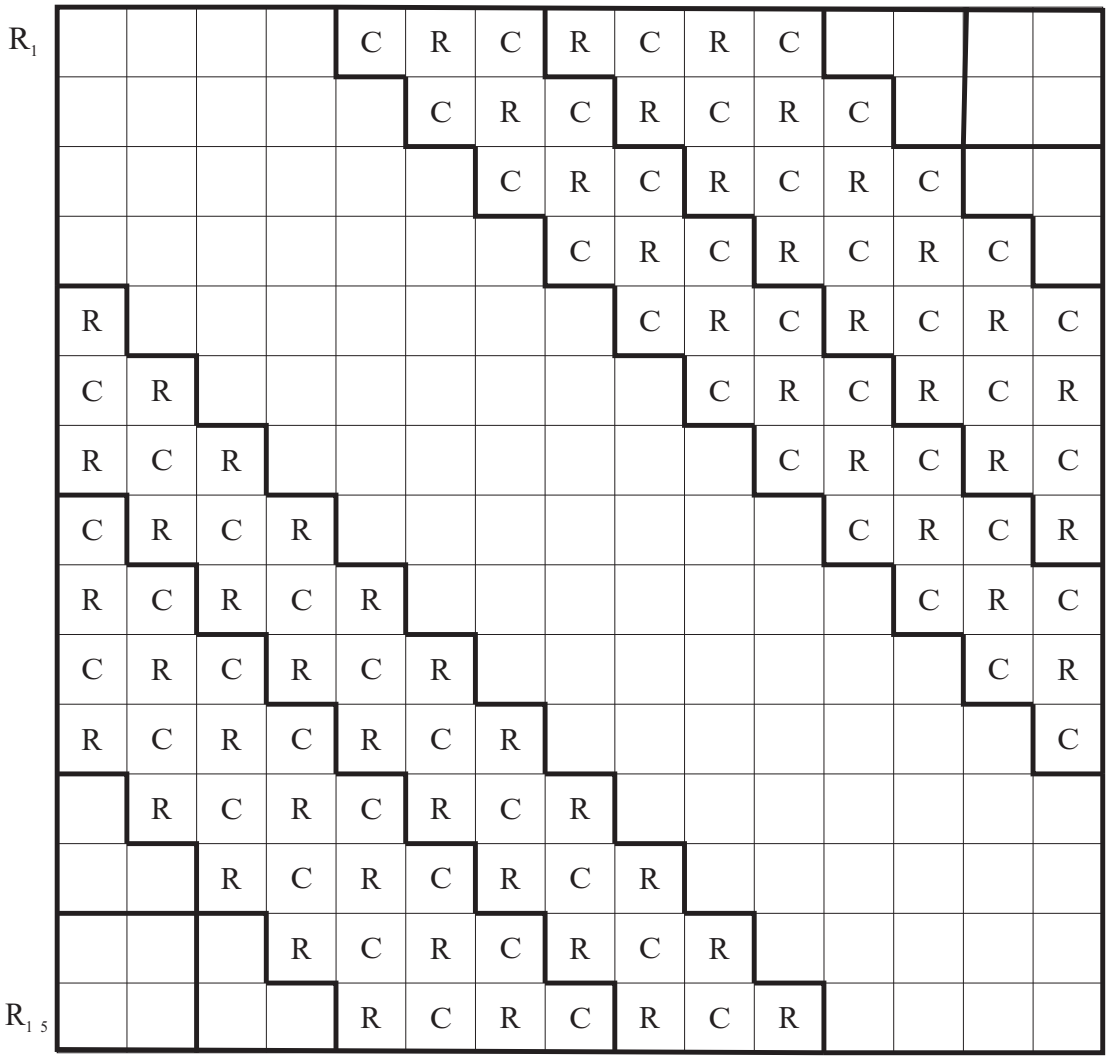

Figure 3.6: Schedule of the row and column buses on a 15×15 bused mesh.

a *R-labeled* processor will be called *C-item* or *R-item*, respectively. Notice that if a processor is C-labeled then its pair is R-labeled.
The next four steps (Step 2.1 - Step 2.4) are scheduled in parallel:

Step 2.1: Pairs in the region $\mathcal{E}$ walk by a greedy algorithm.

- Each element under the main diagonal first moves to its destination column, turns up, and moves step by step to its destination processor.
- Each element above the main diagonal first moves to its destination row, turns left and moves to its destination processor.

Step 2.2: Schedule the pairs of the regions $\mathcal{C}$ and $\mathcal{D}$ according to their labels assigned in Step 1. A row bus transfers messages belonging to a R-labeled processors using farthest first strategy, starting with the elements of the regions $\mathcal{D}$.

- Messages originated from the region $\mathcal{D}$ will walk along their destination column.
- Messages from the region $\mathcal{C}$ will use column buses,so their transfers require always only two steps.

The transfer of the items belonging to C-labeled processors is similar.

Step 2.3: Items of regions $\mathcal{B}$ in the positions $(1, n-z+1)$, (z, n), $(n-z+1, 1)$, and (n, z) walk to the direction of the center until they arrive at positions $(1, y+1), (n-y, n), (y+1, 1)$, and $(n, n-y)$, respectively.

Step 2.4: Schedule the newly arrived items (originated from the regions $\mathcal{B}$) into the region $\mathcal{C}$ to the corresponding row and column buses when the buses become available. The items have been moved up or down will be scheduled to row buses, the items moving left or right will be scheduled to column buses. Whenever such an item is transferred by a bus, we always move a new item to its former position to become transferable in the next step for the bus. After the first bus transfer, these items will wait for the second bus transfer in the processor buffers. To shorten the processor queues these items even can move in the direction of their destinations.

Step 3: When the above steps are finished, the algorithm transfers the items of regions $\mathcal{A}$ and those ones which are waiting in a buffer from Step 2.4 using row and column buses. Since they do not require common buses, the transfer of these two groups can be done in parallel.

3.6 Analysis of LBA

Lemma 3.6.1 ([19]) *Items of the region $\mathcal{E}$ can be routed to their destination in at most $\frac{n}{2}$ steps.*

Proof Since $\max_{p_{i,j}\in E}|p_{i,j}| = 2(n-x-1)$, using the definition of x we get that

$$\max_{p_{i,j}\in E}|p_{i,j}| = 2(n-x-1) = 2\left\lfloor\frac{\lfloor\frac{n}{2}\rfloor}{2}\right\rfloor \leq \left\lfloor\frac{n}{2}\right\rfloor \leq \frac{n}{2}.$$

Since all messages turn at the main diagonal, no items can delay each other.

□

Lemma 3.6.2 ([19]) *Items of the region $\mathcal{D}$ can be routed to their destination in at most $\frac{n}{2}$ steps.*

Proof We give the proof only for the R-items. Similar argument can be used for C-items. Step 2.2 uses farthest first strategy in the region $\mathcal{D}$ for the R-items. Consider an arbitrary row of the mesh. Order the R-items of the region $\mathcal{D}$ in this row by the distances from their pairs in decreasing order. The distance of the i-th item in this order from its pair is $2(n-y-i)$. By the schedule, the i-th item uses the bus in the i-th step, so for the total number of steps S required we get that

$$S = (n-y-i)+(i-1)+1 = n-y = n-\left\lceil\frac{n}{2}\right\rceil \leq \frac{n}{2}.$$

□

Consider now those rows from the region $\mathcal{C}$ which are indexed by $n-x+1,...,x$, i.e. the central rows of the mesh and denote them by $R_{n-x+1},...,R_x$. Let $|R_i|$ be the total number of bus operations required to route all the R-items in row R_i, $n-x+1 \leq i \leq x$, including also those row bus operations which route the C-items destined to the row R_i in the region $\mathcal{C}$.

Lemma 3.6.3 ([19]) $\max_i |R_i| < \frac{3n}{8}+4$ *where* $n-x+1 \leq i \leq n-y$.

Proof By a simple calculation we get the following formula for $|R_i|$.

$$\begin{aligned} |R_i| &= \left\lceil\frac{i-n+x}{2}\right\rceil + \left\lfloor\frac{x-y}{2}\right\rfloor + 2\left\lceil\frac{y-i+1}{2}\right\rceil \leq \\ &\leq \frac{i-n+x}{2}+\frac{x-y}{2}+y-i+3 = \\ &= x+\frac{y}{2}-\frac{n}{2}-\frac{i}{2}+3. \end{aligned}$$

From this it follows that

$$\begin{aligned}
\max_i |R_i| &= x + \frac{y}{2} - \frac{n}{2} - \frac{n-x+1}{2} + 3 = \\
&= \frac{3x}{2} + \frac{y}{2} - n + \frac{5}{2} = \\
&= \frac{3\left(n - \left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor - 1\right) + \lceil \frac{n}{2} \rceil - 2n}{2} + \frac{5}{2} \leq \\
&\leq \frac{n - \frac{3\lfloor \frac{n}{2} \rfloor}{2} + \frac{n}{2} + 1}{2} + \frac{5}{2} \leq \\
&\leq \frac{n - \frac{3n}{4} + \frac{n}{2} + \frac{5}{2}}{2} + \frac{5}{2} < \\
&< \frac{3n}{8} + 4.
\end{aligned}$$

□

Lemma 3.6.4 ([19]) $|R_{n-i+1}| \leq |R_i| + 1$ *where* $n - x + 1 \leq i \leq n - y$. *If* n *is odd then* $|R_{n-y+1}| = |R_{n-y}|$

Proof

$$\begin{aligned}
|R_{n-i+1}| &= \left\lfloor \frac{i-n+x}{2} \right\rfloor + \left\lceil \frac{x-y}{2} \right\rceil + 2\left\lfloor \frac{y-i+1}{2} \right\rfloor \leq \\
&\leq \left\lceil \frac{i-n+x}{2} \right\rceil + \left\lceil \frac{x-y}{2} \right\rceil + 2\left\lceil \frac{y-i+1}{2} \right\rceil \leq |R_i| + 1
\end{aligned}$$

If n is odd then

$$|R_{n-y+1}| = \left\lfloor \frac{x-y}{2} \right\rfloor + \left\lceil \frac{x-y}{2} \right\rceil + 2 = |R_{n-y}|$$

□

Corollary 3.6.5 ([19]) $\max_i |R_i| < \frac{3n}{8} + 5$ *where* $n - x + 1 \leq i \leq x$.

Lemma 3.6.6 ([19]) $|R_{n-y}| > x - y$.

Proof

$$|R_{n-y}| = \left\lceil \frac{x-y}{2} \right\rceil + \left\lfloor \frac{x-y}{2} \right\rfloor + 2\left\lceil \frac{2y-n+1}{2} \right\rceil > x - y.$$

□

Corollary 3.6.7 ([19]) *The items at positions* $(1, n-z+1), (z, n), (n-z+1, 1), (n, z)$ *arrive at positions* $(1, y+1), (n-y, n), (y+1, 1), (n, n-y)$ *in Step 2.3 of LBA before the row and column buses in rows* R_{n-y} *and* R_{y+1} *and in columns* C_{y+1} *and* C_{n-y} *finish their task defined in Step 2.2.*

Lemma 3.6.8 ([19]) *Step 2.4 needs at most* $\frac{3n}{8}+6$ *steps.*

Proof From Corollary (3.6.7) it follows that the buses in rows R_{n-y-i}, where $1 \leq i \leq z-u$ can continue their work with Step 2.4 immediately after finishing Step 2.2. We calculate the total number of operations defined in Steps 2.2 and 2.4 for the buses in rows R_{n-y-i}. Denote again this number by $|R_{n-y-i}|$. We get the following:

$$\begin{aligned} |R_{n-y-i}| &= \left\lceil \frac{x-y-i}{2} \right\rceil + \left\lceil \frac{x-y}{2} \right\rceil + 2 \left\lceil \frac{2y-n+i+1}{2} \right\rceil \\ &+ z-u-i+1 \leq \\ &\leq x+y+z-n-u-\frac{i}{2}+5. \end{aligned}$$

From this

$$\max_i |R_{n-y-i}| = x+y+z-n-u+\frac{9}{2} = y-u+\frac{9}{2} < \frac{3n}{8}+6. \tag{3.5}$$

From Corollary (3.6.5) and (3.5) the statement of the lemma follows.

□

Notice that when Step 2.4 is finished, then all the items in the region $\mathcal{C}$ have arrived at their pair.

Lemma 3.6.9 ([19]) *Step 2.2 finishes in at most* $\frac{n}{8}+3$ *steps.*

Proof Since $u < \frac{n}{8}+2$, the items in the region $\mathcal{A}$ and the waiting items from Step 2.4 can be routed in at most $\frac{n}{8}+3$ steps using bus operations.

□

Combining the statements of Lemmas (3.6.1),(3.6.2),(3.6.8),(3.6.9) we get the following theorem.

Theorem 3.6.10 ([19]) *Algorithm LBA solves the matrix transpose problem in less than* $\frac{n}{2}+9$ *steps.*

The next table shows on which step the specific elements arrive at their destinations on a 10x10 mesh. The second code presents the way of transfer in the last step, which can be walking (W), row bus (R) and column bus(C).

