Windows Scheduling as a Restricted Version of Bin Packing

AMOTZ BAR-NOY
Brooklyn College.
RICHARD E. LADNER
University of Washington.
and
TAMI TAMIR
The Interdisciplinary Center.

Given is a sequence of n positive integers $w_1, w_2, \ldots, w_n$ that are associated with the items $1, 2, \ldots, n$ respectively. In the *windows scheduling* problem, the goal is to schedule all the items (equal length information pages) on broadcasting channels such that the gap between two consecutive appearances of page i on any of the channels is at most w_i slots (a slot is the transmission time of one page). In the *unit fractions bin packing* problem, the goal is to pack all the items in bins of unit size where the size (width) of item i is $1/w_i$. The optimization objective is to minimize the number of channels or bins. In the off-line setting, the sequence is known in advance; whereas in the on-line setting, the items arrive in order and assignment decisions are irrevocable. Since a page requires at least $1/w_i$ of a channel's bandwidth, it follows that windows scheduling without migration (all broadcasts of a page must be from the same channel) is a restricted version of unit fractions bin packing.

Let $H = \lceil \sum_{i=1}^{n} (1/w_i) \rceil$ be the bandwidth lower bound on the required number of bins (channels). The best known off-line algorithm for the windows scheduling problem used $H + O(\ln H)$ channels. This paper presents an off-line algorithm for the unit fractions bin packing problem with at most $H + 1$ bins. In the on-line setting, this paper presents algorithms for both problems with $H + O(\sqrt{H})$ channels or bins where the one for the unit fractions bin packing problem is simpler. On the other hand, this paper shows that already for the unit fractions bin packing problem, any on-line algorithm must use at least $H + \Omega(\ln H)$ bins. For instances in which the window sizes form a divisible sequence, an optimal online algorithm is presented. Finally, this paper includes a new NP-hardness proof for the windows scheduling problem.

Categories and Subject Descriptors: F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity; G.2.3 [**Mathematics of Computing**]: Discrete Mathematics—*Applications*

General Terms: Algorithms

Additional Key Words and Phrases: periodic scheduling, approximation algorithms, bin-packing, on-line algorithms

A preliminary version appeared in the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 217–226, 2004.
Author's address: A. Bar-Noy, Computer & Information Science Department, Brooklyn College, 2900 Bedford Ave., Brooklyn, NY 11210. amotz@sci.brooklyn.cuny.edu..
R. Ladner, Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195. ladner@cs.washington.edu.
T.Tamir, School of Computer Science, The Interdisciplinary Center, Herzliya, Israel, tami@idc.ac.il. Work done while the author was at the University of Washington.

1. INTRODUCTION

The input for the well known bin packing problem (BP) is a set of n item sizes $s_1, s_2, \ldots, s_n$ where $0 < s_i < 1$ for all $1 \leq i \leq n$. The goal is to pack these items in unit size *bins* using as few as possible bins where the total size of items packed in one bin does not exceed one. We study a variant of bin packing, called the *unit fractions bin packing* problem (UFBP), in which all sizes are unit fractions, i.e, of the form $1/w$ for some integer $w \geq 2$. In particular, we are interested in a packing that forms a solution to the *windows scheduling* problem (WS): given a sequence of n positive integers $w_1, w_2, \ldots, w_n$, called *windows*, that are associated with n equal length information pages (requests), the goal is to schedule all the pages on broadcasting channels such that the gap between two consecutive appearances of page i on the channels is at most w_i slots, where a slot is the time to broadcast one page. For example, the sequence of windows (and page names) $\langle 2, 4, 5\rangle$ can be scheduled on one channel by repeatedly transmitting the sequence $[2, 4, 2, 5]$ and the sequence of windows $\langle 2, 3, \ldots, 9\rangle$ can be scheduled on two channels by repeatedly transmitting the sequence $[2, 4, 2, 5]$ on the first channel and the sequence $[3, 6, 7, 3, 8, 9]$ on the second channel.

The following example illustrates the difference between UFBP and WS. Consider the sequence of windows $\langle 2, 3, 6\rangle$. Since $1/2 + 1/3 + 1/6 = 1$, the three items can be packed in one bin. On the other hand, there is no schedule of these pages for WS that uses only one channel since $gcd(2, 3) = 1$, and the only two ways to schedule 2 and 3 on the same channel is by repeatedly transmitting either the sequence $[2, 3]$ or the sequence $[2, 2, 3]$, which leaves no slots for scheduling the 6. In general, since a page requires at least $1/w_i$ of a channel's bandwidth, it follows that windows scheduling without migration (that is, when all broadcasts of a page must be from the same channel) is a restricted version of unit fractions bin packing.

The objective of our work is to compare the hardness of the two problems UFBP and WS in both the on-line and the off-line settings. In particular, we present the first off-line results for UFBP and the first on-line results for both UFBP and WS. As UFBP is a special case of BP, we expect algorithms for UFBP to have a better performance than the performance of known algorithms for BP. On the other hand, because WS without migrations is a restriction of UFBP, we expect algorithms for WS to have a worse performance than the performance of algorithms for UFBP.

There are two difficulties in solving WS. The first is the assignment of requests to channels and the second is determining the transmission slots for each request. The UFBP problem isolates the first difficulty. In a way, UFBP is the fractional version of WS that measures the power of unlimited preemptions. That is, UFBP demonstrates what can be achieved when a request is not necessarily transmitted non-preemptively in one slot, but instead can be partitioned into small segments as long as the total length of these segments in any window of w_i slots is one.

1.1 Notations and Definitions

Given a sequence σ of n item widths $\langle 1/w_1, \ldots, 1/w_n\rangle$ and a UFBP algorithm $\mathcal{B}$, define $N_{\mathcal{B}}(\sigma)$ to be the number of bins of unit size used by the algorithm to pack all the n items. Similarly, given a sequence σ of request windows $\langle w_1, \ldots, w_n\rangle$, and a WS algorithm $\mathcal{W}$, define $N_{\mathcal{W}}(\sigma)$ to be the number of channels used by the

algorithm to schedule all the n requests. Note that we use σ to denote both a sequence of items width $\langle 1/w_1, \ldots, 1/w_n \rangle$ for the UFBP problem, and a sequence of windows $\langle w_1, \ldots, w_n \rangle$ for the WS problem. In both cases, w_i is an integer for all $1 \leq i \leq n$.

In the *off-line* setting, for either UFBP or WS, the sequence σ is completely known to the algorithm in advance. In the *on-line* setting, for either UFBP or WS, the sequence σ is provided one item at a time and the algorithm must augment its current solution to accommodate a new item. That is, a decision made about which bin to pack an item in UFBP or how to schedule a request on the channels in WS, cannot be revoked.

Let OPTB denote an optimal off-line algorithm for UFBP and OPTW denote an optimal off-line algorithm for WS. The quantity $\sum_{i=1}^{n}(1/w_i)$ is the total width of all the items in σ. Since the number of bins in UFBP and the number of channels in WS must be an integer,

$$H(\sigma) = \left\lceil \sum_{i=1}^{n}(1/w_i) \right\rceil \tag{1}$$

is a lower bound on the performance of any algorithm for UFBP and WS on the sequence σ.

We do not know if the problem of finding the minimum number of bins in UFBP is NP-hard, but we do know that a compact representation of WS is NP-hard[1]. We conjecture that both problems are NP-hard, and thus, we seek approximation algorithms for the off-line UFBP and WS problems and competitive algorithms for their on-line versions. We express the bounds on the performance of an algorithm $\mathcal{A}$ in the form

$$H(\sigma) \leq N_{\mathcal{A}}(\sigma) \leq H(\sigma) + f(H(\sigma)) \tag{2}$$

for all σ, where f is a non-decreasing function. These bounds translate to upper bounds on approximation and competitive ratios in a natural way: The approximation ratio for an off-line algorithm $\mathcal{A}$ or the competitive ratio of an on-line algorithm $\mathcal{A}$ is

$$\rho(\mathcal{A}) = \sup_{\sigma} \left\{ \frac{N_{\mathcal{A}}(\sigma)}{N_{\mathsf{OPT}}(\sigma)} \right\} .$$

Suppose that for algorithm $\mathcal{A}$ there exists a bound of the form in inequality (2). Since f is non-decreasing and $H(\sigma)$ is a lower bound on $N_{\mathsf{OPT}}(\sigma)$ we have:

$$N_{\mathsf{OPT}}(\sigma) \leq N_{\mathcal{A}}(\sigma) \leq N_{\mathsf{OPT}}(\sigma) + f(N_{\mathsf{OPT}}(\sigma)) . \tag{3}$$

Hence,

$$\rho(\mathcal{A}) \leq 1 + \sup_{\sigma} \{ f(N_{\mathsf{OPT}}(\sigma))/N_{\mathsf{OPT}}(\sigma) \} .$$

This ratio can be interesting, but does not yield as much information as inequalities (2) and (3). Consequently, we prefer the form of the right inequalities of (2) and (3).