0	2W	4W	5W	5W	5C	5R	5C	4R	4C
4W	0	2W	5W	5W	5W	3C	5R	4C	4R
4W	4W	0	2W	5W	5W	5W	3C	3R	4R
5W	4W	4W	0	2W	5W	4W	5W	5C	4R
5W	5W	4W	4W	0	2W	4W	4W	5W	5C
5R	5W	5W	4W	4W	0	4W	4W	4W	5W
5C	3R	5W	4W	4W	4W	0	4W	4W	4W
5R	5C	3R	5W	4W	4W	2W	0	4W	4W
4C	4R	3C	5R	5W	4W	4W	2W	0	4W
4R	4C	4C	4C	5R	5W	4W	4W	2W	0

Finally, we show some values of the upper bound on the required steps and the corresponding lower bounds:

n	10	100	500	1000	4000
LB	5	45	226	451	1803
MTB	14	59	259	509	2009
Ratio	2.8	1.3111	1.1460	1.1286	1.1142

3.7 Conclusions

In this section we considered the matrix transposition problem for parallel machines, and we proved that using meshes with buses one can expect better results than for those architectures where buses are not available.

Chapter 4

Route Planning for Public Transport

4.1 Introduction

Nowadays, the use of computer-based route planners is widespread among passengers. Individual passengers have a large choice of websites and GPS navigation devices. Generally speaking, with these kind of systems the available road network is modeled by a graph. The vertices of the graph represent the points of contact of the roads, while the edges represent the road sections connecting the points. If we assign the length of the road section to the edges, we get a weighted graph. We can use the well-known Dijkstra algorithm to calculate the shortest distance between two points. The efficiency of the Dijkstra algorithm is quite good, but the size of the graphs describing real-world road networks can be very large, especially in the case of a larger geographical area like the road network of a continent. As passengers generally expect an almost immediate response to their searches, even the Dijkstra algorithm with polynomial running time is not good enough for large graphs. Over the last two decades, many potential speed-ups have been investigated to address the problem. More details about this can be found in [34].

Routing services are available not only for private passengers, but for passengers travelling by public transportation as well. In this case, usually the road network does not need to be described by the graph. The reason for this is that the route is pre-determined, or the journey is often not on a conventional road network but on a bound track (e.g. rail) or in the air, or even on water. The nodes of public transport routing graphs generally represent the vehicle stops, and the edges provide travel possibilities between stops. The weight of the edges may represent the travel time, but as different users may look at travel from different aspects, other models should be considered. Public transport networks may generally be larger than those of road networks, given that travel is time-dependent, and handling this is also required in the model. Therefore, in this case, the opportunities for speeding-up searches are particularly important, and they have also been widely investigated. Some of the speed-up options available on road graphs

can be used here, but not those that use the special features of road networks. For more details on models and speed-up options, see [15, 64].

In practice, it is usual for public transportation companies to provide route planners for their own service area. This usually includes the services of the company or other companies closely related to that city, region or country. In most cases, however, these services provide only route planning for one kind of journey; for example, a route planner of a rail company can be searched for rail routes, while route planners of bus companies can be used for bus routes. In the case of local transport, it is common that there are search engines that cover different modes of transport in a given city, such as buses, trams and metro, trains, but they are usually not intended for long-distance transport. So, for instance, if a passenger wants to get from a point in a city to a point in another city, using local and then several long-distance modes of transport, he or she usually will not find a search engine that offers such a travel option, not even in a country or in a region. And the graphs of road networks can handle the same problem for larger areas such as continents. This suggests that finding a route to public transport is a more difficult issue than planning private trips.

In this chapter, we present a route planning system and its search algorithm, where the latter operates on a complete public transport network of two regions from two countries, namely Hungary and Serbia [17]. The databases on which the model is based include long-distance trains, buses and complete local transport of the major cities. The databases are based on timetables taken from service providers operating in the regions. The system can also model the pedestrian traffic between stops not too far from each other. It can take into account individual user preferences, like walking distances, modes of transport, and properties of the objective function. The graph representing the transport network was very large, but with the help of some speed-up techniques, we managed to create an effective search algorithm that is able to handle user requirements.

The algorithm was developed under the EU-funded Hungary-Serbia Interreg-IPA CBC Program as part of a complex route planning application. This study presents the most common methods used in the search algorithm, as well as the experiences and results obtained during the practical running.

4.2 Definitions and preliminaries

The simplest modelling of public transport networks can be achieved using the so-called Station Graph [77]. In this case, the graph's nodes represent the stops and edges only exist between two stops if one of the stops can be reached directly from the other. Generally speaking, the Station Graph is a relatively small graph, since each stop has exactly one node and there is at most one edge between any two stops. However, this graph contains limited information as it does not represent the time of the routes at all. Hence it is not suitable for precise route planning, and it can only provide a poor estimate for the length of the trip or for the minimum required transfers.

The Time-Dependent model may be regarded as an extension of the Station

Graph [26], in which the edges may have multiple weights. During a route search, the current weight is calculated using the given time, taking into account the bus lines departing from the stop. In this case, the size of the graph does not increase, but the determination of the weight requires more complex calculations, which may slow down the search. The model has often been studied on railway networks and has been developed, for instance, for the correct modelling of transfers [69].

The most common model used is the Time-Expanded model [76]. Here, the starting and arrival times of the vehicles are represented by special nodes. The nodes belonging to a station can be sorted by time and the waiting can be represented by edges. Trips between the different stations can be expressed by edges between the appropriate departure and arrival times. Therefore the model can handle transfers, and pedestrian traffic between two specific stops can also be modelled. Initially, due to the large size of the graph, searches in this model were not sufficiently fast, but due to technological developments and improvements, we can now consider it a competitive method [32].

While the aim of search engines is normally to find the shortest route between departures and arrivals, this is not always the case for public transport networks. Passengers may view things from different aspects; some may want to minimize the number of passes against the earliest arrival, while others may want to keep their travel costs as low as possible. What is more, we usually choose one mode of transport on the route; for example, like that for a motorist or pedestrian. With public transport, someone may want to use a variety of vehicles or does not even have the opportunity to reach the goal otherwise; for example, he or she needs to combine buses and railways, and has to reach the stop on foot. Hence special models have also been developed for public transport networks. This is why besides the classical earliest arrival problem, [76] multiobjective optimization methods were applied [64].

The research work of recent years has focused mainly on developing complex multiobjective, multimodal route-planning algorithms [14, 59], which include appropriate speed-up techniques [33].

4.3 Modeling

All three types of graphs described above were used to model our transport network. Only the Station Graph and the Time-Dependent Graph built on it were stored permanently in the memory. The number of edges of the Time-Dependent Graph was already quite large as there were seven different timetable versions in use during that period. This meant that searches for different days and periods were subject to different schedules. For example, the Sunday schedule was different from the weekday one. In the Time-Dependent Graph there were different edges for the different lines, and the departure and arrival times of the lines were stored in separate structures. It also included possible walking routes. And the Station graph and the Time Dependent graph did not have any direct search, but only had roles in the preprocessing of searches. In the actual search, the Dijkstra algorithm was implemented on a Time Expanded Graph. The current Time

Expanded Graph was not stored permanently in memory as the many different timetable versions dynamically changed it. The generation of the current graph was always related to the actual query.

Graph type	Number of nodes	Number of edges
Station	12014	416163
Time Dependent	12014	6786662
Time Expanded	≈ 562000	≈ 4616000

Table 4.1: Sizes of the graphs.

Table 4.1 lists the sizes of different graphs. Figure 4.1 depicts the modelling of

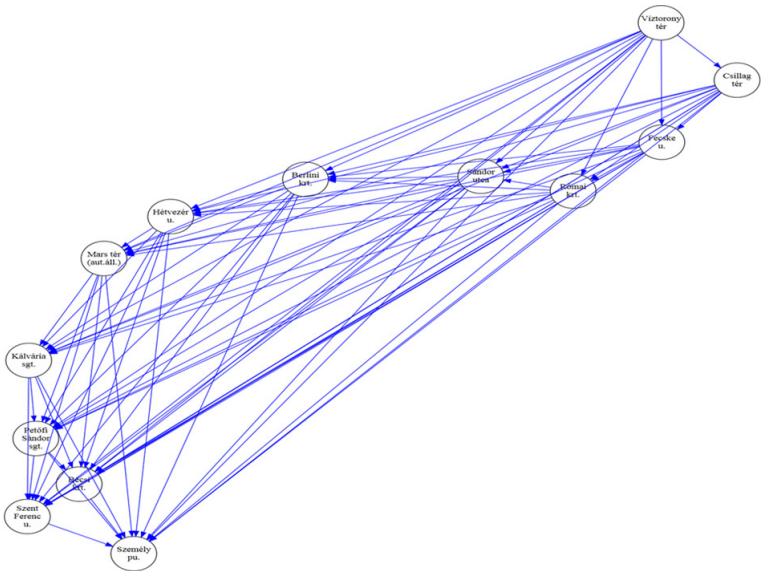

Figure 4.1: The Station Graph of a line.

a line with the Station Graph. Figure 4.2 shows a part of the Time Expanded Graph. The horizontally positioned nodes are part of a stop's timeline, and the pink nodes represent the departure times, the green ones represent the arrival times. The orange edges are the waiting edges of the timeline. The black edges model the lines between the stops, while the red edges show the walking options. The departure timeline is in the upper line, the arrival is in the lowest one. Thickly marked paths indicate the travel possibilities chosen by the system.

4.3.1 Search algorithm

After producing the Time Expanded Graph and weighing the edges appropriately, with the help of the Dijsktra algorithm we can find the shortest path. In our case, we came up against a difficulty caused by the search dependence of the Time Expanded Graph. Its structure was influenced by the time specified in the search and by the modes of transport that the user wanted to use, or by other

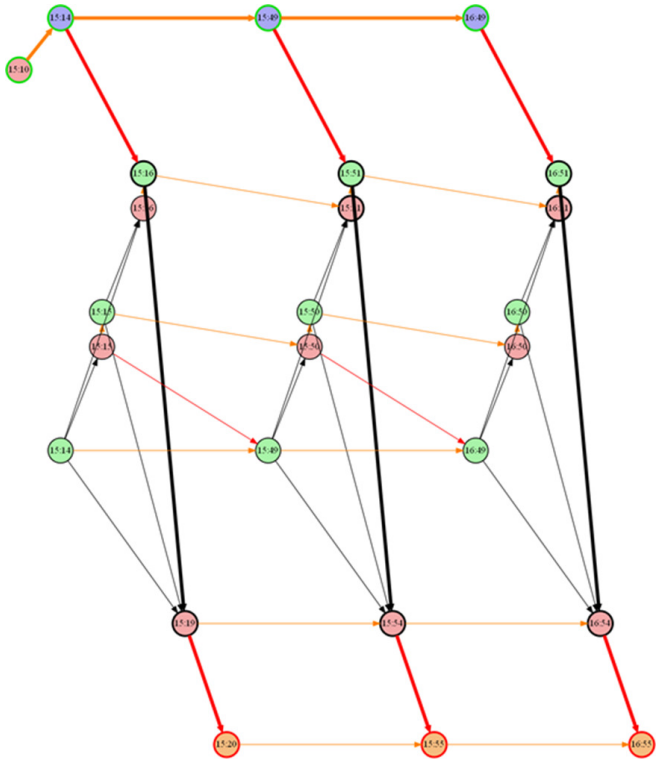

Figure 4.2: Part of the Time Expanded Graph.

parameters such as the maximum allowed walking distance. Despite the large size of the graph, its generation was relatively fast, but after including the search time, in some cases we could not achieve the desired speed with this method. In addition, the structure of the entire graph resulted in excessive memory usage during a search, which was a real problem, because multiple clients wanted to use the services of the server simultaneously.

To overcome the problems listed above, we performed a preprocessing step before producing the Time Expanded Graph and the search. The aim was to determine the nodes which include some of the shortest paths corresponding to the user parameters and the objective function. The method is similar to the TRANSIT algorithms described in the literature [59]. In the end, the Time Expanded Graph contained far fewer nodes, and it led to a significant speed-up of the search.

The purpose of the preprocessing was to determine the nodes that lie on the shortest paths from the source to the target. Here, we used the Station Graph for the preprocessing part. Our aim was to determine all the nodes that lie on the maximum k-length paths between the source and destination. In this case, the parameter k means the maximum number of transfers that can be made during the journey. Here, we used a modified version of the depth first search algorithm that did not take into account the nodes farther from the source than the desired length. Given that in some cases the execution of the procedure might take longer, the value of the parameter k and the execution time were also limited in the search. Experience has shown that these heuristics worked well in practice so, in almost

every case, the Time Expanded Graph built on these nodes contained the shortest path of the entire graph. For a more detailed analysis of this, see section 4.4.

4.3.2 The objective function

In general, different objective functions are used for public transport route planning. One of the most common is the earliest arrival, but it does not always fully meet the user's preferences. Finding different Pareto optima is also a common problem. We used an objective function that included a weighted sum of different goals. And the weights were determined based on information provided by the users, using preset patterns.

The weight of a travel edge is the following: $t_r w_r + f_r$, where t_r is the time of travel, w_r is the actual weight of the journeys, and f_r is the additive factor for the travel cost.

The weight of a waiting edge is the following: $t_i w_i$, where t_i is the waiting time, and w_i is the actual weight of the waiting.

The weight of a pedestrian edge is the following: $t_a w_a + f_a$, where t_a is the time of walking, w_a is the actual weight of the walking, and f_a is the additive factor for pedestrian cost.

In the search, there were three options available to the users regarding the search query. These three options are the fastest route, the minimum number of connections and the minimum walking distance. And the parameters of the weights for each goal are given in Table 4.2.

Options	w_r	f_r	w_i	w_a	f_a
Fastest	1	30	0.8	1.5	30
Less transfer	1	1000	0.8	1.5	10
Less walking	1	30	0.8	30	1000

Table 4.2: Parameters of the weights.

4.4 Results and analysis

We investigated the proportion of the investigated instances such that the reduced graph contained all nodes of the shortest path. This ensured that the search on the reduced Time Expanded Graph would give an optimal solution. In the pre-processing heuristics we modified two parameters. One was the previously mentioned k parameter, while the other was the search time. For the possible values of k, we selected s+1 and s+2, where s is the shortest path between two points in the Station Graph.

Here, we used two types of test queries. They both contained 1000 queries, one of which concerned questions regarding local traffic (T1) and the other concerned long-distance traffic (T2). In the table below, we summarize how efficient the pre-processing heuristics was for the two types of test data, for different values of

k. In the table out of the 1000 questions we see how many times the reduced-size Time Expanded Graph gave a solution that differed from the optimum, which we calculated using the complete Time Expanded Graph.

Input type	$k = s + 1$	$k = s + 2$
T1	13	0
T2	54	11

Table 4.3: Non-optimality statistics of the preprocessing heuristics. ([17])

Tables 4.4 and 4.5 contain the average and maximum search times in milliseconds on the above-mentioned query lists for the reduced and for the complete Time Expanded Graphs. The notations used for the columns are the following:
RB: Reduced graph build time
RS: Reduced graph search time
RC (=RB+RS): Reduced graph complete search time
FB: Full graph build time
FS: Full graph search time
FC (=FB+FS): Full graph complete search time

	RB	RS	RC	FB	FS	FC
$T1, k = s + 1$	308	18	327	5327	166	5493
$T1, k = s + 2$	1394	406	1801			
$T2, k = s + 1$	256	55	312	5427	1137	6564
$T2, k = s + 2$	1186	316	1502			

Table 4.4: Average running times in milliseconds.([17])

	RB	RS	RC	FB	FS	FC
$T1, k = s + 1$	1478	446	1529	5744	4130	9502
$T1, k = s + 2$	4959	2458	5453			
$T2, k = s + 1$	2491	788	3279	6290	10216	15650
$T2, k = s + 2$	3119	3938	6469			

Table 4.5: Maximal running times in milliseconds.([17])

4.5 Conclusions

We presented an algorithm used by a route planning system for a complete public transport network of two regions from Hungary and Serbia. The graph representing the transport network was very large, but with the help of some speed-up techniques, we managed to create an effective search algorithm that is able to handle user requirements.

Bibliography

[1] Alexander A. Ageev and Alexei E. Baburin. Approximation algorithms for UET scheduling problems with exact delays. *Operations Research Letters*, 35(4):533–540, July 2007.

[2] Dino Ahr, József Békési, Gábor Galambos, Marcus Oswald, and Gerhard Reinelt. An exact algorithm for scheduling identical coupled tasks. *Mathematical Methods of Operations Research (ZOR)*, 59(2):193–203, June 2004.

[3] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, May 1997.

[4] L. Babel, B. Chen, H. Kellerer, and V. Kotov. Algorithms for on-line bin-packing problems with cardinality constraints. *Discrete Applied Mathematics*, 143(1-3):238–251, 2004.

[5] B. S. Baker and E. G. Coffman, Jr. A tight asymptotic bound for next-fit-decreasing bin-packing. *SIAM J. on Algebraic and Discrete Methods*, 2(2):147–152, 1981.

[6] János Balogh, József Békési, Gábor Dósa, György Galambos, and Zhiyi Tan. Lower bound for 3-batched bin packing. *Discrete Optimization*, 21:14–24, 2016.

[7] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A New and Improved Algorithm for Online Bin Packing. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[8] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. Online bin packing with cardinality constraints resolved. *Journal of Computer and System Sciences*, 112:34–49, September 2020.

[9] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. Lower bounds for batched bin packing. *Journal of Combinatorial Optimization*, August 2021.

[10] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new lower bound for classic online bin packing. *Algorithmica*, 83(7):2047–2062, March 2021.

[11] János Balogh, József Békési, György Dósa, Jiří Sgall, and Rob van Stee. The optimal absolute ratio for online bin packing. *Journal of Computer and System Sciences*, 102:1–17, June 2019.

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

[13] Amotz Bar-Noy, Ari Freund, and Joseph (Seffi) Naor. New algorithms for related machines with temporary jobs. *Journal of Scheduling*, 3(5):259–272, 2000.

[14] Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *Algorithms – ESA 2010*, pages 290–301. Springer Berlin Heidelberg, 2010.

[15] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. In *Algorithm Engineering*, pages 19–80. Springer International Publishing, 2016.

[16] J. Békési, Gy. Dósa, and L. Epstein. Bounds for online bin packing with cardinality constraints. *Information and Computation*, 249:190–204, 2016.

[17] József Békési. Regional multicriteria and multimodal route planning system for public transportation: A case study. *Acta Cybernetica*, 23(3):773–782, 2018.

[18] József Békési, György Dósa, and Gábor Galambos. A first fit type algorithm for the coupled task scheduling problem with unit execution time and two exact delays. *European Journal of Operational Research*, 297(3):844–852, March 2022.

[19] József Békési and Gábor Galambos. Matrix transpose on meshes with buses. *Journal of Parallel and Distributed Computing*, 96:194–201, October 2016.

[20] József Békési and Gábor Galambos. Tight bounds for NF-based bounded-space online bin packing algorithms. *Journal of Combinatorial Optimization*, 35(2):350–364, September 2017.

[21] József Békési, Gábor Galambos, and Péter Hajnal. Analysis of permutation routing algorithms. *European Journal of Operational Research*, 125(2):249–256, September 2000.

[22] József Békési, Gábor Galambos, Michael N. Jung, Marcus Oswald, and Gerhard Reinelt. A branch-and-bound algorithm for the coupled task problem. *Mathematical Methods of Operations Research*, 80(1):47–81, April 2014.

[23] József Békési, Gábor Galambos, Marcus Oswald, and Gerhard Reinelt. Improved analysis of an algorithm for the coupled task problem with UET jobs. *Operations Research Letters*, 37(2):93–96, March 2009.

[24] Piotr Berman, Moses Charikar, and Marek Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35(1):108–121, April 2000.

[25] A. Borodin and J.E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30(1):130–145, February 1985.

[26] Gerth Stølting Brodal and Riko Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electronic Notes in Theoretical Computer Science*, 92:3–15, February 2004.

[27] Sheng-Yi Cai and Qi-Fan Yang. Online scheduling on three uniform machines. *Discrete Applied Mathematics*, 160(3):291–302, February 2012.

[28] Yee Kit Chan and Voon Chet Koo. AN INTRODUCTION TO SYNTHETIC APERTURE RADAR (SAR). *Progress In Electromagnetics Research B*, 2:27–60, 2008.

[29] S. Cheung and F.C.M. Lau. A lower bound for permutation routing on two-dimensional bused meshes. *Information Processing Letters*, 45(5):225–228, April 1993.

[30] Yookun Cho and Sartaj Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, February 1980.

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

[32] Daniel Delling. *Engineering and augmenting route planning algorithms*. PhD thesis, Universität Karlsruhe, Fakultät für Informatik, 2009.

[33] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating multimodal route planning by access-nodes. In *Lecture Notes in Computer Science*, pages 587–598. Springer Berlin Heidelberg, 2009.

[34] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Engineering time-expanded graphs for faster timetable information. In *Robust and Online Large-Scale Optimization*, pages 182–206. Springer Berlin Heidelberg, 2009.

[35] Kuo-Shun Ding, Ching-Tien Ho, and Jyh-Jong Tsay. Matrix transpose on meshes with wormhole and XY routing. In *Proceedings of 1994 6th IEEE Symposium on Parallel and Distributed Processing*, pages 656–663. IEEE Comput. Soc. Press, 1994.

[36] Gy. Dósa. Batched bin packing revisited. *Journal of Scheduling*, 20(2):199–209, 2017.

[37] Tomáš Ebenlendr and Jiří Sgall. A lower bound on deterministic online algorithms for scheduling on related machines without preemption. *Theory of Computing Systems*, 56(1):73–81, February 2013.

[38] Leah Epstein, John Noga, Steve Seiden, Jiří Sgall, and Gerhard Woeginger. Randomized on-line scheduling on two uniform machines. *Journal of Scheduling*, 4(2):71–92, 2001.

[39] Leah Epstein and Jiří Sgall. A lower bound for on-line scheduling on uniformly related machines. *Operations Research Letters*, 26(1):17–22, February 2000.

[40] S. Fujita and M. Yamashita. Fast gossiping on mesh-bus computers. *IEEE Transactions on Computers*, 45(11):1326–1330, 1996.

[41] H. Fujiwara and K. M. Kobayashi. Improved lower bounds for the online bin packing problem with cardinality constraints. *Journal of Combinatorial Optimization*, 29(1):67–87, 2015.

[42] G. Galambos. Parametric lower bound for on-line bin-packing. *SIAM Journal on Algebraic Discrete Methods*, 7(3):362–367, July 1986.

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

[44] 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. In *Discrete Optimization II, Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver*, pages 287–326. Elsevier, 1979.

[45] Ananth Grama, George Karypis, Vipin Kumar, and Anshul Gupta. *Introduction to Parallel Computing*. Pearson, London, 2003.

[46] E.F. Grove. Online bin packing with lookahead. In *Proceedings of the sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 430–436, 1995.

[47] G. Gutin, T. Jensen, and A. Yeo. Batched bin packing. *Discrete Optimization*, 2(1):71–82, 2005.

[48] Fangqiu Han, Zhiyi Tan, and Yang Yang. On the optimality of list scheduling for online uniform machines scheduling. *Optimization Letters*, 6(7):1551–1571, May 2011.

[49] X. Han, F. Y. L. Chin, H.-F. Ting, G. Zhang, and Y. Zhang. A new upper bound 2.5545 on 2D online bin packing. *ACM Transactions on Algorithms*, 7(4):50:1–50:18, 2011.

[50] S. Heydrich and R. van Stee. Beating the harmonic lower bound for online bin packing. In *Proc. of 43rd International Colloquium on Automata, Languages, and Programming (ICALP2016)*, pages 41:1–41:14, 2016.

[51] Kazuo Iwama, Eiji Miyano, and Yahiko Kambayashi. Routing problems on the mesh of buses. *Journal of Algorithms*, 20(3):613–631, May 1996.

[52] Kazuo Iwama, Eiji Miyano, Satoshi Tajima, and Hisao Tamaki. Efficient randomized routing algorithms on the two-dimensional mesh of buses. *Theoretical Computer Science*, 261(2):227–239, June 2001.

[53] Łukasz Jeż, Jarett Schwartz, Jiří Sgall, and József Békési. Lower bounds for online makespan minimization on a small number of related machines. *Journal of Scheduling*, 16(5):539–547, September 2012.

[54] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.

[55] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.

[56] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3:256–278, 1974.

[57] M. Kaufmann, U. Meyer, and J.F. Sibeyn. Matrix transpose on meshes: theory and practice. In *Proceedings 11th International Parallel Processing Symposium*, pages 315–319. IEEE Comput. Soc. Press, 1997.

[58] M. Kaufmann, R. Raman, and J. F. Sibeyn. Routing on meshes with buses. *Algorithmica*, 18(3):417–444, July 1997.

[59] Jolanta Koszelew. Two methods of quasi-optimal routes generation in public transportation network. In *2008 7th Computer Information Systems and Industrial Management Applications*. IEEE, June 2008.

[60] Manfred Kunde. Routing and sorting on mesh-connected arrays. In *VLSI Algorithms and Architectures*, pages 423–433. Springer New York, 1988.

[61] C. C. Lee and D. T. Lee. A simple online bin packing algorithm. *Journal of the ACM*, 32(3):562–572, 1985.

[62] F. T. Leighton. Average case analysis of greedy routing algorithms on arrays. In *Proceedings of the second annual ACM symposium on Parallel algorithms and architectures - SPAA '90*. ACM Press, 1990.

[63] Frank M Liang. A lower bound for on-line bin packing. *Information processing letters*, 10(2):76–79, 1980.

[64] Matthias Müller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Timetable information: Models and algorithms. In F. Geraets, L. Kroon, A. Schoebel, D. Wagner, and C.D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization. Lecture Notes in Computer Science*, pages 67–90. Springer Berlin Heidelberg, 2007.

[65] Antoine Musitelli and Jean-Marc Nicoletti. Competitive ratio of list scheduling on uniform machines and randomized heuristics. *Journal of Scheduling*, 14(1):89–101, May 2010.

[66] Ted Nesson and S. Lennart Johnsson. ROMM routing on mesh and torus networks. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures - SPAA '95*. ACM Press, 1995.

[67] A.J. Orman and C.N. Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72(1-2):141–154, January 1997.

[68] Behrooz Parhami. *Introduction to Parallel Processing*. Kluwer Academic Publishers, 2002.

[69] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM Journal of Experimental Algorithmics*, 12:1–39, June 2008.

[70] P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee. Online bin packing in linear time. *Journal of Algorithms*, 10:305–326, 1989.

[71] Abhijeet A. Ravankar and Stanislav G. Sedukhin. Mesh-of-tori: A novel interconnection network for frontal plane cellular processors. In *2010 First International Conference on Networking and Computing*. IEEE, November 2010.

[72] Abhijeet A. Ravankar and Stanislav G. Sedukhin. An o(n) time-complexity matrix transpose on torus array processor. In *2011 Second International Conference on Networking and Computing*. IEEE, November 2011.