[1]In a compact representation the number of different windows is polynomial but the number of items could be exponential. See Section 2

1.2 Related Work

There is a wide literature on the general bin packing problem, see the survey [11]. First, bin packing is an NP-hard problem [13]. For the off-line problem, there exists an asymptotic PTAS that uses $(1+\varepsilon)N_{\mathsf{OPT}}(\sigma)+1$ bins [25]. The performance of the on-line algorithms *first-fit* (FF) and *best-fit* (BF) is analyzed in [17], where it is shown that $\rho(\mathsf{FF}), \rho(\mathsf{BF}) \leq 1.7$. The best known lower bound for any on-line bin packing algorithm $\mathcal{A}$, is $\rho(\mathcal{A}) \geq 1.540$ [24]. The best known on-line bin packing algorithm is *Harmonic++* whose competitive ratio is 1.589 [22]. In [10], the special case of BP with *divisible item sizes*, where in the sorted sequence $a_1 < a_2 < \cdots < a_m$ of item sizes, a_{i-1} divides a_i for all $1 < i \leq m$, is shown to be optimally solvable with a polynomial time algorithm. Another special version of the bin-packing problem, considered in [9], is *bin packing with discrete item sizes*, i.e., when items sizes are in $\{1/k, 2/k, \ldots, j/k\}$ for some $1 \leq j \leq k$. This version is not directly related to our problem, but its study demonstrates how the classic bin packing problem can be handled when the input set is restricted.

The windows scheduling problem for one channel is known as the pinwheel problem (e.g., [15; 7]). The windows scheduling problem for many channels was first defined in [4]. This paper shows how to construct schedules that use $H(\sigma) + O(\ln(H(\sigma))$ channels. This asymptotic result is complemented with a natural greedy algorithm that performs well in practice, but does not have a provable approximation bound.

Online UFBP and online WS was not studied earlier. Recently, the papers [27] and [8] studied the general online case in which deletions are allowed. The former addressed the online UFBP and the latter addressed the online WS.

There are several interesting applications of windows scheduling. The simplest is *harmonic* windows scheduling where the requests represent segments of popular movies. For $1 \leq i \leq n$, the window of segment i is $w_i = i$, where n is the number of equal size segments the movie is partitioned into. If segment i appears in every window of i time slots, then the maximum waiting time for any client who wishes to view the movie with no interruptions is the time it takes to broadcast one segment, or $1/n$ of the movie length. Harmonic windows scheduling is the basis of many popular media delivery schemes (e.g., [18; 16]). This concept of receiving from multiple channels and buffering data for future playback was first developed by [26]. A variant of harmonic WS for popular movie delivery is where the movie is partitioned into n segments and the window of segment i is $w_i = i + d - 1$ for a fixed constant d. As shown in [5], for any number of channels h, this variant can be used to construct schedules whose maximum delay is asymptotically close to the information theoretic lower bound of $1/(e^h - 1)$ that follows from [12].

Windows scheduling can be thought of as a scheduling problem for *push* (*proactive*) broadcast systems. One example is the Broadcast Disks environment (e.g., [1]) where satellites broadcast popular information pages to clients. Another example is the TeleText environment (e.g., [2]) in which running banners appear in some television networks. In such a system, there are clients and servers where the servers choose what information to push and in what frequency in order to optimize the quality of service for the clients (usually the waiting time). In a more generalized model, the servers are not the information providers (e.g., [14; 6]). They sell their

service to various providers who supply content and request that the content be broadcast regularly. The regularity can be defined by a window that translates to the maximum delay until a client receives a particular content.

Windows scheduling belongs to the general class of periodic scheduling problems that has applications in many disciplines (e.g., operations research, networking). The traditional optimization goal in periodic scheduling is an "average" type goal in which a request should be scheduled $1/w_i$ fraction of the time. The quality of an algorithm is determined by fairness issues. Among the most prominent examples are the hard real-time scheduling problem [21] and the chairman assignment problem [23]. On the other hand, the windows scheduling problem has a "max" type optimization goal in which the gap between two consecutive appearances of a request must be smaller than w_i. Both optimization goals may be practical to the many applications of periodic scheduling. Note that unit fractions bin packing is a relaxed version of both optimization goals.

1.3 Summary of Results

	Lower Bound	UFBP upper bound	WS upper bound
Off-line	$H(\sigma)$	$H(\sigma)+1$ [$\star$]	$H(\sigma)+O(\ln(H(\sigma)))$ [4]
On-line	$H(\sigma)+\Omega(\ln(H(\sigma)))$ [$\star$]	$H(\sigma)+O(\sqrt{H(\sigma)})$ [$\star$]	$H(\sigma)+O(\sqrt{H(\sigma)})$ [$\star$]

Table I. Results for UFBP and WS in the off-line and the on-line settings ([$\star$]: our results).

We prove the NP-hardness of WS without migrations in a compact representation of WS in which the number of different windows is polynomial but the number of items could be exponential. A previous known hardness result suits only the case of one channel when the gap between consecutive schedules of request i must be exactly w_i ([3]).

The off-line WS problem has a polynomial time algorithm [4] that uses $H(\sigma)+O(\ln(H(\sigma))$ channels. By contrast, we show that the *any-fit decreasing* strategy is a polynomial time algorithm for off-line UFBP that uses $H(\sigma)+1$ bins. This result demonstrates that UFBP is "easier" than WS in the off-line setting.

In the on-line setting for unit fractional bin packing, we first show that for any value h_0 there exists a sequence of requests σ with $H(\sigma)\geq h_0$ such that any on-line UFBP algorithm requires $H(\sigma)+\Omega(\ln(H(\sigma)))$ bins. This demonstrates that the on-line UFBP problem is "harder" than the off-line UFBP problem. Next, we give a tight analysis of the natural packing strategies: next-fit and first-fit. Finally, we give a polynomial time algorithm that uses $H(\sigma)+O(\sqrt{H(\sigma)})$ bins.

In the on-line setting for windows scheduling, we first present a non-trivial algorithm that uses the optimal number of channels $H(\sigma)$ when the windows form a divisible sequence (the equivalent problem for UFBP has a simpler greedy optimal algorithm [10]). Then, for general instances, we give an on-line algorithm that uses $H(\sigma)+O(\sqrt{H(\sigma)})$ channels. We emphasize that although this bound is the same as for on-line UFBP, the WS algorithm is substantially more complicated.

Table I summarizes the results for UFBP and WS in the off-line and on-line settings. Our results are marked with [⋆].

1.4 Paper Organization

Section 2 presents the NP-hardness result for WS. Section 3 describes the near optimal off-line algorithm for UFBP. Sections 4 and Section 5 present the results for on-line UFBP and on-line WS respectively. Section 6 discusses the open problems regarding UFBP and WS.

2. NP-HARDNESS

We distinguish between two representations of the windows scheduling problem. In the *standard representation*, the input is a sequence $\langle w_1, w_2, \ldots, w_n \rangle$, where each w_i is a positive integer represented by a binary sequence. In the *compact representation*, the input is a sequence of pairs $\langle (w_1, n_1)(w_2, n_2) \ldots, (w_m, n_m) \rangle$, where w_i and n_i are positive integers represented by binary sequences. The interpretation of this sequence is that there are n_i requests for window size w_i. Note that it is possible for the standard representation of a windows scheduling problem to be exponentially larger than its compact representation.

In this section, we show that windows scheduling, in the compact representation, without migration is NP-hard. That is, the problem is NP-hard when broadcasts of any particular page must be from the same channel. We do not know if the problem is still NP-hard when migration is permitted or when the problem is represented in its standard form. A previous hardness proof for the standard representation of the windows scheduling problem ([3]) is suitable only for a single channel for the restricted case where all the gaps between two consecutive appearances of request i must be exactly w_i. Our proof holds for arbitrary number of channels. For the standard representation it suits the exact gaps constraint, and for the compact representation we show hardness of the most general case in which gaps between schedules of request i may vary.

THEOREM 2.1. *The windows scheduling problem without migrations, in its compact representation, is NP-hard.*

PROOF. The proof is in two stages. In the first stage, the maximum 3-dimensional matching problem is reduced to a restricted version of the windows scheduling problem, called *periodic scheduling*. In the periodic scheduling version, given a sequence of windows $\langle w_1, w_2, \ldots, w_n \rangle$, the objective is to find a schedule with a minimum number of channels, where each request i is granted its exact window. In the second stage, the exact-window requirement is dropped by adding dummy requests all of the same size. Unfortunately, the number of dummy requests can be exponentially large, and therefore the proof holds only for the compact representation.

The maximum 3-dimensional matching problem (3DM), which is known to be NP-hard [19], is defined as follows on three disjoint sets X, Y, and Z each contains h items:

Input:. A set of $t \geq h$ distinct triplets $T \subseteq X \times Y \times Z$.

Output:. A 3-dimensional matching in T of maximum cardinality, i.e., a subset $T' \subseteq T$, such that any item in X, Y, Z appears at most once in T', and $|T'|$ is

maximum.

Given an instance of 3DM, construct the following instance of WS. Let $T = \{T_1, \ldots, T_t\}$ be the set of triplets in the 3DM instance. Assign a distinct prime number $p_j > 2$ to each triplet T_j for $1 \leq j \leq t$. The WS instance consists of $3h$ requests, one request for each item in $I = X \cup Y \cup Z$, where $w_i = \Pi_{(i \in T_j)} p_j$ for any $i \in I$.

Running example: Assume that the three sets are $X = \{a, b\}$, $Y = \{c, d\}$, and $Z = \{e, f\}$, and that there are three triplets: $T = \{(a, c, e), (a, d, f), (b, c, e)\}$. Then the prime number 3 is assigned to the triplet $\{a, c, e\}$, the prime number 5 is assigned to the triplet $\{a, d, f\}$, and the prime number 7 is assigned to the triplet $\{b, c, e\}$. The WS instance is $I = \{a, b, c, d, e, f\}$ for which $w_a = 15$, $w_b = 7$, $w_c = 21$, $w_d = 5$, $w_e = 21$, and $w_f = 5$.