[73] M. B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34(1–3):203–227, 1991.

[74] R. van Stee S. Heydrich. Beating the harmonic lower bound for online bin packing. *The Computing Res. Rep. (CoRR)*, abs/1707.01728, 2017.

[75] C P Schnorr and A Shamir. An optimal sorting algorithm for mesh connected computers. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing - STOC '86*. ACM Press, 1986.

[76] Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line. *ACM Journal of Experimental Algorithmics*, 5:12, December 2000.

[77] Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Using multi-level graphs for timetable information in railway systems. In *Algorithm Engineering and Experiments*, pages 43–59. Springer Berlin Heidelberg, 2002.

[78] Stanislav G. Sedukhin and Marcin Paprzycki. Generalizing matrix multiplication for efficient computations on modern computers. In *Parallel Processing and Applied Mathematics*, pages 225–234. Springer Berlin Heidelberg, 2012.

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

[80] Roy D. Shapiro. Scheduling coupled tasks. *Naval Research Logistics Quarterly*, 27(3):489–498, September 1980.

[81] Quentin F. Stout and Bruce Wagar. Intensive hypercube communication prearranged communication in link-bound machines. *Journal of Parallel and Distributed Computing*, 10(2):167–181, October 1990.

[82] J. J. Sylvester. On a point in the theory of vulgar fractions. *American Journal of Mathematics*, 3(4):332, December 1880.

[83] Jyh-Jong Tsay, KS Ding, and WT Wang. Optimal algorithm for matrix transpose on wormhole-switched meshes. *Journal of Information Science and Engineering*, 19(1):167–177, 2003.

[84] J. D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, Princeton, NJ, 1971.

[85] A. van Vliet. An improved lower bound for online bin packing algorithms. *Information Processing Letters*, 43(5):277–284, 1992.

[86] A. C. C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27:207–227, 1980.

[87] Wenci Yu, Han Hoogeveen, and Jan Karel Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5):333–348, September 2004.

[88] Minghui Zhang, Xin Han, Yan Lan, and Hing-Fung Ting. Online bin packing problem with buffer and bounded size revisited. *Journal of Combinatorial Optimization*, 33(2):530–542, December 2015.

[89] Feifeng Zheng, Li Luo, and E. Zhang. NF-based algorithms for online bin packing with buffer and bounded item size. *Journal of Combinatorial Optimization*, 30(2):360–369, July 2014.

Acknowledgement

The author is grateful to his co-authors, János Balogh, György Dósa, Gábor Galambos, Leah Epstein, Łukasz Jeż, Asaf Levin, Marcus Oswald, Gerhard Reinelt, Jarret Schwartz, Jiří Sgall and Zhiyi Tan. Without their contribution, the articles containing the results would not have been possible.

Appendix A: The parameters of the algorithm AH

Class index: j	Left endpoint t_j	Right endpoint t_{j-1}	i	α_{ij} or $\alpha_{i,j}$
1	$\frac{1}{2} = 0.5$	1	1	
2	$\frac{3}{7} \approx 0.42857$	$\frac{1}{2} = 0.5$	2	1
3	$\frac{43}{120} \approx 0.35833$	$\frac{3}{7} \approx 0.42857$	1	$\frac{31755722}{150095589} \approx 0.2115$
3			2	$\frac{118339867}{150095589} \approx 0.7884$
4	$\frac{59}{166} \approx 0.35542$	$\frac{43}{120} \approx 0.35833$	1	$\frac{33382666}{150909061} \approx 0.2212$
4			2	$\frac{117526395}{150909061} \approx 0.7787$
5	$\frac{7}{20} = 0.35$	$\frac{59}{166} \approx 0.35542$	1	$\frac{4493270}{19023851} \approx 0.2361$
5			2	$\frac{14530581}{19023851} \approx 0.7638$
166	$\frac{271}{960} \approx 0.28229$	$\frac{1}{3} \approx 0.33333$	1	$\frac{3445801}{1433952966} \approx 0.0024$
166			3	$\frac{1430507165}{1433952966} \approx 0.9975$
167	$\frac{1}{4} = 0.25$	$\frac{271}{960} \approx 0.28229$	1	$\frac{18718929}{79588150} \approx 0.2351$
167			3	$\frac{60869221}{79588150} \approx 0.7648$
168	$\frac{97}{480} \approx 0.20208$	$\frac{1}{4} = 0.25$	1	$\frac{10193524}{41199575} \approx 0.2474$
168			4	$\frac{31006051}{41199575} \approx 0.7525$
169	$\frac{1}{5} = 0.2$	$\frac{97}{480} \approx 0.20208$	1	$\frac{22658284}{84102577} \approx 0.26945$
169			4	$\frac{61444293}{84102577} \approx 0.7305$
170	$\frac{15}{88} \approx 0.17045$	$\frac{1}{5} = 0.2$	5	1
171	$\frac{1}{6} \approx 0.16667$	$\frac{15}{88} \approx 0.17045$	1	$\frac{76872685}{1135239972} \approx 0.0677$
171			5	$\frac{1058367287}{1135239972} \approx 0.9322$
172	$\frac{3}{20} = 0.15$	$\frac{1}{6} \approx 0.16667$	2	$\frac{24797889}{191010959} \approx 0.1298$
172			1	$\frac{48313566}{191010959} \approx 0.2529$
172			6	$\frac{117899504}{191010959} \approx 0.6172$
173	$\frac{12}{83} \approx 0.14458$	$\frac{3}{20} = 0.15$	2	$\frac{158075552}{480752651} \approx 0.3288$
173			1	$\frac{20946010}{480752651} \approx 0.0435$
173			6	$\frac{301731089}{480752651} \approx 0.6276$

174	$\frac{1}{7} \approx 0.14286$	$\frac{12}{83} \approx 0.14458$	2	$\frac{5682641}{14973238} \approx 0.3795$
174			6	$\frac{9290597}{14973238} \approx 0.6204$
175	$\frac{11}{83} \approx 0.13253$	$\frac{1}{7} \approx 0.14286$	2	$\frac{11653567744}{42727973215} \approx 0.2727$
175			1	$\frac{103268403}{85455946430} \approx 0.0012$
175			7	$\frac{62045542539}{85455946430} \approx 0.7260$
176	$\frac{1}{8} = 0.125$	$\frac{11}{83} \approx 0.13253$	2	$\frac{4313813}{11469903} \approx 0.3760$
176			7	$\frac{7156090}{11469903} \approx 0.6239$
177	$\frac{1}{9} \approx 0.1111$	$\frac{1}{8} = 0.125$	2	$\frac{35844844}{93992497} \approx 0.3813$
177			8	$\frac{58147653}{93992497} \approx 0.6186$
178	$\frac{1}{10} = 0.1$	$\frac{1}{9} \approx 0.1111$	2	$\frac{145576935}{381661961} \approx 0.3814$
178			9	$\frac{236085026}{381661961} \approx 0.6185$
179	$\frac{1}{11} \approx 0.09091$	$\frac{1}{10} = 0.1$	2	$\frac{8723245}{23755812} \approx 0.3672$
179			10	$\frac{15032567}{23755812} \approx 0.6327$
180	$\frac{1}{12} \approx 0.08333$	$\frac{1}{11} \approx 0.09091$	3	$\frac{145045373}{508140728} \approx 0.2854$
180			11	$\frac{363095355}{508140728} \approx 0.7145$
181	$\frac{1}{13} \approx 0.07692$	$\frac{1}{12} \approx 0.08333$	3	$\frac{16276212}{45761591} \approx 0.3556$
181			12	$\frac{29485379}{45761591} \approx 0.6443$
182	$\frac{1}{14} \approx 0.07143$	$\frac{1}{13} \approx 0.07692$	3	$\frac{72087509}{189669658} \approx 0.38$
182			13	$\frac{117582149}{189669658} \approx 0.6199$
183	$\frac{1}{15} \approx 0.06667$	$\frac{1}{14} \approx 0.07143$	3	$\frac{36413948928}{97499546341} \approx 0.3734$
183			1	$\frac{182118174}{97499546341} \approx 0.0018$
183			14	$\frac{60903479239}{97499546341} \approx 0.6246$
184	$\frac{1}{16} = 0.0625$	$\frac{1}{15} \approx 0.06667$	4	$\frac{36527825}{116265557} \approx 0.3141$
184			15	$\frac{79737732}{116265557} \approx 0.6858$
185	$\frac{1}{17} \approx 0.05882$	$\frac{1}{16} = 0.0625$	4	$\frac{30799804}{90208717} \approx 0.3414$
185			16	$\frac{59408913}{90208717} \approx 0.6585$
186	$\frac{1}{18} \approx 0.05556$	$\frac{1}{17} \approx 0.05882$	4	$\frac{61076393}{180923205} \approx 0.3375$
186			17	$\frac{119846812}{180923205} \approx 0.6624$
187	$\frac{1}{19} \approx 0.05263$	$\frac{1}{18} \approx 0.05556$	5	$\frac{246282282}{848959177} \approx 0.2900$
187			18	$\frac{602676895}{848959177} \approx 0.7099$
188	$\frac{1}{20} = 0.05$	$\frac{1}{19} \approx 0.05263$	5	$\frac{3612237}{11491762} \approx 0.3143$
188			19	$\frac{7879525}{11491762} \approx 0.6856$
189	$\frac{1}{21} \approx 0.04762$	$\frac{1}{20} = 0.05$	5	$\frac{1086689792}{34169004389} \approx 0.3244$
189			3	$\frac{99039140}{34169004389} \approx 0.0028$
189			20	$\frac{22983275457}{34169004389} \approx 0.6726$
190	$\frac{1}{22} \approx 0.04545$	$\frac{1}{21} \approx 0.0476$	5	$\frac{1773973504}{5453794899} \approx 0.3252$

190			3	$\frac{11750995}{4674681342} \approx 0.0025$
190			21	$\frac{21996671405}{32722769394} \approx 0.6722$
191	$\frac{1}{23} \approx 0.04348$	$\frac{1}{22} \approx 0.0454$	6	$\frac{72660709}{254170744} \approx 0.2858$
191			22	$\frac{181510035}{254170744} \approx 0.7141$
192	$\frac{1}{24} \approx 0.04167$	$\frac{1}{23} \approx 0.0434$	6	$\frac{17990899}{63629269} \approx 0.2827$
192			23	$\frac{45638370}{63629269} \approx 0.7172$
193	$\frac{1}{25} = 0.04$	$\frac{1}{24} \approx 0.0416$	6	$\frac{6868668}{21928717} \approx 0.3132$
193			24	$\frac{15060049}{21928717} \approx 0.6867$
194	$\frac{1}{26} \approx 0.03846$	$\frac{1}{25} = 0.04$	7	$\frac{6623739}{23559574} \approx 0.2811$
194			25	$\frac{16935835}{23559574} \approx 0.7188$
195	$\frac{1}{27} \approx 0.03704$	$\frac{1}{26} \approx 0.0384$	7	$\frac{20598370}{73772911} \approx 0.2792$
195			26	$\frac{53174541}{73772911} \approx 0.7207$
196	$\frac{1}{28} \approx 0.03571$	$\frac{1}{27} \approx 0.037$	7	$\frac{20611449}{73987996} \approx 0.2785$
196			27	$\frac{53376547}{73987996} \approx 0.72142$
197	$\frac{1}{29} \approx 0.03448$	$\frac{1}{28} \approx 0.0357$	7	$\frac{5843252}{21159655} \approx 0.2761$
197			28	$\frac{15316403}{21159655} \approx 0.7238$
198	$\frac{1}{30} \approx 0.03333$	$\frac{1}{29} \approx 0.0344$	8	$\frac{97422165}{338982541} \approx 0.2873$
198			29	$\frac{241560376}{338982541} \approx 0.7126$
199	$\frac{1}{31} \approx 0.03226$	$\frac{1}{30} \approx 0.0333$	8	$\frac{246577815}{717694643} \approx 0.3435$
199			30	$\frac{471116828}{717694643} \approx 0.6564$
200	$\frac{1}{32} = 0.03125$	$\frac{1}{31} \approx 0.03226$	8	$\frac{136787965}{369923301} \approx 0.3697$
200			31	$\frac{233135336}{369923301} \approx 0.6302$
201	$\frac{1}{33} \approx 0.0303$	$\frac{1}{32} = 0.03125$	9	$\frac{193885600}{743335051} \approx 0.2608$
201			32	$\frac{549449451}{743335051} \approx 0.7391$
202	$\frac{1}{34} \approx 0.02941$	$\frac{1}{33} \approx 0.0303$	9	$\frac{2009051}{7752584} \approx 0.2591$
202			33	$\frac{5743533}{7752584} \approx 0.7408$
203	$\frac{1}{35} \approx 0.02857$	$\frac{1}{34} \approx 0.02941$	9	$\frac{63841426}{248268817} \approx 0.2571$
203			34	$\frac{184427391}{248268817} \approx 0.7428$
204	$\frac{1}{36} \approx 0.02778$	$\frac{1}{35} \approx 0.02857$	9	$\frac{389848025}{1497560942} \approx 0.2603$
204			35	$\frac{1107712917}{1497560942} \approx 0.7396$
205	$\frac{1}{37} \approx 0.02703$	$\frac{1}{36} \approx 0.02778$	10	$\frac{99052686}{407082371} \approx 0.2433$
205			36	$\frac{308029685}{407082371} \approx 0.7566$
206	$\frac{1}{38} \approx 0.02632$	$\frac{1}{37} \approx 0.02703$	10	$\frac{407411107}{1639477277} \approx 0.2485$
206			37	$\frac{1232066170}{1639477277} \approx 0.7514$
207	$\frac{1}{39} \approx 0.02564$	$\frac{1}{38} \approx 0.02632$	10	$\frac{44138539}{166740862} \approx 0.2647$
207			38	$\frac{122602323}{166740862} \approx 0.7352$