LEMMA 2.2. *There is a matching of size h if and only if there is a periodic schedule for $\langle w_i : i \in I \rangle$ on h channels.*

PROOF. Given a matching, the following is a schedule of the windows $\langle w_i : i \in I \rangle$ on h channels. Let (x, y, z) be a triplet in the matching. By the construction, $gcd(w_x, w_y, w_z) = p_j > 2$ since all the triplets in T are distinct. A valid schedule assigns x to slots $1 + kw_x$, assigns y to slots $2 + kw_y$, and assigns z to slots $3 + kw_z$ for $k = 0, 1, 2, \ldots$. Since all the items of X, Y, Z are covered by the h triplets of the matching, all the requests are scheduled on h machines.

Given a schedule, the following is a 3-dimensional matching in T. It is not hard to see that in periodic scheduling, two requests i_1 and i_2 can be assigned to the same channel if and only if $gcd(w_{i_1}, w_{i_2}) > 1$. The prime assignment implies that if items i_1 and i_2 appear in the same triplet then $gcd(w_{i_1}, w_{i_2}) > 1$ and if no triplet contain both items, then $gcd(w_{i_1}, w_{i_2}) = 1$. As a result, in periodic scheduling, two requests i_1 and i_2 can be assigned to the same channel if and only if in the corresponding 3DM-3 problem the corresponding items appear in at least one triplet. Therefore, each channel contains at most three requests. Now, since $3h$ requests are scheduled, each channel must broadcast exactly three distinct requests and the whole schedule induces a 3-dimensional matching. □

Running example: The matching $\{(a, d, f), (b, c, e)\}$ implies that the requests a, d, and f can be assigned to one channel and that the requests b, c, and e can be assigned to another channel.

To remove the periodic constraint, add m dummy requests whose window is $w = \prod_{j=1}^{t} p_j$, such that the total width of all the requests is h. That is, construct the compact representation $\langle (w, m), (w_i, 1) : i \in I \rangle$ where $m/w + \sum_{i \in I} 1/w_i = h$. Since the width of all the requests in this instance is exactly h, then any schedule for these requests that uses h channels must be periodic. In particular, such a schedule contains a periodic schedule for the requests $\langle w_i : i \in I \rangle$. Hence, by the above claim, there is a matching of size h. On the other hand, if there is matching of size h, then by the above claim there is a periodic schedule for the requests $\langle w_i : i \in I \rangle$. It is not hard to see that the m dummy requests of size w can also be periodically scheduled in remaining empty slots.

Running example: Add 136 dummy requests with $w = 105$ ($105 = 3 \cdot 5 \cdot 7$) since $136/105 + \sum_{i \in I} 1/w_i = 2$.

It remains to argue that this compact representation can be constructed in polynomial time of the length of the instance of the 3DM problem.

Let n be the number of triplets in T. By the prime number theorem, the n-th prime number is of order $O(n \log n)$ and therefore the first n prime numbers all have an $O(\log n)$ length. Clearly, the first n prime numbers can be computed in polynomial time in n. Each w_i is the product of at most three of these prime numbers and w is the product of all the n prime numbers, so they can be computed in polynomial time. Finally, $m = w \cdot h - \sum_{i \in I} w/w_i$ can also be computed in polynomial time and its length is polynomial in n.

Hardness of UFBP: For the UFBP problem, migration makes the problem trivial since it means that items can be split among several bins. In this case, the greedy packing is clearly optimal. However, without splits, we only know that it is NP-hard if the size of a bin is not necessarily 1, but some fraction $1/k$ where k is an integer given as part of the input. Formally,

THEOREM 2.3. *For bins of arbitrary unit-fraction size, the UFBP problem is NP-hard.*

PROOF. We describe a reduction from *Partition*: Given a set A of n items of sizes $\{a_1, a_2, \ldots, a_n\}$ having total size $2B$, and the question whether a subset having total size B exists, construct the following instance for UFBP: Let $L = LCM\{a_1, a_2, \ldots, a_n, B\}$. The unit fractions to be packed have sizes $\{\frac{a_1}{L}, \frac{a_2}{L}, \ldots, \frac{a_n}{L}\}$, and the bin size is $\frac{B}{L}$. Clearly, the definition of L guarantees that all involved values are unit fraction. It is easy to verify that this instance can be packed in two bins if and only if A has a partition. □

3. OFF-LINE UNIT FRACTIONS BIN PACKING

In this section, we present a polynomial time algorithm for UFBP which is optimal up to an additive constant of 1. Consider the following off-line bin packing algorithm.

Any Fit Decreasing (AFD): The items are processed in a non-increasing order of their widths. The current item is packed in any bin it fits in if such a bin exists. Otherwise, the current item is packed in a new bin.

Note that for the AFD strategy, it is not important which bin is used to pack an item because the analysis suits any bin selection (e.g, first-fit or best-fit).

THEOREM 3.1. $N_{\mathsf{AFD}}(\sigma) \leq H(\sigma) + 1$ *for any sequence* σ.

PROOF. After being sorted in a non-increasing order, the input sequence has the form

$$\sigma = \left\langle \left(\frac{1}{2}\right)^{n_2}, \left(\frac{1}{3}\right)^{n_3}, \ldots, \left(\frac{1}{z}\right)^{n_z} \right\rangle$$

for some integers $z \geq 2$ and $n_i \geq 0$ for $2 \leq i \leq z$. Assume that AFD uses h *full* bins (filled to capacity 1) and h' *non-full* bins. Thus, $N_{\mathsf{AFD}}(\sigma) = h' + h$.

CLAIM 3.2. *After packing all the items of width at least* $1/k$, *there are at most* $k - 1$ *non-full bins.*

PROOF. The proof is by induction on k. The base case is $k = 2$. Clearly, the items of width $1/2$ are packed in $\lceil n_2/2 \rceil$ bins, where only the last one might be non-full. Assume that the claim holds before packing the items of width $1/k$. That is, after packing the items of width at least $1/(k-1)$, there are at most $k-2$ non-full bins. Since items of size $1/k$ are first added to currently non-full bins that can accommodate them, it follows that only one bin that contains only items of size $1/k$ might be non-full after all the $1/k$-items are packed. □

Suppose the last bin that AFD opened was opened for an item of width $1/z'$ where $z' \leq z$. By Claim 3.2, at this stage, there are less than z' non-full bins. Since this is the last opened bin, it follows that $h' < z'$. Furthermore, each of the first $h'-1$ non-full bins must contain items whose total width is greater than $1-(1/z')$, because otherwise AFD would not open a new bin for $1/z'$. By definition, the last non-full bin contains at least one item of width $1/z'$. It follows that

$$\begin{aligned} H(\sigma) &\geq h + (h'-1)\left(1 - \frac{1}{z'}\right) + \frac{1}{z'} \\ &= h + h' - 1 - \frac{h'-2}{z'} > h + h' - 2 \ . \end{aligned}$$

Since $H(\sigma)$ is an integer, it must be at least $h'+h-1$. Thus, $N_{\mathsf{AFD}}(\sigma) = h'+h \leq H(\sigma)+1$.

Since $H(\sigma) \leq N_{\mathsf{OPTB}}(\sigma)$, it follows that

COROLLARY 3.3. $N_{\mathsf{AFD}}(\sigma) \leq N_{\mathsf{OPTB}}(\sigma) + 1$.

The above analysis is tight. Consider the sequence $\sigma = \left\langle \frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4} \right\rangle$. An optimal solution uses two bins, the first contains $\left\{\frac{1}{2}, \frac{1}{4}, \frac{1}{4}\right\}$ and the second contains $\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{4}\right\}$. On the other hand, AFD packs the first two items in one bin, the next three items in another bin, and then it is forced to pack the last item of width $1/4$ in a third bin since the available free space in each of the first two bins is $1/6$. Hence, $N_{\mathsf{AFD}}(\sigma) \geq N_{\mathsf{OPTB}}(\sigma) + 1$.

Remark: The above theorem gives a clear distinction between BP and UFBP. This is because in BP, $N_{\mathsf{OPT}}(\sigma)$ can be arbitrarily close to $2H(\sigma)$. For example, this is the case for the sequence σ that contains items of width $1/2 + \varepsilon$ for a very small ε.

4. ON-LINE UNIT FRACTIONS BIN PACKING

In this section we address the on-line UFBP problem. We first show a non-trivial lower bound. Next, we analyze "fit" greedy algorithms. Finally, we show a better algorithm that sometimes opens a new bin for an item even if a bin with enough free space to accommodate this item exists.

4.1 An $H(\sigma) + \Omega(\ln H(\sigma))$ Lower Bound for On-line UFBP

We prove that no on-line algorithm for UFBP can guarantee a solution with $H(\sigma) + o(\ln(H(\sigma)))$ bins for any sequence σ. Since there exists an upper bound of $H(\sigma)+1$ for the off-line UFBP, this result shows a significant gap between what can be achieved off-line and what can be achieved on-line for UFBP.

THEOREM 4.1. *For any on-line algorithm $\mathcal{B}$ for UFBP and for any integer $h_0 > 0$, there exists a sequence σ such that $H(\sigma) \geq h_0$ and $N_{\mathcal{B}}(\sigma) = H(\sigma) + \Omega(\ln(H(\sigma)))$.*

PROOF. Let w be the smallest integer such that $\sum_{i=2}^{w} 1/i \geq h_0$. It follows that $h_0 = \Theta(\ln w)$. We describe an adversary strategy that constructs a non-decreasing sequence σ of the type

$$\left\langle \left(\frac{1}{w}\right)^{x_w}, \left(\frac{1}{w-1}\right)^{x_{w-1}}, \ldots, \left(\frac{1}{2}\right)^{x_2} \right\rangle$$

for integers $x_i \geq 0$ for $2 \leq i \leq w$. The adversary sets the values of $x_w, x_{w-1}, \ldots, x_2$ as follows. Let $2 \leq i \leq w$ and assume $x_w, \ldots, x_{i+1}$ have been already set. The adversary requests items of width $1/i$ until one of the following two conditions holds for the bins used by Algorithm $\mathcal{B}$:

(1) There is a bin containing exactly $i-1$ items of width $1/i$ and no other items.
(2) There are at least ih_0 bins which are filled only with items of width $1/i$ but no more than $i-2$ such items.

Note that if x_i is large enough then one of the two conditions must hold (for $i = 2$ the first condition must hold). If the first condition holds and $i > 2$, then the adversary starts requesting items of width $1/(i-1)$. If the second condition holds the adversary stops (formally, it sets $x_{i-1} = \cdots = x_3 = x_2 = 0$).

The Theorem is proved using the following claims.

CLAIM 4.2. $H(\sigma) \geq h_0$.

PROOF. If the first condition holds for all i, then $H(\sigma) \geq \sum_{i=2}^{w}((i-1)/i)$ which by definition is at least h_0. If the second condition holds, then for some i there are ih_0 bins each filled with at least one item of width $1/i$. Hence, $H(\sigma) \geq h_0$. □

CLAIM 4.3. *The total free space in all of the bins of $\mathcal{B}$ is at least $\Theta(\ln(w))$.*

PROOF. Assume first that the first condition holds for all $2 \leq i \leq w$. This implies that when the first item of width $1/(i-1)$ is given to $\mathcal{B}$, there is a bin accommodating exactly $i-1$ items of size $1/i$. This bin can never be filled more by $\mathcal{B}$ since later items all have width larger than $1/i$. Thus at the end, the total free space is at least $1/2 + 1/3 + \cdots + 1/w = \Theta(\ln(w))$. Now assume that the second condition holds for some i. Hence, there are at least ih_0 bins which are filled only with items of width $1/i$ each with a free capacity of at least $2/i$. Thus, the total free capacity in these bins alone is at least $2h_0 = \Theta(\ln w)$. □

CLAIM 4.4. $N_{\mathcal{B}}(\sigma) = O(w^2 \ln(w))$.

PROOF. The worst case for Algorithm $\mathcal{B}$ happens when for each $2 \leq i \leq w$, the first condition holds after there are exactly $ih_0 - 1$ bins each containing at most $i-2$ items of width $1/i$. In this case, the total number of bins used by Algorithm $\mathcal{B}$ is at most $\sum_{i=2}^{w} ih_0 = O(w^2 \ln(w))$. □

By Claim 4.3, $H(\sigma) \leq N_{\mathcal{B}}(\sigma) - \Theta(\ln(w))$. Hence, $N_{\mathcal{B}}(\sigma) \geq H(\sigma) + \Theta(\ln(w))$. By Claim 4.4, $\ln(w) = \Omega(\ln(N_{\mathcal{B}}(\sigma)))$. Because $N_{\mathcal{B}}(\sigma) \geq H(\sigma)$, we have $\ln(w) = \Omega(\ln(H(\sigma)))$. Hence, $N_{\mathcal{B}}(\sigma) \geq H(\sigma) + \Theta(\ln(H(\sigma)))$.

By Theorem 3.1, $N_{\mathsf{OPTB}}(\sigma) \leq H(\sigma) + 1$. Combined with the above, we have

COROLLARY 4.5. *For any on-line algorithm $\mathcal{B}$ for UFBP and for any constant h_0, there exists a sequence σ such that $H(\sigma) \geq h_0$ and $N_{\mathcal{B}}(\sigma) = N_{\mathsf{OPTB}}(\sigma) + \Omega(\ln(N_{\mathsf{OPTB}}(\sigma)))$.*

4.2 First-fit and Next-Fit

In this section we consider two simple on-line strategies for UFBP. We analyze the performance of the *Next-fit* and the *First-fit* strategies defined as follows:
Next Fit (NF): An item is packed in the last opened bin if this bin has enough free space for it. Otherwise, a new bin is opened.
First Fit (FF): An item is packed in the first bin that has enough free space for it. If no bin has enough space, a new bin is opened.

The following results present the exact competitive ratios of NF and FF. That is, for $\mathcal{A} \in \{\mathsf{NF}, \mathsf{FF}\}$, (i) for any sequence σ, $N_{\mathcal{A}}(\sigma) \leq \rho(\mathcal{A})N_{\mathsf{OPTB}}(\sigma)$, and, (ii) for any n there exists a sequence σ of size $\Theta(n)$ for which $N_{\mathcal{A}}(\sigma) = \rho(\mathcal{A})N_{\mathsf{OPTB}}(\sigma)$.

THEOREM 4.6. $\rho(\mathsf{NF}) = 2$.

PROOF. The known upper bound for general BP applies also for UFBP thus, $\rho(\mathsf{NF}) \leq 2$. To see that this bound is tight consider the following sequence of $4x$ item width for $x \geq 1$:

$$\frac{1}{2}, \frac{1}{2x}, \frac{1}{2}, \frac{1}{2x}, \ldots, \frac{1}{2}, \frac{1}{2x} .$$

An optimal solution packs all the items with width $1/2$ in x bins and the rest of the items in one additional bin for a total of $x+1$ bins. NF allocates $2x$ bins for $2x$ pairs of items one of width $1/2$ and one of width $1/2x$. Thus, the competitive ratio of NF for this sequence is $\frac{2x}{x+1}$ which tends to 2 when x tends to infinity. □

THEOREM 4.7. $\rho(\mathsf{FF}) = \frac{6}{5}$.

PROOF. To see that $\rho(\mathsf{FF}) \geq \frac{6}{5}$, consider the following sequence of $12x$ item width for $x \geq 1$:

$$\frac{1}{2}, \frac{1}{3}, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{2}, \frac{1}{3} .$$

An optimal solution packs all the items with width $1/2$ in $3x$ bins and all the items with width $1/3$ in $2x$ bins for a total of $5x$ bins. FF allocates a bin for any pair of adjacent items one of width $1/2$ and one of width $1/3$ for a total of $6x$ bins. Thus, the competitive ratio of FF is at least $\frac{6x}{5x} = \frac{6}{5}$.

For the upper bound, let σ be an arbitrary sequence of items. We claim that all the bins have load at least $5/6$ except maybe 4 special bins.

(1) After packing the first bin whose final load is at most $5/6$, all items that are packed in later bins must have width larger than $1/6$ (because first-fit will place the first item of size at most $1/6$ in the bin). Also note that no linear combination of $1/2$, $1/3$, $1/4$, and $1/5$ (which are the possible values for items in following bins) is between $48/60$ and $50/60$. That is, less than $5/6$ and greater than $4/5$. Thus, if another bin with load less than $5/6$ exists its load is at most $4/5$.
(2) After packing the first bin whose final load is at most $4/5$, all later bins contain only items with widths $1/2, 1/3$, or $1/4$. The load of each such bin is at least $10/12 = 5/6$ or at most $9/12 = 3/4$.

(3) After packing the first bin whose final load is at most 3/4, all later bins contain only items with widths 1/2 or 1/3. The load of each such bin is at least 5/6 or at most 2/3.

(4) After packing the first bin whose final load is at most 2/3, all later bins contain only items with widths 1/2, and only a single such bin (the last one) is not full.

Therefore, FF allocates at most $(6/5)H(S)+4$ bins to the sequence σ. Thus, for any $\varepsilon > 0$ and for any long enough sequences $\rho(\mathsf{FF}) \leq \frac{6}{5} + \varepsilon$. □

Remark: For *Next-fit*, the competitive ratio for UFBP is the same as the one known for BP. For *First-fit*, the 1.2-competitive ratio for UFBP is smaller than the 1.7-competitive ratio for BP ([17]). In a way, this demonstrates that UFBP is "easier" than BP.