208	$\frac{1}{40} = 0.025$	$\frac{1}{39} \approx 0.02564$	11	$\frac{73429915}{298784748} \approx 0.2457$
208			39	$\frac{225354833}{298784748} \approx 0.7542$
209	$\frac{1}{41} \approx 0.02439$	$\frac{1}{40} = 0.025$	11	$\frac{479473800}{1824013513} \approx 0.2628$
209			40	$\frac{1344539713}{1824013513} \approx 0.7371$
210	$\frac{1}{42} \approx 0.02381$	$\frac{1}{41} \approx 0.02439$	11	$\frac{478583529}{1826578078} \approx 0.2620$
210			41	$\frac{1347994549}{1826578078} \approx 0.7379$
211	$\frac{1}{43} \approx 0.02326$	$\frac{1}{42} \approx 0.02381$	11	$\frac{119627466}{457395215} \approx 0.2615$
211			42	$\frac{337767749}{457395215} \approx 0.7384$
212	0	$\frac{1}{43} \approx 0.02326$	$A_{1,212} = \frac{17}{60}$	$\frac{13701867480}{32568497273} \approx 0.4207$
212			$A_{2,212} = 1$	$\frac{18866629793}{32568497273} \approx 0.5792$

Appendix B: The results of the analysis of AH

Small classes

Threshold Class	Scenario Interval	w	UB
$\left(\frac{1}{6}, \frac{15}{88}\right]$	$\left(\frac{1}{6}, \frac{15}{88}\right]$	$\frac{40165}{4194304} \approx 0.0095$	$\frac{134279683919467}{85106790236160} \approx 1.5777$
$\left(\frac{15}{88}, \frac{1}{5}\right]$	$\left(\frac{15}{88}, \frac{23}{120}\right]$	$\frac{40165}{4194304} \approx 0.0095$	$\frac{134279683919467}{85106790236160} \approx 1.5777$
$\left(\frac{15}{88}, \frac{1}{5}\right]$	$\left(\frac{23}{120}, \frac{1}{5}\right]$	$\frac{40165}{4194304} \approx 0.0095$	$\frac{134279683919467}{85106790236160} \approx 1.5777$
$\left(\frac{1}{5}, \frac{97}{480}\right]$	$\left(\frac{1}{5}, \frac{97}{480}\right]$	$\frac{2754177}{536870912} \approx 0.0051$	$\frac{134279683919467}{85106790236160} \approx 1.5777$
$\left(\frac{97}{480}, \frac{1}{4}\right]$	$\left(\frac{97}{480}, \frac{3}{14}\right]$	$\frac{2754177}{536870912} \approx 0.0051$	$\frac{134279683919467}{85106790236160} \approx 1.5777$
$\left(\frac{97}{480}, \frac{1}{4}\right]$	$\left(\frac{3}{14}, \frac{2}{9}\right]$	$\frac{2754177}{536870912} \approx 0.0051$	$\frac{134279683919467}{85106790236160} \approx 1.5777$
$\left(\frac{97}{480}, \frac{1}{4}\right]$	$\left(\frac{2}{9}, \frac{3}{13}\right]$	$\frac{9224745}{1073741824} \approx 0.0085$	$\frac{176162272658562716766643}{111689991334728680079360} \approx 1.5772$
$\left(\frac{97}{480}, \frac{1}{4}\right]$	$\left(\frac{3}{13}, \frac{4}{17}\right]$	$\frac{9224745}{1073741824} \approx 0.0085$	$\frac{176162272658562716766643}{111689991334728680079360} \approx 1.5772$
$\left(\frac{97}{480}, \frac{1}{4}\right]$	$\left(\frac{4}{17}, \frac{5}{21}\right]$	$\frac{9224745}{1073741824} \approx 0.0085$	$\frac{176162272658562716766643}{111689991334728680079360} \approx 1.5772$
$\left(\frac{97}{480}, \frac{1}{4}\right]$	$\left(\frac{5}{21}, \frac{1}{4}\right]$	$\frac{9224745}{1073741824} \approx 0.0085$	$\frac{176162272658562716766643}{111689991334728680079360} \approx 1.5772$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{1}{4}, \frac{9}{35}\right]$	$\frac{2179203}{16777216} \approx 0.1298$	$\frac{10460110890923925809177}{6663323346674154209280} \approx 1.5698$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{9}{35}, \frac{8}{31}\right]$	$\frac{2179203}{16777216} \approx 0.1298$	$\frac{10460110890923925809177}{6663323346674154209280} \approx 1.5698$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{8}{31}, \frac{7}{27}\right]$	$\frac{2179203}{16777216} \approx 0.1298$	$\frac{10460110890923925809177}{6663323346674154209280} \approx 1.5698$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{7}{27}, \frac{6}{23}\right]$	$\frac{2179203}{16777216} \approx 0.1298$	$\frac{10460110890923925809177}{6663323346674154209280} \approx 1.5698$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{6}{23}, \frac{11}{42}\right]$	$\frac{2179203}{16777216} \approx 0.1298$	$\frac{10460110890923925809177}{6663323346674154209280} \approx 1.5698$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{11}{42}, \frac{5}{19}\right]$	$\frac{2179203}{16777216} \approx 0.1298$	$\frac{10460110890923925809177}{6663323346674154209280} \approx 1.5698$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{5}{19}, \frac{9}{34}\right]$	$\frac{2179203}{16777216} \approx 0.1298$	$\frac{10460110890923925809177}{6663323346674154209280} \approx 1.5698$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{9}{34}, \frac{22}{83}\right]$	$\frac{2179203}{16777216} \approx 0.1298$	$\frac{10460110890923925809177}{6663323346674154209280} \approx 1.5698$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{22}{83}, \frac{4}{15}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{4}{15}, \frac{11}{41}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{11}{41}, \frac{7}{26}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{7}{26}, \frac{10}{37}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{10}{37}, \frac{3}{11}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{3}{11}, \frac{11}{40}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{11}{40}, \frac{8}{29}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{8}{29}, \frac{5}{18}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{5}{18}, \frac{7}{25}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{7}{25}, \frac{9}{32}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{9}{32}, \frac{11}{39}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{1}{4}, \frac{271}{960}\right]$	$\left(\frac{11}{39}, \frac{271}{960}\right]$	$\frac{2750781}{16777216} \approx 0.1639$	$\frac{104464318175074886843 27}{6663323346674154209280} \approx 1.5677$
$\left(\frac{271}{960}, \frac{1}{3}\right]$	$\left(\frac{271}{960}, \frac{17}{60}\right]$	$\frac{10600561}{134217728} \approx 0.0789$	$\frac{138282609904578664033 7}{888443112889887227904} \approx 1.5564$
$\left(\frac{271}{960}, \frac{1}{3}\right]$	$\left(\frac{17}{60}, \frac{2}{7}\right]$	$\frac{13203731}{16777216} \approx 0.7870$	$\frac{906785414291053674997}{576099831014536249344} \approx 1.574$
$\left(\frac{271}{960}, \frac{1}{3}\right]$	$\left(\frac{2}{7}, \frac{24}{83}\right]$	$\frac{13203731}{16777216} \approx 0.7870$	$\frac{906785414291053674997}{576099831014536249344} \approx 1.574$

$\left(\frac{271}{960}, \frac{1}{3}\right]$	$\left(\frac{24}{83}, \frac{3}{10}\right]$	$\frac{6614407}{8388608} \approx 0.7884$	$\frac{25170769021147998936 09}{15961207439363774873 60} \approx 1.5769$

Large classes

Scenario Interval	u	v	w	RBB
$\left(\frac{1}{3}, \frac{1067}{3200}\right]$	$\frac{4849071}{8388608}$	$\frac{1228277}{2097152}$	$\frac{13637073}{16777216}$	$\frac{976466097504059936537}{6187371679054571765 76} \approx 1.5781$
$\left(\frac{1067}{3200}, \frac{1601}{4800}\right]$	$\frac{4848391}{8388608}$	$\frac{9827191}{16777216}$	$\frac{13636371}{16777216}$	$\frac{650943964532874532311 17}{412491445270304784384 00} \approx 1.57807$
$\left(\frac{1601}{4800}, \frac{3203}{9600}\right]$	$\frac{4847711}{8388608}$	$\frac{4914083}{8388608}$	$\frac{3408917}{4194304}$	$\frac{221901316916127842579 9}{140622083614876631040 0} \approx 1.5779$
$\left(\frac{3203}{9600}, \frac{267}{800}\right]$	$\frac{4847031}{8388608}$	$\frac{2457285}{4194304}$	$\frac{13634965}{16777216}$	$\frac{195263112314549308140883}{123747433581091435315200} \approx 1.5779$
$\left(\frac{267}{800}, \frac{641}{1920}\right]$	$\frac{9692701}{16777216}$	$\frac{9830115}{16777216}$	$\frac{6817131}{8388608}$	$\frac{813554496452583589891 1}{515614306587880980480 0} \approx 1.5778$
$\left(\frac{641}{1920}, \frac{1603}{4800}\right]$	$\frac{9691341}{16777216}$	$\frac{9831091}{16777216}$	$\frac{13633559}{16777216}$	$\frac{354987380706190475944 1}{224995333783802609664 0} \approx 1.5777$
$\left(\frac{1603}{4800}, \frac{1069}{3200}\right]$	$\frac{9689981}{16777216}$	$\frac{4916033}{8388608}$	$\frac{1704107}{2097152}$	$\frac{174315199580550113266 3}{110488779983117352960 0} \approx 1.5776$
$\left(\frac{1069}{3200}, \frac{401}{1200}\right]$	$\frac{2422155}{4194304}$	$\frac{9833041}{16777216}$	$\frac{13632153}{16777216}$	$\frac{216914429250997553083 63}{137497148423434928128 00} \approx 1.5775$
$\left(\frac{401}{1200}, \frac{3209}{9600}\right]$	$\frac{9687259}{16777216}$	$\frac{307313}{524288}$	$\frac{13631449}{16777216}$	$\frac{938523792217307183471}{594939584524478054400} \approx 1.5775$
$\left(\frac{3209}{9600}, \frac{107}{320}\right]$	$\frac{9685899}{16777216}$	$\frac{9834991}{16777216}$	$\frac{6815373}{8388608}$	$\frac{976014544418323793724 17}{618737167905457176576 00} \approx 1.5774$
$\left(\frac{107}{320}, \frac{3211}{9600}\right]$	$\frac{4842269}{8388608}$	$\frac{9835967}{16777216}$	$\frac{13630043}{16777216}$	$\frac{106662777263302497317}{676215484049679974 40} \approx 1.5773$
$\left(\frac{3211}{9600}, \frac{803}{2400}\right]$	$\frac{9683177}{16777216}$	$\frac{4918471}{8388608}$	$\frac{3407335}{4194304}$	$\frac{487957099505122615822 49}{309368583952728588288 00} \approx 1.5772$
$\left(\frac{803}{2400}, \frac{1071}{3200}\right]$	$\frac{1210227}{2097152}$	$\frac{9837917}{16777216}$	$\frac{13628637}{16777216}$	$\frac{799888507648052575393}{507161613037259980800} \approx 1.5771$
$\left(\frac{1071}{3200}, \frac{1607}{4800}\right]$	$\frac{9680455}{16777216}$	$\frac{9838893}{16777216}$	$\frac{13627933}{16777216}$	$\frac{844860460332803406563}{535703175675720499200} \approx 1.5771$
$\left(\frac{1607}{4800}, \frac{643}{1920}\right]$	$\frac{4839547}{8388608}$	$\frac{2459967}{4194304}$	$\frac{6813615}{8388608}$	$\frac{487881799269341406622 01}{309368583952728588288 00} \approx 1.577$
$\left(\frac{643}{1920}, \frac{67}{200}\right]$	$\frac{9677733}{16777216}$	$\frac{2460211}{4194304}$	$\frac{13626527}{16777216}$	$\frac{390285372968618970287 93}{247494867162182870630 40} \approx 1.5769$
$\left(\frac{67}{200}, \frac{3217}{9600}\right]$	$\frac{2419093}{4194304}$	$\frac{9841819}{16777216}$	$\frac{13625823}{16777216}$	$\frac{271017553001388678129 3}{171871435529293660160 0} \approx 1.5768$
$\left(\frac{3217}{9600}, \frac{1609}{4800}\right]$	$\frac{9675011}{16777216}$	$\frac{9842795}{16777216}$	$\frac{425785}{524288}$	$\frac{609758132013612388794 1}{386710729940910735360 0} \approx 1.5767$
$\left(\frac{1609}{4800}, \frac{1073}{3200}\right]$	$\frac{9673649}{16777216}$	$\frac{9843771}{16777216}$	$\frac{425763}{524288}$	$\frac{234510295646344141999}{148734896131119513600} \approx 1.5766$
$\left(\frac{1073}{3200}, \frac{161}{480}\right]$	$\frac{302259}{524288}$	$\frac{9844747}{16777216}$	$\frac{13623713}{16777216}$	$\frac{650341757892992119900 13}{412491445270304784384 00} \approx 1.5766$
$\left(\frac{161}{480}, \frac{3221}{9600}\right]$	$\frac{4835463}{8388608}$	$\frac{9845723}{16777216}$	$\frac{13623009}{16777216}$	$\frac{139351777432977396703 3}{883910239864938823680} \approx 1.5765$
$\left(\frac{3221}{9600}, \frac{537}{1600}\right]$	$\frac{9669565}{16777216}$	$\frac{4923349}{8388608}$	$\frac{6811153}{8388608}$	$\frac{886738328869305536104 3}{562488334459506524160 0} \approx 1.5764$
$\left(\frac{537}{1600}, \frac{3223}{9600}\right]$	$\frac{9668203}{16777216}$	$\frac{4923837}{8388608}$	$\frac{6810801}{8388608}$	$\frac{147782114162413859881 1}{937480557432510873600} \approx 1.5763$
$\left(\frac{3223}{9600}, \frac{403}{1200}\right]$	$\frac{9666841}{16777216}$	$\frac{4924325}{8388608}$	$\frac{6810449}{8388608}$	$\frac{886647034952185645412 9}{562488334459506524160 0} \approx 1.5762$
$\left(\frac{403}{1200}, \frac{43}{128}\right]$	$\frac{1208185}{2097152}$	$\frac{4924813}{8388608}$	$\frac{6810097}{8388608}$	$\frac{346328663594692575 97}{219722005648244736 00} \approx 1.5762$
$\left(\frac{43}{128}, \frac{1613}{4800}\right]$	$\frac{4832059}{8388608}$	$\frac{4925301}{8388608}$	$\frac{13619491}{16777216}$	$\frac{866854473982499251647}{549988593693739712512} \approx 1.5761$
$\left(\frac{1613}{4800}, \frac{3227}{9600}\right]$	$\frac{2415689}{4194304}$	$\frac{9851579}{16777216}$	$\frac{13618787}{16777216}$	$\frac{750123939807016719369 1}{475951667619582443520 0} \approx 1.576$
$\left(\frac{3227}{9600}, \frac{269}{800}\right]$	$\frac{4830697}{8388608}$	$\frac{9852555}{16777216}$	$\frac{13618083}{16777216}$	$\frac{278603108505283130520 91}{176782047972987764736 00} \approx 1.5759$
$\left(\frac{269}{800}, \frac{3229}{9600}\right]$	$\frac{75469}{131072}$	$\frac{9853531}{16777216}$	$\frac{13617379}{16777216}$	$\frac{162510105174606254617 49}{103122861317576196096 00} \approx 1.5758$
$\left(\frac{3229}{9600}, \frac{323}{960}\right]$	$\frac{4829335}{8388608}$	$\frac{9854507}{16777216}$	$\frac{13616675}{16777216}$	$\frac{195002075117312289565 019}{123747433581091435315200} \approx 1.5758$
$\left(\frac{323}{960}, \frac{1077}{3200}\right]$	$\frac{9657307}{16777216}$	$\frac{2463871}{4194304}$	$\frac{13615971}{16777216}$	$\frac{194992037428896184371 13}{123747433581091435315 20} \approx 1.5757$
$\left(\frac{1077}{3200}, \frac{101}{300}\right]$	$\frac{9654005}{16777216}$	$\frac{4928925}{8388608}$	$\frac{13611571}{16777216}$	$\frac{745777983035567922074 9}{473350838834775982080 0} \approx 1.5755$
$\left(\frac{101}{300}, \frac{3233}{9600}\right]$	$\frac{2413161}{4194304}$	$\frac{4929413}{8388608}$	$\frac{13610869}{16777216}$	$\frac{310159872932208393188 41}{196870917060827283456 00} \approx 1.5754$
$\left(\frac{3233}{9600}, \frac{539}{1600}\right]$	$\frac{4825641}{8388608}$	$\frac{4929901}{8388608}$	$\frac{13610167}{16777216}$	$\frac{136463316343884055531 9321}{866232035067640047206 400} \approx 1.5753$
$\left(\frac{539}{1600}, \frac{647}{1920}\right]$	$\frac{9649921}{16777216}$	$\frac{4930389}{8388608}$	$\frac{13609465}{16777216}$	$\frac{227427146094410028878003}{144372005844606674534400} \approx 1.5752$