4.3 An $H(\sigma) + O(\sqrt{H(\sigma)})$ Algorithm

The previous section demonstrated the limitation of "must fit" type algorithms. In order to get a better result, we develop algorithms that sometimes open a new bin for an item even if this item fits into one of the previously opened bins. The idea is similar to the well known *Harmonic* algorithm for classical Bin packing problem [20]. In the Harmonic algorithm, for some parameter k, and for each i, $1 \leq i < k$ there is an active bin for items of size in $(\frac{1}{i+1}, \frac{1}{i}]$, and one active bin for all items of size at most $1/k$. Note that when the input is limited to unit fractions, it means that each active bin can hold items of a single size - $\frac{1}{i}$. Formally,

DEFINITION 4.1. *A bin is* i-dedicated *if only items of size* $1/i$ *are packed in it.*

Similar to the Harmonic algorithm, we define a set of on-line algorithms $\{\mathcal{B}_k^*\}$ for $k = 1, 2, \ldots$ such that for any sequence σ, $N_{\mathcal{B}_k^*}(\sigma) \leq \frac{k+1}{k} H(\sigma) + k$. Algorithm $\mathcal{B}_1^*$ is the first-fit algorithm. Algorithm $\mathcal{B}_2^*$ dedicates bins to items of width 1/2 and packs all other items according to the first-fit rule. That is, an item of width 1/2 is either packed in an open 2-dedicated bin or in a new 2-dedicated bin and any item with a smaller width is packed in the first non-dedicated bin that can accommodate it. In general, Algorithm $\mathcal{B}_k^*$ dedicates bins to items of width $1/2, 1/3, \ldots, 1/k$ and packs all the items with smaller width according to first-fit rule. That is, an item of width $1/j$, for $2 \leq j \leq k$, is either packed in an open j-dedicated bin or in a new j-dedicated bin and any item with a smaller width is packed in the first non-dedicated bin that can accommodate it.

We note that unlike the Harmonic algorithm, the small items are packed according to first fit rule instead of next fit. This change does not affect the analysis of Algorithm $\mathcal{B}_k^*$, but is going to be crucial in the analysis of our final algorithm in which the value of k is changing dynamically during the execution of the algorithm.

LEMMA 4.8. *For any sequence* σ *and* $k > 0$,

$$N_{\mathcal{B}_k^*}(\sigma) \leq \frac{k+1}{k} H(\sigma) + k \ .$$

PROOF. Let $\sigma = \langle w_1, w_2, \ldots, w_n \rangle$. For all $j = 2, \ldots, k$, all j-dedicated bins, except maybe for the last one, are full (each containing j items of size $1/j$). Thus, there are at most $k-1$ non-full dedicated bins. The other bins are filled according

to first-fit rule with items whose width is at most $1/(k+1)$. Hence, a new non-dedicated bin is opened only if all previously opened non-dedicated bins are at least $k/(k+1)$-full. Thus, all the non-dedicated bins except maybe for the last one are at least $k/(k+1)$-full. Adding the last non-dedicated bin to the $k-1$ non-full dedicated bins, we get that there are at most k bins whose load could be small, and the total number of bins used is

$$N_{\mathcal{B}_k^*}(\sigma) \leq \frac{k+1}{k}\left(\sum_{i=1}^{n} 1/w_i\right) + k \leq \frac{k+1}{k}H(\sigma) + k \ .$$

□

Let $h = \sqrt{H(\sigma)}$. For simplicity, assume that h is an integer. Otherwise, round h to the nearest integer and the analysis is similar. Assume first that $H(\sigma)$ is known in advance. The expression $\frac{k+1}{k}H(\sigma) + k$ is minimized for $k = h$. The following lemma gives the bound for $\mathcal{B}_h^*$.

LEMMA 4.9. *For any sequence σ,*

$$N_{\mathcal{B}_h^*}(\sigma) \leq H(\sigma) + 2\sqrt{H(\sigma)}.$$

PROOF. Setting $k = \sqrt{H(\sigma)}$ in the statement of Lemma 4.8 implies that

$$\begin{aligned} N_{\mathcal{B}_k^*}(\sigma) \ &\leq \ H(\sigma) + \frac{H(\sigma)}{\sqrt{H(\sigma)}} + \sqrt{H(\sigma)} \\ &= \ H(\sigma) + 2\sqrt{H(\sigma)} \ . \end{aligned}$$

□

When $H(\sigma)$ is not known in advance, the algorithm dynamically increases the parameter k for which dedicated bins exist for $1/2, 1/3, \ldots, 1/k$. Algorithm $\mathcal{B}_{dyn}^*$ is defined as follows: Let H' denote the total width of the already packed items. As long as $H' \leq 1$, use $\mathcal{B}_1^*$ (regular first-fit), when $1 < H' \leq 4$, shift to $\mathcal{B}_2^*$, and in general, when $(k-1)^2 < H' \leq k^2$, use Algorithm $\mathcal{B}_k^*$. Note that when $\mathcal{B}_{dyn}^*$ shifts to Algorithm $\mathcal{B}_k^*$, it continues to use the bins that were used for $\mathcal{B}_{k-1}^*$, it just adds a new (initially empty) k-dedicated bin.

THEOREM 4.10. *For any sequence σ,*

$$N_{\mathcal{B}_{dyn}^*}(\sigma) \leq H(\sigma) + 4\sqrt{H(\sigma)}.$$

PROOF. Let $\sigma = \langle w_1, w_2, \ldots, w_n \rangle$ and $\mathcal{B}_h^*$ be the last algorithm performed by $\mathcal{B}_{dyn}^*$ on this sequence. Define s_k, for $1 \leq k \leq h$, to be the index of the first item packed while executing $\mathcal{B}_k^*$ and $s_{h+1} = n+1$. It follows that algorithm $\mathcal{B}_k^*$ packs the items of σ indexed s_k to $s_{k+1} - 1$. The total width of items indexed s_1 to $s_2 - 1$ is less than $1 + \frac{1}{2}$, the total width of items indexed s_2 to $s_3 - 1$ is less than $2^2 + \frac{1}{2} - 1^2 = 3$, and in general, the total width of items indexed s_k to $s_{k+1} - 1$ is less than $k^2 + \frac{1}{2} - (k-1)^2 < 2k$. The additive term $\frac{1}{2}$ exists since there might be an overflow of $\frac{1}{2}$ beyond $(k-1)^2$ before the algorithm shifts to $\mathcal{B}_k^*$.

As in the proof of Lemma 4.8, it follows that while $\mathcal{B}_k^*$ is executed, the algorithm opens a new bin only if all the current opened bins (including non-dedicated bins that have been opened during the execution of $\mathcal{B}_{k-1}^*$) are at least $k/(k+1)$-full.

This is true because the algorithm packs the small items according to the first-fit rule (unlike the original Harmonic algorithm). In addition, during the execution of $\mathcal{B}^*_k$, there might be at most $k-1$ non-full dedicated bins and only one non-dedicated bin with small load (the last one).

Recall that $h = \sqrt{H(\sigma)}$. Thus, $\mathcal{B}^*_h$ is the last algorithm executed by $\mathcal{B}^*_{dyn}$. The total number of bins used by $\mathcal{B}^*_{dyn}$ is at most

$$h + \sum_{k=1}^{h} \frac{k+1}{k} \sum_{i=s_k}^{s_{k+1}-1} \frac{1}{w_i} < h + \sum_{k=1}^{h} \frac{k+1}{k} 2k$$
$$= h + \sum_{k=1}^{h} 2k + \sum_{k=1}^{h} 2 = h^2 + 4h = H(\sigma) + 4\sqrt{H(\sigma)}.$$

□

5. ON-LINE WINDOWS SCHEDULING

In this Section we consider on-line algorithms for the windows scheduling problem. Since in these algorithms each request is scheduled only on one channel (no migration), each of these algorithms is also suitable for the unit fractions bin packing problem. We first describe and analyze an optimal algorithm for the special case in which the sequence σ contains only windows with divisible window sizes. In such instances, in the sorted sequence $w'_1 < w'_2 < \cdots < w'_m$ of window sizes, w'_{i-1} divides w'_i for all $1 < i \leq m$. In particular, for any c, the algorithm is optimal for instances in which all the w_i's are of the form $c2^{v_i}$ for an odd constant integer $c \geq 1$ and integers $v_i \geq 0$ for $1 \leq i \leq n$. By combining the optimal algorithms for different c values we get our final algorithm, $\mathcal{W}^*_{dyn}$ that has the same performance for the windows scheduling problem as Algorithm $\mathcal{B}^*_{dyn}$ has for the unit fractions bin packing problem.

5.1 An Optimal Algorithm for Instance with Divisible Window Sizes

Recall that a WS instance with *divisible window sizes* is one in which the sorted sequence of window request sizes is $a_1 < a_2 < \cdots < a_m$ such that a_{i-1} divides a_i for all $1 < i \leq m$. In this section we present an optimal algorithm for online WS of such instances. The algorithm is "strongly" online in the sense that there is no need to know in advance the ratios $\{r_i = (a_i/a_{i-1})\}$, the number of requests, and their total width. The only a-priori knowledge required is that the input instance has divisible window sizes.

For simplicity, we first describe an optimal algorithm for instances in which all window sizes are powers of 2. That is, $w_i = 2^{v_i}$ for an integer $v_i \geq 1$ for $1 \leq i \leq n$. Later, we generalize this optimal algorithm for any instance with divisible size windows.

5.1.1 *Instances with Power-2 Window Sizes.* Each channel schedule is represented by a rooted tree. For power-2 windows, each channel schedule is represented by a rooted binary tree, in which all the internal nodes have 2 children. In this tree, each item that is scheduled on this channel is assigned to a leaf (and this leaf is assigned to this item). To construct the schedule from a tree, alternate between

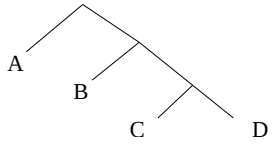

Fig. 1. Tree representation of the cyclic schedule $[A, B, A, C, A, B, A, D]$.

scheduling an item from the left and the right subtrees. The item selected from each subtree is selected by alternating recursively between the left and right subtree in each subtree. For example, the tree in Figure 1 represents a schedule that alternates between 'A' (the only item in the left subtree) and an item from the right subtree. In selecting this right-subtree item, the schedule alternates between 'B' and an item from the right subtree, and so on. It follows that a leaf, ℓ, whose depth in the tree is $d(\ell)$ represents an item scheduled with window size $2^{d(\ell)}$. The whole schedule is represented by a forest of rooted binary trees, each representing one channel.

Call a leaf that is not assigned yet to any request an *open* leaf and call a tree that has at least one open leaf an *active* tree. By definition, an active tree represents a channel schedule with idle slots. Let the label of a leaf ℓ of depth $d(\ell)$ be $2^{d(\ell)}$.

DEFINITION 5.1. *A* lace binary tree *of height h is a binary tree of height h in which there is a single leaf in each of the depths $1, 2, \ldots, h-1$, and two leaves in depth h. For example, the tree in Figure 1 is a lace binary tree of height* 3.

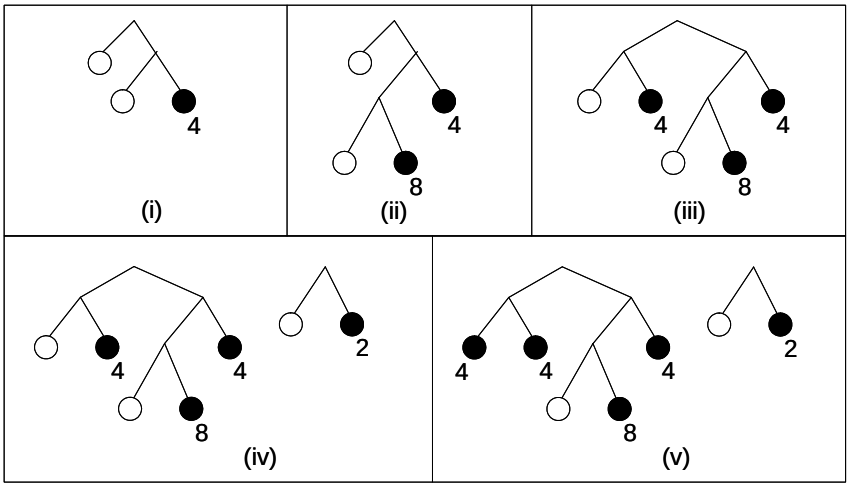

Fig. 2. Algorithm $\mathcal{W}_1$ evolution for $\sigma = \langle 4, 8, 4, 2, 4 \rangle$. White circles denote open leaves.

Algorithm $\mathcal{W}_1$: Let $w_i = 2^{v_i}$ be the next request in σ. Let $u \leq v_i$ be maximal such that an open leaf in one of the active trees whose label is 2^u exists. If there is no such u, open a new active tree with one open leaf whose label is $2^0 = 1$ and then define $u = 0$. Let ℓ be the leaf whose label is 2^u and let T be the active tree

containing the leaf ℓ. If $u = v_i$, then assigned request i to leaf ℓ. Otherwise, append to T a lace binary sub-tree of depth $v_i - u$ whose root is ℓ and assign request i to one of the two leaves with depth v_i in T. Figure 2 illustrates the evolution of the algorithm for $\sigma = \langle 4, 8, 4, 2, 4\rangle$.

LEMMA 5.1. *During the execution of algorithm $\mathcal{W}_1$, the forest of trees contains at most one open leaf in each depth.*

PROOF. The proof is by induction on the number of requests scheduled by the algorithm. Initially, there are no trees and therefore no open leaves. Assume the claim holds for the first $k-1$ requests and let the window size of the kth request be 2^v. When scheduling this request, algorithm $\mathcal{W}_1$ either closes an open leaf or replaces an open leaf with a lace binary tree. In the former case, there are no new open leaves and the claim remains true due to the induction hypothesis. In the latter case, since the replacement is performed on an open leaf with depth u, where $u \leq v$ and u is maximal, there are no open leaves of depths d for $u < d \leq v$. By the definition of a lace binary tree, after the replacement and the assignment of the request to the leaf of depth v, there is one new open leaf at depths d for $u < d \leq v$, exactly where no open leaves existed before. Note that at depth v two open leaves are added, but only one is open, because the other is assigned to the new request. □

LEMMA 5.2. *$N_{\mathcal{W}_1}(\sigma) = H(\sigma)$ for any sequence σ in which $w_i = 2^{v_i}$ for all $1 \leq i \leq n$.*

PROOF. Let $\sigma = \langle w_1 = 2^{v_1}, w_2 = 2^{v_2}, \ldots, w_n = 2^{v_n}\rangle$. We show that when the forest contains h trees and a new tree is opened by $\mathcal{W}_1$, then the total bandwidth of requests in σ (including the new one) $\sum_{i=1}^{n} 1/w_i$ is greater than h. Assume that the request that caused $\mathcal{W}_1$ to open a new tree has a window 2^{v_i}. According to the algorithm, there is no open leaf whose label is less than 2^{v_i}. Also, by Lemma 5.1, there is at most one open leaf whose label is 2^{v_i+j} for all $j > 0$. It follows that the total bandwidth of the open leaves in all the first h opened trees is at most $\sum_{j=1}^{\infty} 1/2^{v_i+j}$ which is less than $1/2^{v_i}$. Therefore, the total bandwidth required by the new $1/2^{v_i}$ request and the requests already scheduled is more than h. As a result, at the end, if the algorithm opened h trees then $H(\sigma) = h$. □

5.1.2 *Generalization for any Divisible Size Instance.* Let $a_1 < a_2 < \cdots < a_m$ be the divisible sequence of window request sizes, such that a_{i-1} divides a_i for all $1 < i \leq m$. To simplify the notation we add a dummy window size $a_0 = 1$. For $i > 0$, denote by r_i the ratio a_i/a_{i-1}. For example, the input $\langle 6, 30, 2, 60, 30, 6, 420\rangle$ is an instance with divisible windows in which the sorted window sizes are $1, 2, 6, 30, 60, 420$. The window request sizes can be described by the sequence of ratios. In the example, $r_1 = 2, r_2 = 3, r_3 = 5, r_4 = 2, r_5 = 7$. A power of 2 instance is the special case in which for all i, $r_i = 2$. We assume that all the ratios r_i are *prime* numbers. This assumption is w.l.o.g since otherwise we can add the intermediate values of window sizes without changing the instance (as the sequence $\{a_i\}$ describes just the potential values of requests). For example $r_1 = 2, r_2 = 3, r_3 = 5, r_4 = 2, r_5 = 7$ describes the window sizes in the input $\langle 6, 420\rangle$ and in the input $\langle 30, 6, 2, 60, 420\rangle$.

Given a divisible windows instance with ratio values $r_1, \ldots, r_z$, a *lace tree over the ratios* $\langle r_i\rangle$ is a tree whose root has r_1 children. For $1 \leq i < z$, at depth i there

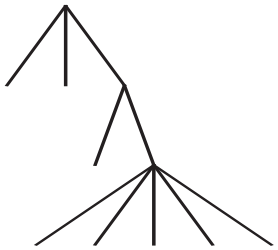

Fig. 3. A lace tree for window sizes $\{3, 6, 30\}$.

are r_i nodes, out of which $r_i - 1$ are leaves and a single node has r_{i+1} children. At depth z there are r_z leaves. Figure 3 illustrates a lace tree over the ratios $\langle 3, 2, 5\rangle$.

The following is an optimal on-line algorithm for any divisible instance. It is based on the tree representation described for power-2 instances in Section 5.1.1.

Algorithm $\mathcal{W}_d$: Let the window request sizes be $a_0 = 1 < a_1 < a_2 < \cdots < a_m$. Each channel schedule is represented by a lace tree over the ratios $\{r_i = a_i/a_{i-1}\}$. In this tree, each item that is scheduled on this channel is assigned to a leaf (and this leaf is assigned to this item). The construction of the schedule from a tree is done in the same manner as schedules are constructed in lace binary trees. It follows that a leaf, ℓ, whose depth in the tree is $d(\ell)$ represents an item scheduled with window size $a_{d(\ell)}$. The whole schedule is represented by a forest of lace trees, each representing one channel.

Call a leaf that is not assigned yet to any request an *open* leaf and call a tree that has at least one open leaf an *active* tree. By definition, an active tree represents a channel schedule with idle slots. Let the label of a leaf ℓ of depth $d(\ell)$ be $a_{d(\ell)}$.

Let $w_i = a_k$ be the next request in σ. Let $a_j \le a_k$ be maximal such that an open leaf in one of the active trees whose label is a_j exists. If there is no such a_j, open a new active tree with one open leaf whose label is $a_0 = 1$. Let ℓ be the leaf whose label is a_j and let T be the active tree containing the leaf ℓ. If $a_j = a_k$, then assigned request i to leaf ℓ. Otherwise, build a lace tree of depth $k - j$ over the ratios $r_{j+1}, r_{j+2}, \ldots, r_k$, and append it to T with ℓ as the root. Assign request i to one of the r_k leaves with label a_k in T.

Figure 4 illustrates the evolution of the algorithm for $\sigma = \langle 3, 6, 18, 3, 6\rangle$.

The following is a generalization of Lemma 5.1 (proof omitted).