$\left(\frac{647}{1920}, \frac{809}{2400}\right]$	$\frac{9648559}{16777216}$	$\frac{4930877}{8388608}$	$\frac{13608763}{16777216}$	$\frac{272898516056915156775019}{173246407013528009441280}$	≈ 1.5752
$\left(\frac{809}{2400}, \frac{1079}{3200}\right]$	$\frac{9647197}{16777216}$	$\frac{4931365}{8388608}$	$\frac{13608061}{16777216}$	$\frac{155047985749225839447 79}{98435458530413641728 00}$	≈ 1.5751
$\left(\frac{1079}{3200}, \frac{1619}{4800}\right]$	$\frac{2411459}{4194304}$	$\frac{4931853}{8388608}$	$\frac{13607359}{16777216}$	$\frac{151594662165810553673821}{962480038964044496896 00}$	≈ 1.575
$\left(\frac{1619}{4800}, \frac{3239}{9600}\right]$	$\frac{4822237}{8388608}$	$\frac{4932341}{8388608}$	$\frac{53151}{65536}$	$\frac{2664612582117044275993}{1691859443491484467200}$	≈ 1.5749
$\left(\frac{3239}{9600}, \frac{27}{80}\right]$	$\frac{1205389}{2097152}$	$\frac{4932829}{8388608}$	$\frac{6802977}{8388608}$	$\frac{68210565405866216230892 9}{43311601753382002360320 0}$	≈ 1.5748
$\left(\frac{27}{80}, \frac{3241}{9600}\right]$	$\frac{4820875}{8388608}$	$\frac{4933317}{8388608}$	$\frac{3401313}{4194304}$	$\frac{56839206865884082191 07}{36093001461151668633 60}$	≈ 1.5747
$\left(\frac{3241}{9600}, \frac{1621}{4800}\right]$	$\frac{2410097}{4194304}$	$\frac{9867611}{16777216}$	$\frac{6802275}{8388608}$	$\frac{68203535764925654987634 1}{43311601753382002360320 0}$	≈ 1.5747
$\left(\frac{1621}{4800}, \frac{1081}{3200}\right]$	$\frac{4819513}{8388608}$	$\frac{9868587}{16777216}$	$\frac{13603847}{16777216}$	$\frac{68200018008930956348722 7}{43311601753382002360320 0}$	≈ 1.5746
$\left(\frac{1081}{3200}, \frac{811}{2400}\right]$	$\frac{9637663}{16777216}$	$\frac{9869563}{16777216}$	$\frac{13603145}{16777216}$	$\frac{41331211775978752332917}{26249455608110304460800}$	≈ 1.5745
$\left(\frac{811}{2400}, \frac{649}{1920}\right]$	$\frac{9636301}{16777216}$	$\frac{2467635}{4194304}$	$\frac{6801221}{8388608}$	$\frac{332973563943163219793}{211482430436435558400}$	≈ 1.5744
$\left(\frac{649}{1920}, \frac{541}{1600}\right]$	$\frac{9634939}{16777216}$	$\frac{2467879}{4194304}$	$\frac{3400435}{4194304}$	$\frac{68189466375708315383477}{43311601753382002360320}$	≈ 1.5743
$\left(\frac{541}{1600}, \frac{3247}{9600}\right]$	$\frac{1204197}{2097152}$	$\frac{9872493}{16777216}$	$\frac{6800519}{8388608}$	$\frac{37881084193626042951837}{24062000974101112422400}$	≈ 1.5743
$\left(\frac{3247}{9600}, \frac{203}{600}\right]$	$\frac{4816107}{8388608}$	$\frac{9873469}{16777216}$	$\frac{13600335}{16777216}$	$\frac{13636486287562100552617 19}{86623203506764004720640 0}$	≈ 1.5742
$\left(\frac{203}{600}, \frac{1083}{3200}\right]$	$\frac{9630851}{16777216}$	$\frac{4937223}{8388608}$	$\frac{849977}{1048576}$	$\frac{21305911255966877993659}{13534875547931875737600}$	≈ 1.5741
$\left(\frac{1083}{3200}, \frac{65}{192}\right]$	$\frac{601843}{1048576}$	$\frac{9875423}{16777216}$	$\frac{6799465}{8388608}$	$\frac{22725133259938925938393 33}{14437200584460667453440 0}$	≈ 1.574
$\left(\frac{65}{192}, \frac{3251}{9600}\right]$	$\frac{4814063}{8388608}$	$\frac{9876399}{16777216}$	$\frac{13598227}{16777216}$	$\frac{24789773911272480247 61}{15749673364866182676 48}$	≈ 1.5739
$\left(\frac{3251}{9600}, \frac{271}{800}\right]$	$\frac{9626763}{16777216}$	$\frac{77167}{131072}$	$\frac{3399381}{4194304}$	$\frac{34084180712090872456759 3}{21655800876691001180160 0}$	≈ 1.5739
$\left(\frac{271}{800}, \frac{3253}{9600}\right]$	$\frac{1203175}{2097152}$	$\frac{9878353}{16777216}$	$\frac{6798411}{8388608}$	$\frac{28402018238670299772763}{18046500730575834316800}$	≈ 1.5738
$\left(\frac{3253}{9600}, \frac{1627}{4800}\right]$	$\frac{9624037}{16777216}$	$\frac{4939665}{8388608}$	$\frac{13596119}{16777216}$	$\frac{13632265199807090812677 41}{86623203506764004720640 0}$	≈ 1.5737
$\left(\frac{1627}{4800}, \frac{217}{640}\right]$	$\frac{4811337}{8388608}$	$\frac{9880307}{16777216}$	$\frac{1699427}{2097152}$	$\frac{77452054266041813309 03}{49217729265206820864 00}$	≈ 1.5736
$\left(\frac{217}{640}, \frac{407}{1200}\right]$	$\frac{9621311}{16777216}$	$\frac{2470321}{4194304}$	$\frac{13594713}{16777216}$	$\frac{30290795128023441955693}{19249600779280889937920}$	≈ 1.5735
$\left(\frac{407}{1200}, \frac{3257}{9600}\right]$	$\frac{2404987}{4194304}$	$\frac{9882261}{16777216}$	$\frac{6797005}{8388608}$	$\frac{17037692462710825032118 9}{10827900438345500590080 0}$	≈ 1.5734
$\left(\frac{3257}{9600}, \frac{543}{1600}\right]$	$\frac{1202323}{2097152}$	$\frac{4941619}{8388608}$	$\frac{3398327}{4194304}$	$\frac{34073624922876943826 1761}{21655800876691001180160 0}$	≈ 1.5734
$\left(\frac{543}{1600}, \frac{3259}{9600}\right]$	$\frac{2404215}{4194304}$	$\frac{4942237}{8388608}$	$\frac{6795959}{8388608}$	$\frac{14284532431345121417 99}{90792568487789199360 0}$	≈ 1.5733
$\left(\frac{3259}{9600}, \frac{163}{480}\right]$	$\frac{9615497}{16777216}$	$\frac{9885451}{16777216}$	$\frac{13591215}{16777216}$	$\frac{63896172965744945035396 3}{40614542303537701847040 0}$	≈ 1.5732
$\left(\frac{163}{480}, \frac{1087}{3200}\right]$	$\frac{9614133}{16777216}$	$\frac{2471607}{4194304}$	$\frac{849407}{1048576}$	$\frac{23959826404898791776701}{15230453363826638192640}$	≈ 1.5731
$\left(\frac{1087}{3200}, \frac{1631}{4800}\right]$	$\frac{9612769}{16777216}$	$\frac{4943703}{8388608}$	$\frac{13589809}{16777216}$	$\frac{19166871678061802705632 31}{12184362691061310554112 00}$	≈ 1.573
$\left(\frac{1631}{4800}, \frac{3263}{9600}\right]$	$\frac{4805703}{8388608}$	$\frac{9888383}{16777216}$	$\frac{13589105}{16777216}$	$\frac{91266098781049354432547}{58020774719339574067200}$	≈ 1.5729
$\left(\frac{3263}{9600}, \frac{17}{50}\right]$	$\frac{4805021}{8388608}$	$\frac{9889361}{16777216}$	$\frac{6794201}{8388608}$	$\frac{95824455164924317764875 9}{60921813455306552770560 0}$	≈ 1.5729
$\left(\frac{17}{50}, \frac{653}{1920}\right]$	$\frac{4804339}{8388608}$	$\frac{4945169}{8388608}$	$\frac{13587699}{16777216}$	$\frac{19163899736918141987717 09}{12184362691061310554112 00}$	≈ 1.5728
$\left(\frac{653}{1920}, \frac{1633}{4800}\right]$	$\frac{4803657}{8388608}$	$\frac{2472829}{4194304}$	$\frac{13586995}{16777216}$	$\frac{42584244127472991954513}{27076361535691801231360}$	≈ 1.5727
$\left(\frac{1633}{4800}, \frac{1089}{3200}\right]$	$\frac{4802975}{8388608}$	$\frac{9892293}{16777216}$	$\frac{3396573}{4194304}$	$\frac{43549814309358760573049}{27691733388775705804800}$	≈ 1.5726
$\left(\frac{1089}{3200}, \frac{817}{2400}\right]$	$\frac{4802293}{8388608}$	$\frac{9893271}{16777216}$	$\frac{3396397}{4194304}$	$\frac{43547563979990343602953}{27691733388775705804800}$	≈ 1.5725
$\left(\frac{817}{2400}, \frac{3269}{9600}\right]$	$\frac{4801611}{8388608}$	$\frac{9894249}{16777216}$	$\frac{13584885}{16777216}$	$\frac{63866459257539031087476 1}{40614542303537701847040 0}$	≈ 1.5725
$\left(\frac{3269}{9600}, \frac{109}{320}\right]$	$\frac{4800929}{8388608}$	$\frac{9895227}{16777216}$	$\frac{13584181}{16777216}$	$\frac{24881749827237400000967}{15823847650728974745600}$	≈ 1.5724
$\left(\frac{109}{320}, \frac{3271}{9600}\right]$	$\frac{9600493}{16777216}$	$\frac{9896205}{16777216}$	$\frac{6791739}{8388608}$	$\frac{19157956727860485979556 9}{12184362691061310554112 0}$	≈ 1.5723
$\left(\frac{3271}{9600}, \frac{409}{1200}\right]$	$\frac{9599129}{16777216}$	$\frac{4948591}{8388608}$	$\frac{6791387}{8388608}$	$\frac{31928274332859244608231 1}{20307271151768850923520 0}$	≈ 1.5722
$\left(\frac{409}{1200}, \frac{1091}{3200}\right]$	$\frac{2399441}{4194304}$	$\frac{618635}{1048576}$	$\frac{6791035}{8388608}$	$\frac{95779868956646461366261}{60921813455306552770560 0}$	≈ 1.5721
$\left(\frac{1091}{3200}, \frac{1637}{4800}\right]$	$\frac{599775}{1048576}$	$\frac{4949569}{8388608}$	$\frac{13581367}{16777216}$	$\frac{19154982754402437538809 53}{12184362691061310554112 00}$	≈ 1.572
$\left(\frac{1637}{4800}, \frac{131}{384}\right]$	$\frac{9595035}{16777216}$	$\frac{9900117}{16777216}$	$\frac{13580663}{16777216}$	$\frac{21282214592376213525560 1}{13538180767845900615680 0}$	≈ 1.572
$\left(\frac{131}{384}, \frac{273}{800}\right]$	$\frac{9593671}{16777216}$	$\frac{9901095}{16777216}$	$\frac{13579959}{16777216}$	$\frac{38306003853309976846550 9}{24368725382122621108224 0}$	≈ 1.5719
$\left(\frac{273}{800}, \frac{3277}{9600}\right]$	$\frac{4796153}{8388608}$	$\frac{9902073}{16777216}$	$\frac{1697407}{2097152}$	$\frac{34200018735559994763271}{21757790519752340275200}$	≈ 1.5718
$\left(\frac{3277}{9600}, \frac{1639}{4800}\right]$	$\frac{9590941}{16777216}$	$\frac{9903051}{16777216}$	$\frac{1697319}{2097152}$	$\frac{79795912583596606079363}{50768177879422127308800}$	≈ 1.5717