LEMMA 5.3. *During the execution of algorithm $\mathcal{W}_d$, the forest of trees contains at most $r_k - 1$ open leaves in depth k.*

THEOREM 5.4. *$N_{\mathcal{W}_d}(\sigma) = H(\sigma)$ for any sequence σ with divisible sizes.*

PROOF. Let $\sigma = \langle w_1, w_2, \ldots, w_n\rangle$. We show that when the forest contains h trees and a new tree is opened by $\mathcal{W}_d$, then the total bandwidth of requests in σ (including the new one) $\sum_{i=1}^{n} 1/w_i$ is greater than h. Assume that the request that caused $\mathcal{W}_d$ to open a new tree has a window a_k. According to the algorithm, there is no open leaf whose label is at most a_k. Also, by Lemma 5.3, there are at most $r_j - 1$ open leaves whose label is a_j for all $j > 0$. In particular for all $j \ge k$. Let d denote the maximum depth of any active tree. Then the total bandwidth of open leaves in all the first h opened trees is at most $\sum_{j=k+1}^{d}(r_j - 1)/a_j$ which is less than

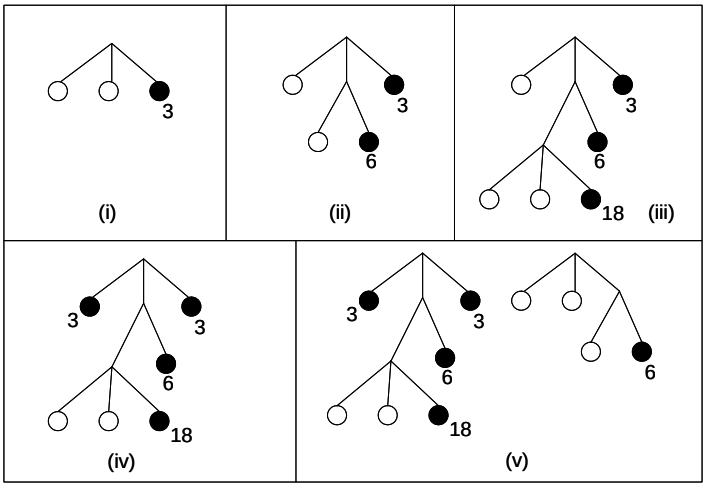

Fig. 4. Algorithm $\mathcal{W}_d$ evolution for $\sigma = \langle 3, 6, 18, 3, 6\rangle$. White circles denote open leaves.

$1/a_k$. To justify this bound consider a lace tree over the ratios $a_k, r_{k+1}, \ldots, r_d$. The total bandwidth of leaves in the sub-tree rooted at a node at depth 1 is $1/a_k$ and the sum $\sum_{j=k+1}^{d}(r_j - 1)/a_j$ is the total bandwidth of open leaves in this sub-tree excluding one node in the deepest level. Therefore, the total bandwidth required by the new $1/a_k$ request and the requests already scheduled is more than h. As a result, at the end, if the algorithm opened h trees then $H(\sigma) = h$. □

Note that there is no need to know in advance the ratios $\{r_i\}$, nor the number of requests, or their total width. The only a-priori knowledge required is that the input instance has divisible item sizes. Indeed, building a lace tree over the ratios r_1, r_2 is different from building a lace tree over the ratios r_2, r_1 (the former can accommodate future requests of window size r_1 while the latter cannot and similarly the latter will be able to accept requests of window size r_2 while the former cannot). However, the decisions about the number of children of the internal nodes of the lace tree can be delayed until the sequence actually gets a window of value r_1 or r_2.

5.2 An $H(\sigma) + O(\sqrt{H(\sigma)})$ Algorithm for Arbitrary Windows

For instances with arbitrary window size, we define a parameterized family of on-line algorithms $\mathcal{W}_k^*$ for $k = 1, 2, \ldots$ such that $N_{\mathcal{W}_k^*}(\sigma) \leq \frac{k+1}{k}H(\sigma) + k$ for any request sequence σ. The building blocks of our algorithm are optimal algorithms for divisible window sizes instances. In particular, for an odd constant integer $c \geq 1$ let $\mathcal{W}_c$ be an optimal online algorithm for an instance in which all window sizes are of the form $c2^{v_i}$ for integers $v_i \geq 0$.

Algorithm $\mathcal{W}_1^*$: Let w_i be the window of the next request to be scheduled. Round down w_i to the nearest power of 2 and apply algorithm $\mathcal{W}_1$ on the new and smaller window size.

Algorithm $\mathcal{W}_k^*$: Maintain k sets of channels: $C_1, \ldots, C_k$. On the channel-set C_j, schedule requests whose window sizes are rounded down to $(2j-1)2^{v_i}$. Let w_i be the window size of the next request to be scheduled. Round down w_i to the

nearest number of the form $c2^{v_i}$ where $c \in \{1, 3, \ldots, 2k-1\}$ and use algorithm $\mathcal{W}_c$ to schedule the rounded request on $C_{\frac{c+1}{2}}$.

The following is used to show that the new rounded down window sizes are not too small: For $i \geq 1$, define $I_i = \{(2i-1)2^v : v \geq 0\}$ and let $S_k = \cup_{i=1}^{k} I_i$. For example,

$$\begin{aligned} I_1 &= \{1, 2, 4, \ldots, 2^v, \ldots\} \\ I_2 &= \{3, 6, 12, \ldots, 3 \cdot 2^v, \ldots\} \\ I_3 &= \{5, 10, 20, \ldots, 5 \cdot 2^v, \ldots\} \\ S_3 &= \{1, 2, 3, 4, 5, 6, 8, 10, 12, 16, 20, 24, 32, 48, \ldots\} \ . \end{aligned}$$

CLAIM 5.5. *The set S_k contains the set $P_k = \{1, 2, \ldots, 2k\}$ and the sets $P_{k,v} = \{(k)2^v, (k+1)2^v, \ldots, (2k)2^v\}$ for any $k \geq 1$ and for all $v \geq 1$.*

PROOF. The proof is by induction on k. By definition, $S_1 = I_1$ is the set of powers of 2, and the lemma holds since $P_1 = \{1, 2\}$ and $P_{1,v} = \{2^v, 2^{v+1}\}$. Assume that the claim holds for $k \geq 1$ and consider S_{k+1}. By definition, $S_{k+1} = S_k \cup I_{k+1}$.

We first prove that $P_{k+1} \subseteq S_{k+1}$: (i) $P_k = \{1, 2, \ldots, 2k\}$ is a subset of S_{k+1} because $S_k \subseteq S_{k+1}$, (ii) $2k+1 \in S_{k+1}$ because $2k+1$ is the smallest member of I_{k+1}, and (iii) $2k+2 = (k+1)2 \in S_{k+1}$ because by the induction hypothesis, $(k+1)2 \in P_{k,1} \subseteq S_k$. As a result, $P_{k+1} = P_k \cup \{2k+1, 2k+2\} \subseteq S_{k+1}$.

Next we prove that $P_{k+1,v} \subseteq S_{k+1}$ for any $v \geq 1$: (i) $\{(k+1)2^v, \ldots, (2k)2^v\} \subseteq P_{k,v}$ is a subset of S_{k+1} because $P_{k,v} \subseteq S_k \subseteq S_{k+1}$, (ii) $(2k+1)2^v \in S_{k+1}$ because $(2k+1)2^v \in I_{k+1}$, and (iii) $(2k+2)2^v = (k+1)2^{v+1} \in S_{k+1}$ because by the induction hypothesis, $(k+1)2^{v+1} \in P_{k,v+1} \subseteq S_k$. As a result, $P_{k+1,v} = \{(k+1)2^v, (k+2)2^v, \ldots, (2k+2)2^v\} \subseteq S_{k+1}$. □

LEMMA 5.6. *For any positive integer w and any $k \geq 1$, there exists an integer $w' \leq w$ such that (i) $\frac{w}{w'} < \frac{k+1}{k}$, and (ii) there exist $c \in \{1, 3, \ldots, 2k-1\}$ and v such that $w' = c2^v$.*

PROOF. Let w be any positive integer and let w' be the largest member of S_k that is less than or equal to w. The ratio w/w' is bounded above by the largest ratio between consecutive members of S_k. By Claim 5.5, the ratios between consecutive members are less than or equal to the maximum of $(i+1)/i$ for $k \leq i < 2k$, which is maximized at $(k+1)/k$. Hence, $w/w' < (k+1)/k$. □

LEMMA 5.7. *For any sequence σ and $k > 0$,*

$$N_{\mathcal{W}_k^*}(\sigma) \leq \frac{k+1}{k} H(\sigma) + k \ .$$

PROOF. Let $\sigma = \langle w_1, w_2, \ldots, w_n \rangle$ and let $\sigma' = \langle w'_1, w'_2, \ldots, w'_n \rangle$ be the sequence of corresponding rounded windows. Let N_c denote the subset of indices of the requests whose windows are rounded to $c2^v$ for some integer $v \geq 0$. Let σ'_c be the subsequence of σ' whose indices are restricted to N_c. By Theorem 5.4, each algorithm $\mathcal{W}_c$ uses $H(\sigma'_c) = \left\lceil \sum_{i \in N_c} 1/w'_i \right\rceil \leq 1 + \sum_{i \in N_c} 1/w'_i$ channels to schedule σ'_c. Summing over all the k channel-sets, we get that all the requests are scheduled on at most $\sum_{i=1}^{n} 1/w'_i + k$ channels. By Lemma 5.6, $\sum_{i=1}^{n} 1/w'_i \leq \frac{k+1}{k} \sum_{i=1}^{n} 1/w_i$.

Thus,

$$N_{\mathcal{W}_k^*}(\sigma) \leq \frac{k}{k+1} H(\sigma) + k.$$

□

Let $h = \sqrt{H(\sigma)}$. For simplicity, assume that h is an integer. Otherwise, round h to the nearest integer and the analysis is similar. Assume first that $H(\sigma)$ is known in advance. The expression $\frac{k+1}{k} H(\sigma) + k$ is minimized for $k = h$. The following lemma gives the bound for $\mathcal{W}_h^*$. The proof is identical to the proof of Lemma 4.9.

LEMMA 5.8. *For any sequence σ, $N_{\mathcal{W}_h^*}(\sigma) \leq H(\sigma) + 2\sqrt{H(\sigma)}$.*

When $H(\sigma)$ is not known in advance, the algorithm dynamically increases the number of channel-sets (the parameter k). Algorithm $\mathcal{W}_{dyn}^*$ is defined as follows: Let H' denote the total bandwidth requirement of the already scheduled requests. As long as $H' \leq 1$, use $\mathcal{W}_1^*$. That is, all the windows are rounded down to the closest power of 2. When $1 < H' \leq 4$, shift to $\mathcal{W}_2^*$. That is, the windows are rounded down to the closest number of the form either 2^{v_i} or $3 \cdot 2^{v_j}$. In general, when $(k-1)^2 < H' \leq k^2$, use algorithm $\mathcal{W}_k^*$. That is, a window w_i is rounded down to the closest number of the form $c2^{v_i}$ where $c \in \{1, 3, \ldots, 2k-1\}$. Note that when the algorithm shifts to Algorithm $\mathcal{W}_k^*$, it continues to use the channel-sets that were used for $\mathcal{W}_{k-1}^*$, it just adds a new (initially empty) channel-set C_k.

THEOREM 5.9. *For any sequence σ,*

$$N_{\mathcal{W}_{dyn}^*}(\sigma) \leq H(\sigma) + 4\sqrt{H(\sigma)}\ .$$

PROOF. Let $\sigma = \langle w_1, w_2, \ldots, w_n \rangle$ and $h = \sqrt{H(\sigma)}$. That is, $\mathcal{W}_h^*$ is the last algorithm performed by $\mathcal{W}_{dyn}^*$ on this sequence. Recall that $h = \sqrt{H(\sigma)}$. Let $\sigma' = \langle w'_1, w'_2, \ldots, w'_n \rangle$ be the sequence of corresponding rounded windows. As in the proof of Lemma 5.7, it follows that $\mathcal{W}_{dyn}^*$ uses at most $h + \sum_{i=1}^{n} 1/w'_i$ channels. We now bound the bandwidth lost due to rounding. The idea is that, indeed, a prefix of σ has a smaller range of rounding possibilities, however, this loss is proportional to the total bandwidth request of the prefix. In other words, an additional rounding option is added, only after it is guaranteed that adding the corresponding potentially low-loaded channel still fits the competitive ratio.

Define s_k, for $1 \leq k \leq h$, to be the index of the first item scheduled while executing $\mathcal{W}_k^*$ and $s_{h+1} = n + 1$. It follows that algorithm $\mathcal{W}_k^*$ schedules the items of σ indexed s_k to $s_{k+1} - 1$. As in the proof of Theorem 4.10, the total bandwidth of the requests indexed from s_k to $s_{k+1} - 1$ is at most $2k$. By Lemma 5.6, when executing $\mathcal{W}_k^*$, the requests are rounded such that $w'_i \geq \frac{k}{k+1} w_i$. Thus,

$$\begin{aligned} \sum_{i=s_k}^{s_{k+1}-1} \frac{1}{w'_i} &\leq \sum_{i=s_k}^{s_{k+1}-1} \frac{k+1}{k w_i} = \frac{k+1}{k} \sum_{i=s_k}^{s_{k+1}-1} \frac{1}{w_i} \\ &\leq \frac{k+1}{k}(2k) = 2(k+1). \end{aligned}$$

Therefore, the total number of channels used by $\mathcal{W}^*_{dyn}$ is at most

$$\begin{aligned} h + \sum_{i=1}^{n} \frac{1}{w'_i} &= h + \sum_{k=1}^{h} \sum_{i=s_k}^{s_{k+1}-1} \frac{1}{w'_i} \\ &\leq h + \sum_{k=1}^{h} 2(k+1) = h^2 + 4h \\ &= H(\sigma) + 4\sqrt{H(\sigma)} \end{aligned}$$

□

6. OPEN PROBLEMS

In this paper we addressed the Unit Fractions Bin Packing (UFBP) problem and the Windows Scheduling (WS) problem in the off-line and the on-line settings. A summary of the results can be found in Table I in Section 1.3. The results demonstrate that UFBP is an "easier" problem than the traditional bin packing (BP) problem whereas WS as a restricted version of UFBP is "harder" than UFBP. We do not have matching bounds and it will be interesting to resolve the following open problems.

(1) We know that off-line WS is NP-hard using a compact representation. Is the problem NP-hard in the standard representation?
(2) For off-line UFBP, we know a solution which is optimal up to an additive term of 1. Is this problem NP-hard?
(3) Is there an off-line algorithm for WS that outperforms the solution of [4]? Also, does there exist a non-trivial lower bound, larger than $H(\sigma)+1$, for the off-line WS problem that separates it from the off-line UFBP problem?
(4) The upper bounds for on-line UFBP and WS are the same. Is there a better upper bound for on-line UFBP as is the case in the off-line setting?
(5) The only lower bound we have for on-line WS is the one for UFBP. Is there a larger lower bound for on-line WS, one that takes advantage of the additional restriction imposed by the WS problem?
(6) None of our algorithms for WS migrate requests from channel to another channel. Can migration help in the off-line or the on-line setting? Furthermore, if migration is permitted, is WS an NP-hard problem?

REFERENCES

[1] S. Acharya, M. J. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *IEEE Personal Communications*, 2(6):50–60, 1995.

[2] M. H. Ammar and J. W. Wong. The design of teletext broadcast cycles. *Performance Evaluation*, 5(4):235–242, 1985.

[3] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling. *Mathematics of Operations Research (MOR)*, 27(3):518–544, 2002.

[4] A. Bar-Noy and R. E. Ladner. Windows scheduling problems for broadcast systems. *SIAM Journal on Computing (SICOMP)*, 32(4):1091-1113, 2003.

[5] A. Bar-Noy, R. E. Ladner, and T. Tamir. Scheduling techniques for media-on-demand. In *Proceedings of the 14-th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 791–800, 2002.

[6] A. Bar-Noy, J. Naor, and B. Schieber. Pushing dependent data in clients-providers-servers systems. *Wireless Networks Journal (WINET)*, 9(5):175–186, 2003.

[7] M. Y. Chan and F. Chin. General schedulers for the pinwheel problem based on double-integer reduction *IEEE Transactions on Computers,* 41(6):755–768, 1992.

[8] W. T. Chan and P. W. H. Wong. On-line Windows Scheduling of Temporary Items, *Proc. of the 15th International Symposium on Algorithms and Computation (ISAAC)*, 259-270, 2004.

[9] E. G. Coffman, C. A. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor, R. R. Weber, and M. Yannakakis. Bin packing with discrete item sizes, part I: perfect packing theorems and the average case behavior of optimal packings. *SIAM Journal on Discrete Mathematics*, 13(3):384–402, 2000.

[10] E. G. Coffman, M. R. Garey, and D. S. Johnson. Bin packing with divisible item sizes. *Journal of Complexity*, 3(4):406–428, 1987.

[11] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. Approximation Algorithms for NP-Hard Problems, D. Hochbaum (editor), PWS Publishing, Boston (1996), 46–93.

[12] L. Engebretsen and M. Sudan. Harmonic broadcasting is optimal. In *Proceedings of the 13-th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 431–432, 2002.

[13] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman, 1979.

[14] V. Gondhalekar, R. Jain, and J. Werth. Scheduling on airdisks: efficient access to personalized information services via periodic wireless data broadcast. *IEEE International Conference on Communications (ICC)*, 3:1276–1280, 1997.

[15] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A a real-time scheduling problem. In *Proceedings of the 22-nd Hawaii International Conference of System Science,* 693–702, 1989.

[16] K. A. Hua and S. Sheu. An efficient periodic broadcast technique for digital video libraries. *Multimedia Tools and Applications*, 10(2/3):157–177, 2000.

[17] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithm. *SIAM Journal on Computing (SICOMP)*, 3(4):299–325, 1974.

[18] L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–271, 1997.

[19] V. Kann. Maximum bounded 3-dimensional matching is max SNP-complete. *Information Processing Letters (IPL)*, 37(1):27–35, 1991.

[20] C. C. Lee and D. T. Lee. A simple on-line bin packing algorithm. *Journal of the ACM (JACM)*, 32:562-572, 1985.

[21] C. L. Liu and W. Laylend. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

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

[23] R. Tijdeman. The chairman assignment problem. *Discrete Mathematics*, 32(3):323–330, 1980.

[24] A. van Vliet. On the asymptotic worst case behavior of harmonic fit. *Journal of Algorithms*, 20(1):113–136, 1996.

[25] W. F. Vega and G. S. Leuker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[26] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *ACM Multimedia Systems Journal*, 4(3):197–208, 1996.

[27] P. W. H. Wong, W. T. Chan, and T. W. Lam. Dynamic Bin Packing of Unit Fractions Items. *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, 2005.