$\left(\frac{1639}{4800}, \frac{1093}{3200}\right]$	$\frac{1198697}{2097152}$	$\frac{9904029}{16777216}$	$\frac{1697231}{2097152}$	$\frac{23937534269352794371504 1}{15230453363826638192640 0}$	≈ 1.5716
$\left(\frac{1093}{3200}, \frac{41}{120}\right]$	$\frac{9588211}{16777216}$	$\frac{619063}{1048576}$	$\frac{1697143}{2097152}$	$\frac{23936296413324959015455 3}{15230453363826638192640 0}$	≈ 1.5716
$\left(\frac{41}{120}, \frac{3281}{9600}\right]$	$\frac{9574953}{16777216}$	$\frac{4957255}{8388608}$	$\frac{13553857}{16777216}$	$\frac{14952663338427772269 77}{95190333523916488704 0}$	≈ 1.5708
$\left(\frac{3281}{9600}, \frac{547}{1600}\right]$	$\frac{9573585}{16777216}$	$\frac{9915491}{16777216}$	$\frac{3388287}{4194304}$	$\frac{37379717847575632627 33}{23797583380979122176 00}$	≈ 1.5707
$\left(\frac{547}{1600}, \frac{3283}{9600}\right]$	$\frac{1196527}{2097152}$	$\frac{1239559}{2097152}$	$\frac{1694055}{2097152}$	$\frac{1557407371424976720 59}{99156597420746342400}$	≈ 1.5706
$\left(\frac{3283}{9600}, \frac{821}{2400}\right]$	$\frac{299089}{524288}$	$\frac{9917453}{16777216}$	$\frac{13551731}{16777216}$	$\frac{14950334357491487905151}{95190333523916488704 00}$	≈ 1.5705
$\left(\frac{821}{2400}, \frac{219}{640}\right]$	$\frac{9569479}{16777216}$	$\frac{4959217}{8388608}$	$\frac{6775511}{8388608}$	$\frac{37373894612177747067 37}{23797583380979122176 00}$	≈ 1.5704
$\left(\frac{219}{640}, \frac{1643}{4800}\right]$	$\frac{4784055}{8388608}$	$\frac{9919415}{16777216}$	$\frac{13550313}{16777216}$	$\frac{33219513839596189517 9}{21153407449759219712 0}$	≈ 1.5704
$\left(\frac{1643}{4800}, \frac{3287}{9600}\right]$	$\frac{9566741}{16777216}$	$\frac{2480099}{4194304}$	$\frac{3387401}{4194304}$	$\frac{83415203718378187 91}{53119605761114112 00}$	≈ 1.5703
$\left(\frac{3287}{9600}, \frac{137}{400}\right]$	$\frac{2391343}{4194304}$	$\frac{9921377}{16777216}$	$\frac{13548895}{16777216}$	$\frac{14947227680416248798199}{95190333523916488704 00}$	≈ 1.5702
$\left(\frac{137}{400}, \frac{3289}{9600}\right]$	$\frac{9564003}{16777216}$	$\frac{4961179}{8388608}$	$\frac{6774093}{8388608}$	$\frac{12455375625055764881 03}{79325277936597073920 0}$	≈ 1.5701
$\left(\frac{3289}{9600}, \frac{329}{960}\right]$	$\frac{4781317}{8388608}$	$\frac{9923339}{16777216}$	$\frac{13547477}{16777216}$	$\frac{13586976104804889500 53}{86536666839924080640 0}$	≈ 1.57
$\left(\frac{329}{960}, \frac{1097}{3200}\right]$	$\frac{9561265}{16777216}$	$\frac{9924321}{16777216}$	$\frac{846673}{1048576}$	$\frac{46702805346340105397}{29746979226223902720}$	≈ 1.57
$\left(\frac{1097}{3200}, \frac{823}{2400}\right]$	$\frac{1194987}{2097152}$	$\frac{4962651}{8388608}$	$\frac{13546059}{16777216}$	$\frac{49813734890606959129669}{31730111174638829568 00}$	≈ 1.5699
$\left(\frac{823}{2400}, \frac{3293}{9600}\right]$	$\frac{9558527}{16777216}$	$\frac{2481571}{4194304}$	$\frac{6772675}{8388608}$	$\frac{33962146031504353941 1}{21634166709981020160 0}$	≈ 1.5698
$\left(\frac{3293}{9600}, \frac{549}{1600}\right]$	$\frac{9557157}{16777216}$	$\frac{9927265}{16777216}$	$\frac{13544641}{16777216}$	$\frac{21346524001930774949 53}{13598619074845212672 00}$	≈ 1.5697
$\left(\frac{549}{1600}, \frac{659}{1920}\right]$	$\frac{2388947}{4194304}$	$\frac{9928247}{16777216}$	$\frac{13543931}{16777216}$	$\frac{18865897030054472569}{12018981505545011200}$	≈ 1.5696
$\left(\frac{659}{1920}, \frac{103}{300}\right]$	$\frac{4777209}{8388608}$	$\frac{2482307}{4194304}$	$\frac{6771611}{8388608}$	$\frac{29882025572709409072 3}{19038066704783297740 8}$	≈ 1.5695
$\left(\frac{103}{300}, \frac{1099}{3200}\right]$	$\frac{9553049}{16777216}$	$\frac{4965105}{8388608}$	$\frac{13542513}{16777216}$	$\frac{74701180776156307403 81}{47595166761958244352 00}$	≈ 1.5695
$\left(\frac{1099}{3200}, \frac{1649}{4800}\right]$	$\frac{9551679}{16777216}$	$\frac{9931191}{16777216}$	$\frac{3385451}{4194304}$	$\frac{12449548570765339009 07}{79325277936597073920 0}$	≈ 1.5694
$\left(\frac{1649}{4800}, \frac{3299}{9600}\right]$	$\frac{4775155}{8388608}$	$\frac{9932173}{16777216}$	$\frac{6770547}{8388608}$	$\frac{18673351891591168836 97}{11898791690489561088 00}$	≈ 1.5693
$\left(\frac{3299}{9600}, \frac{11}{32}\right]$	$\frac{2387235}{4194304}$	$\frac{9933155}{16777216}$	$\frac{13540385}{16777216}$	$\frac{14937904568779219079413}{95190333523916488704 00}$	≈ 1.5692
$\left(\frac{11}{32}, \frac{3301}{9600}\right]$	$\frac{2386613}{4194304}$	$\frac{4967469}{8388608}$	$\frac{13537557}{16777216}$	$\frac{38088059000448860006 65}{24273535048598704619 52}$	≈ 1.5691
$\left(\frac{3301}{9600}, \frac{1651}{4800}\right]$	$\frac{9545083}{16777216}$	$\frac{9935919}{16777216}$	$\frac{13536849}{16777216}$	$\frac{76172155535753485365 2867}{48547070097197409239 0400}$	≈ 1.569
$\left(\frac{1651}{4800}, \frac{1101}{3200}\right]$	$\frac{4771857}{8388608}$	$\frac{2484225}{4194304}$	$\frac{13536141}{16777216}$	$\frac{38084096269378096213 4227}{24273535048598704619 5200}$	≈ 1.5689
$\left(\frac{1101}{3200}, \frac{413}{1200}\right]$	$\frac{9542345}{16777216}$	$\frac{9937881}{16777216}$	$\frac{6767717}{8388608}$	$\frac{54403020465723070833 259}{34676478640855292313 6000}$	≈ 1.5688
$\left(\frac{413}{1200}, \frac{661}{1920}\right]$	$\frac{596311}{1048576}$	$\frac{9938863}{16777216}$	$\frac{6767363}{8388608}$	$\frac{86545761794257077441 41}{55167125110451601408 00}$	≈ 1.5687
$\left(\frac{661}{1920}, \frac{551}{1600}\right]$	$\frac{9539607}{16777216}$	$\frac{2484961}{4194304}$	$\frac{6767009}{8388608}$	$\frac{76156305786765362781011}{48547070097197409239040}$	≈ 1.5687
$\left(\frac{551}{1600}, \frac{3307}{9600}\right]$	$\frac{9538237}{16777216}$	$\frac{4970413}{8388608}$	$\frac{13533311}{16777216}$	$\frac{38076173046613220238 7531}{24273535048598704619 5200}$	≈ 1.5686
$\left(\frac{3307}{9600}, \frac{827}{2400}\right]$	$\frac{2384217}{4194304}$	$\frac{9941807}{16777216}$	$\frac{13532603}{16777216}$	$\frac{76148380437715060971 5159}{48547070097197409239 0400}$	≈ 1.5685
$\left(\frac{827}{2400}, \frac{1103}{3200}\right]$	$\frac{4767749}{8388608}$	$\frac{9942789}{16777216}$	$\frac{13531895}{16777216}$	$\frac{11897565630876931130309}{75854797026870951936 00}$	≈ 1.5684
$\left(\frac{1103}{3200}, \frac{331}{960}\right]$	$\frac{9534129}{16777216}$	$\frac{9943771}{16777216}$	$\frac{13531187}{16777216}$	$\frac{76140459105656599206 0113}{48547070097197409239 0400}$	≈ 1.5683
$\left(\frac{331}{960}, \frac{3311}{9600}\right]$	$\frac{9532759}{16777216}$	$\frac{621547}{1048576}$	$\frac{13530479}{16777216}$	$\frac{10876641693512292118391}{69352957281710584627200}$	≈ 1.5683
$\left(\frac{3311}{9600}, \frac{69}{200}\right]$	$\frac{9531389}{16777216}$	$\frac{4972867}{8388608}$	$\frac{13529771}{16777216}$	$\frac{76132529858924168219 9147}{48547070097197409239 0400}$	≈ 1.5682
$\left(\frac{69}{200}, \frac{3313}{9600}\right]$	$\frac{9530019}{16777216}$	$\frac{2486679}{4194304}$	$\frac{13529063}{16777216}$	$\frac{19032141832852308330 1661}{12136767524299352309 7600}$	≈ 1.5681
$\left(\frac{3313}{9600}, \frac{1657}{4800}\right]$	$\frac{9528649}{16777216}$	$\frac{4973849}{8388608}$	$\frac{13528355}{16777216}$	$\frac{11344948475714044728 91}{72350328013707018240 0}$	≈ 1.568
$\left(\frac{1657}{4800}, \frac{221}{640}\right]$	$\frac{595455}{1048576}$	$\frac{1243585}{2097152}$	$\frac{13527647}{16777216}$	$\frac{38060320340411009435 4399}{24273535048598704619 5200}$	≈ 1.5679
$\left(\frac{221}{640}, \frac{829}{2400}\right]$	$\frac{9525909}{16777216}$	$\frac{4974831}{8388608}$	$\frac{13526939}{16777216}$	$\frac{13839395737772014341881}{88267400176722562252 80}$	≈ 1.5678
$\left(\frac{829}{2400}, \frac{3317}{9600}\right]$	$\frac{9524539}{16777216}$	$\frac{2487661}{4194304}$	$\frac{13526231}{16777216}$	$\frac{95140889878523638524229}{60683837621496761548800}$	≈ 1.5678
$\left(\frac{3317}{9600}, \frac{553}{1600}\right]$	$\frac{9523169}{16777216}$	$\frac{4975813}{8388608}$	$\frac{13525523}{16777216}$	$\frac{10872678102291240638 2847}{69352957281710584627200}$	≈ 1.5677
$\left(\frac{553}{1600}, \frac{3319}{9600}\right]$	$\frac{9521799}{16777216}$	$\frac{311019}{524288}$	$\frac{6762407}{8388608}$	$\frac{19026195337416226305 7889}{12136767524299352309 7600}$	≈ 1.5676
$\left(\frac{3319}{9600}, \frac{83}{240}\right]$	$\frac{9520429}{16777216}$	$\frac{4976795}{8388608}$	$\frac{6762053}{8388608}$	$\frac{38050407549401479924 3069}{24273535048598704619 5200}$	≈ 1.5675
$\left(\frac{83}{240}, \frac{1107}{3200}\right]$	$\frac{4759529}{8388608}$	$\frac{2488643}{4194304}$	$\frac{6761699}{8388608}$	$\frac{19024212079021989045911}{12136767524299352309760}$	≈ 1.5674

$\left(\frac{1107}{3200}, \frac{1661}{4800}\right]$	$\frac{1189711}{2097152}$	$\frac{9955555}{16777216}$	$\frac{13522689}{16777216}$	$\frac{760928871399143283572207}{4854707009719740923904000} \approx 1.5674$
$\left(\frac{1661}{4800}, \frac{3323}{9600}\right]$	$\frac{9516317}{16777216}$	$\frac{9956537}{16777216}$	$\frac{13521981}{16777216}$	$\frac{380444596465588028345497}{242735350485987046195200} \approx 1.5673$
$\left(\frac{3323}{9600}, \frac{277}{800}\right]$	$\frac{4757473}{8388608}$	$\frac{9957519}{16777216}$	$\frac{13521273}{16777216}$	$\frac{760849509144678246905501}{4854707009719740923904000} \approx 1.5672$
$\left(\frac{277}{800}, \frac{133}{384}\right]$	$\frac{1189197}{2097152}$	$\frac{4979251}{8388608}$	$\frac{3380141}{4194304}$	$\frac{1698236342438227951421}{1083639957526727884800} \approx 1.5671$
$\left(\frac{133}{384}, \frac{1663}{4800}\right]$	$\frac{9512205}{16777216}$	$\frac{2489871}{4194304}$	$\frac{844991}{1048576}$	$\frac{1901925467462527805549}{1213676752429935230976} \approx 1.567$
$\left(\frac{1663}{4800}, \frac{1109}{3200}\right]$	$\frac{4755417}{8388608}$	$\frac{9960467}{16777216}$	$\frac{13519147}{16777216}$	$\frac{345786613001968072033331}{220668500441806405632000} \approx 1.5669$
$\left(\frac{1109}{3200}, \frac{26}{75}\right]$	$\frac{9509463}{16777216}$	$\frac{9961449}{16777216}$	$\frac{13518439}{16777216}$	$\frac{760690843536197568224759}{4854707009719740923904000} \approx 1.5669$
$\left(\frac{26}{75}, \frac{3329}{9600}\right]$	$\frac{2377023}{4194304}$	$\frac{155663}{262144}$	$\frac{6758865}{8388608}$	$\frac{950813993133983706751 27}{6068383762149676154880 0} \approx 1.5668$
$\left(\frac{3329}{9600}, \frac{111}{320}\right]$	$\frac{9506721}{16777216}$	$\frac{9963415}{16777216}$	$\frac{13517021}{16777216}$	$\frac{760611540152133730827083}{4854707009719740923904000} \approx 1.5667$
$\left(\frac{111}{320}, \frac{3331}{9600}\right]$	$\frac{4752675}{8388608}$	$\frac{9964397}{16777216}$	$\frac{13516313}{16777216}$	$\frac{760571819120897743019 81}{4854707009719740923904 0} \approx 1.5666$
$\left(\frac{3331}{9600}, \frac{833}{2400}\right]$	$\frac{9503979}{16777216}$	$\frac{2491345}{4194304}$	$\frac{3378901}{4194304}$	$\frac{271618626470541167631 59}{1733823932042764615680 0} \approx 1.5665$
$\left(\frac{833}{2400}, \frac{1111}{3200}\right]$	$\frac{9502607}{16777216}$	$\frac{9966363}{16777216}$	$\frac{13514895}{16777216}$	$\frac{950615604735112704958 63}{6068383762149676154880 0} \approx 1.5665$
$\left(\frac{1111}{3200}, \frac{1667}{4800}\right]$	$\frac{2375309}{4194304}$	$\frac{4983673}{8388608}$	$\frac{6757093}{8388608}$	$\frac{380226404066311213407583}{242735350485987046195200} \approx 1.5664$
$\left(\frac{1667}{4800}, \frac{667}{1920}\right]$	$\frac{9493465}{16777216}$	$\frac{2493229}{4194304}$	$\frac{13501373}{16777216}$	$\frac{141611096872472700200291}{9043081684772066426880 0} \approx 1.5659$
$\left(\frac{667}{1920}, \frac{139}{400}\right]$	$\frac{2373023}{4194304}$	$\frac{2493475}{4194304}$	$\frac{13500661}{16777216}$	$\frac{188804921143682657223507}{1205744224636275523584 0} \approx 1.5658$
$\left(\frac{139}{400}, \frac{3337}{9600}\right]$	$\frac{4745359}{8388608}$	$\frac{2493721}{4194304}$	$\frac{6749975}{8388608}$	$\frac{353990709168636178300 51}{2260770421193016660720 0} \approx 1.5657$
$\left(\frac{3337}{9600}, \frac{1669}{4800}\right]$	$\frac{9489345}{16777216}$	$\frac{9975869}{16777216}$	$\frac{6749619}{8388608}$	$\frac{128717169691715456065 03}{8220983349792787660800} \approx 1.5657$
$\left(\frac{1669}{4800}, \frac{1113}{3200}\right]$	$\frac{9487971}{16777216}$	$\frac{9976853}{16777216}$	$\frac{13498527}{16777216}$	$\frac{257889758215323713103}{164719156371075891200} \approx 1.5656$
$\left(\frac{1113}{3200}, \frac{167}{480}\right]$	$\frac{4743299}{8388608}$	$\frac{4988919}{8388608}$	$\frac{13497815}{16777216}$	$\frac{283148156523474240613187}{1808616336954413285376 00} \approx 1.5655$
$\left(\frac{167}{480}, \frac{3341}{9600}\right]$	$\frac{1185653}{2097152}$	$\frac{9978823}{16777216}$	$\frac{13497103}{16777216}$	$\frac{126398819881212171 47}{8074180075689345024} \approx 1.5654$
$\left(\frac{3341}{9600}, \frac{557}{1600}\right]$	$\frac{4741925}{8388608}$	$\frac{9979807}{16777216}$	$\frac{13496391}{16777216}$	$\frac{857934948458671659390 1}{5480655566528525107200} \approx 1.5653$
$\left(\frac{557}{1600}, \frac{3343}{9600}\right]$	$\frac{2370619}{4194304}$	$\frac{1247599}{2097152}$	$\frac{105435}{131072}$	$\frac{110587393666783334596 1}{7064907566228176896 00} \approx 1.5653$
$\left(\frac{3343}{9600}, \frac{209}{600}\right]$	$\frac{4740551}{8388608}$	$\frac{9981777}{16777216}$	$\frac{1686871}{2097152}$	$\frac{353861152276227903528 23}{2260770421193016606720 0} \approx 1.5652$
$\left(\frac{209}{600}, \frac{223}{640}\right]$	$\frac{592483}{1048576}$	$\frac{4991381}{8388608}$	$\frac{843391}{1048576}$	$\frac{147434434303328921319 1}{9419876754970902528 00} \approx 1.5651$
$\left(\frac{223}{640}, \frac{1673}{4800}\right]$	$\frac{4739177}{8388608}$	$\frac{4991873}{8388608}$	$\frac{1686693}{2097152}$	$\frac{707648205877015237672 7}{4521540842386033213440} \approx 1.565$
$\left(\frac{1673}{4800}, \frac{3347}{9600}\right]$	$\frac{2369245}{4194304}$	$\frac{9984731}{16777216}$	$\frac{421651}{524288}$	$\frac{442256985010937095134 7}{2825963026491270758400} \approx 1.5649$
$\left(\frac{3347}{9600}, \frac{279}{800}\right]$	$\frac{4737803}{8388608}$	$\frac{2496429}{4194304}$	$\frac{1686515}{2097152}$	$\frac{561566778711479378089}{3588524478084153344 00} \approx 1.5648$
$\left(\frac{279}{800}, \frac{3349}{9600}\right]$	$\frac{4736809}{8388608}$	$\frac{9987141}{16777216}$	$\frac{6745125}{8388608}$	$\frac{582566669044251570404083}{3722999711157622669312 00} \approx 1.5647$
$\left(\frac{3349}{9600}, \frac{67}{192}\right]$	$\frac{9472245}{16777216}$	$\frac{9988125}{16777216}$	$\frac{3372385}{4194304}$	$\frac{262141285966254773673 8383}{1675349870020930201190400} \approx 1.5646$
$\left(\frac{67}{192}, \frac{1117}{3200}\right]$	$\frac{1183859}{2097152}$	$\frac{9989109}{16777216}$	$\frac{6744415}{8388608}$	$\frac{149787182298284044767131}{9573427828691029721088 0} \approx 1.5646$
$\left(\frac{1117}{3200}, \frac{419}{1200}\right]$	$\frac{4734749}{8388608}$	$\frac{4995047}{8388608}$	$\frac{13488121}{16777216}$	$\frac{349485158710181485198726 1}{2233799826694573601587200} \approx 1.5645$
$\left(\frac{419}{1200}, \frac{3353}{9600}\right]$	$\frac{9468125}{16777216}$	$\frac{4995539}{8388608}$	$\frac{13487411}{16777216}$	$\frac{104840059363390436233829 41}{6701399480083720804761600} \approx 1.5644$
$\left(\frac{3353}{9600}, \frac{559}{1600}\right]$	$\frac{9466751}{16777216}$	$\frac{4996031}{8388608}$	$\frac{13486701}{16777216}$	$\frac{104834570377486721978696 99}{6701399480083720804761600} \approx 1.5643$
$\left(\frac{559}{1600}, \frac{671}{1920}\right]$	$\frac{4732689}{8388608}$	$\frac{9993047}{16777216}$	$\frac{13485991}{16777216}$	$\frac{116476765160037792179747 3}{7445999422315245338624 00} \approx 1.5642$
$\left(\frac{671}{1920}, \frac{839}{2400}\right]$	$\frac{2366001}{4194304}$	$\frac{9994031}{16777216}$	$\frac{6742641}{8388608}$	$\frac{104823597717232490626686 7}{6701399480083720804761 60} \approx 1.5642$
$\left(\frac{839}{2400}, \frac{1119}{3200}\right]$	$\frac{9462631}{16777216}$	$\frac{1249377}{2097152}$	$\frac{3371143}{4194304}$	$\frac{262045286279547126041 2937}{1675349870020930201190400} \approx 1.5641$
$\left(\frac{1119}{3200}, \frac{1679}{4800}\right]$	$\frac{9461257}{16777216}$	$\frac{9996001}{16777216}$	$\frac{6741931}{8388608}$	$\frac{249553882309917869707153}{1595571304781838286848 00} \approx 1.564$
$\left(\frac{1679}{4800}, \frac{3359}{9600}\right]$	$\frac{9459883}{16777216}$	$\frac{9996985}{16777216}$	$\frac{842697}{1048576}$	$\frac{655044611897385906503719}{4188374675052325502976 00} \approx 1.5639$
$\left(\frac{3359}{9600}, \frac{7}{20}\right]$	$\frac{9458509}{16777216}$	$\frac{4998985}{8388608}$	$\frac{6741221}{8388608}$	$\frac{524008262447244950211 9691}{3350699740041860402380 800} \approx 1.5638$
$\left(\frac{7}{20}, \frac{59}{166}\right]$	$\frac{9075501}{16777216}$	$\frac{2461325}{4194304}$	$\frac{13365787}{16777216}$	$\frac{690817828427621169838459}{4467599653389147203174 40} \approx 1.5462$
$\left(\frac{59}{166}, \frac{43}{120}\right]$	$\frac{4456807}{8388608}$	$\frac{9835417}{16777216}$	$\frac{13254243}{16777216}$	$\frac{585585282807580684587 3}{3818718879867783138508 8} \approx 1.5341$
$\left(\frac{43}{120}, \frac{3}{7}\right]$	$\frac{1045721}{2097152}$	$\frac{636339}{1048576}$	$\frac{6779541}{8388608}$	$\frac{447130049349370498541651}{2843017961247639129292 80} \approx 1.5727$
$\left(\frac{3}{7}, \frac{1}{2}\right]$	$\frac{8388625}{16777216}$	$\frac{8388625}{16777216}$	1	$\frac{47073977519962 3}{29826131951616 0} \approx 1.5782$