Windows Scheduling as a Restricted Version of Bin Packing

Amotz Bar-Noy* Richard E. Ladner† Tami Tamir†

Abstract

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 the 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 obvious bandwidth lower bound on the required number of bins (channels). Previously an $H + O(\ln H)$ off-line algorithm for the windows scheduling problem was known. This paper presents an $H + 1$ off-line algorithm to the unit fractions bin packing problem. In the on-line setting, this paper presents an $H + O(\sqrt{H})$ algorithm to both problems 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.

*Computer & Information Science Department, Brooklyn College, 2900 Bedford Ave., Brooklyn, NY 11210. amotz@sci.brooklyn.cuny.edu

†Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195. {ladner, tami}@cs.washington.edu.

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 minimum 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 set of windows $\{2, 3, 6\}$. Since $1/2 + 1/3 + 1/6 = 1$, the three items can be packed in one bin. On the other hand, there is no windows-schedule on one channel of these pages 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 the channel 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 goal 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 get better performance than the algorithms known for BP. On the other hand because WS without migrations is a restriction of UFBP, we expect algorithms for WS to have worse performance than those 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 times of 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 Performance Analysis.

Given a sequence σ of item widths $\langle 1/w_1, \ldots, 1/w_n \rangle$ and an 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, for all i, w_i is an integer.

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 element at a time and the algorithm must augment its current solution to accommodate a new element. That is, a decision that is made about which bin to pack an item in UFBP or how to schedule a request on the channels in WS, cannot be revoked once done.

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 elements in σ. Since the number of bins in UFBP and the number of channels in WS must be an integer,

$$H(\sigma) = \left\lceil \sum_{i=1}^{n}(1/w_i) \right\rceil \tag{1.1}$$

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

At present we do not know if the problem of finding the minimum number of bins in UFBP is NP-hard, but we do know that the associated restricted form of WS is NP-hard as shown below. Thus, we seek "good" *approximation* algorithms for the off-line UFBP and WS problems and "good" *competitive* algorithms for their on-line versions. Generally, we express the bounds on the performance of an algorithm $\mathcal{A}$ in the form

$$H(\sigma) \le N_{\mathcal{A}}(\sigma) \le H(\sigma) + f(H(\sigma)) \tag{1.2}$$

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 (1.2). Since f is non-decreasing and $H(\sigma)$ is a lower bound on $N_{\mathsf{OPT}}(\sigma)$ we have:

$$N_{\mathsf{OPT}}(\sigma) \le N_{\mathcal{A}}(\sigma) \le N_{\mathsf{OPT}}(\sigma) + f(N_{\mathsf{OPT}}(\sigma)) . \tag{1.3}$$

Hence,

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

This ratio can be interesting, but does not yield as much information as inequalities (1.2) and (1.3). Consequently, we will usually express the performance of the algorithms in the paper in the form of the right inequalities of (1.2) and (1.3).

1.2 Related Work.

There is a wide literature on the general bin packing problem, see the survey [9]. First, bin packing is an NP-hard problem [11]. For the off-line problem, there exists an asymptotic PTAS that uses $(1+\varepsilon)N_{\mathsf{OPT}}(\sigma) + 1$ bins [21]. The performance of the on-line algorithms *first-fit* (FF) and *best-fit* (BF) is analyzed in [14], 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 $\rho(\mathcal{A}) \geq 1.540$ [20]. The best known on-line bin packing algorithm is *Harmonic++* whose competitive ratio is 1.589 [18]. In [8], the special case of BP with *divisible item sizes* (where in the sorted sequence of sizes $a_1 < a_2 < \cdots$, for all i, a_i divides a_{i-1}) is shown to be optimally solvable with a polynomial time algorithm. Bin packing with discrete item sizes, i.e., when items sizes are in $\{1/k, 2/k, \ldots, j/k\}$ for some $1 \leq j \leq k$ is considered in [7].

The windows scheduling problem was first defined in [4]. That 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.

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., [15, 13]). This concept of receiving from multiple channels and buffering data for future playback was first developed by [22]. 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 [10].

Windows scheduling can be thought of as a scheduling problem for push 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 response time). In a more generalized model, the servers are not the information providers (e.g., [12, 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 [17] and the chairman assignment problem [19]. 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 UFBP is a relaxed version of both optimization goals.

1.3 Summary of Results. In Section 2, we prove the NP-hardness of WS without migrations. 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. The off-line WS problem has polynomial time algorithm [4] that uses $H(\sigma) + O(\ln(H(\sigma))$ channels. By contrast, in Section 3, we show the *any-fit decreasing* is a polynomial time algorithm for off-line UFBP that uses $H(\sigma) + 1$ bins. So it appears that UFBP is "easier" than WS in the off-line setting.

In Section 4 we consider on-line algorithms for UFBP. 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 problem. Next, we give a tight analysis of the natural algorithms next-fit and any-fit. Finally, we give a polynomial time algorithm that uses $H(\sigma) + O(\sqrt{H(\sigma)})$ bins.

in Section 5 we consider on-line algorithms for WS. First, we give a non-trivial algorithm that uses

	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 1: Results for UFBP and WS in the off-line and the on-line settings.

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 [8]). 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 1 summarizes the results for UFBP and WS in the off-line and on-line settings. Our results are marked with [$\star$]. Due to space limitations some of the proofs are omitted and some are sketched.

2 NP-hardness

We show that window scheduling without migration is NP-hard. That is, the problem is NP-Hard when broadcasts of a page have to be from the same channel. We do not know if the problem is still NP-hard when migration is permitted. A previous hardness proof for WS ([3]) works 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 is simpler, holds for arbitrary number of channels, and covers the case in which gaps between schedules of request i may vary.

Theorem 2.1. *Windows scheduling without migration is NP-hard.* □

For the UFBP problem, migration makes the problem trivial since it means that items can 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 $1/k$ for arbitrary k.

3 Off-line UFBP

We present a polynomial time algorithm for UFBP which is optimal up to an additive term of 1.

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 if such exists. Otherwise, the current item is packed in a new opened bin.

We are not specific about which bin to pack an item because our analysis suits any bin selection (e.g, first-fit or best-fit).

Theorem 3.1. *For any sequence σ,*

$$N_{\mathsf{AFD}}(\sigma) \le H(\sigma) + 1.$$

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

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

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

Claim 3.1. *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 $1/w'$ where $w' \le w$. By Claim 3.1, at this stage, there are less than w' non-full bins. Since this is the last opened bin, it follows that $h' < w'$. Furthermore, each of the first $h'-1$ non-full bins must contain items whose total width is greater than $1-(1/w')$, because otherwise AFD would not open a new bin for $1/w'$. By definition, the last non-full bin contains one item of width $1/w'$. It follows that

$$H(\sigma) \geq h + (h'-1)\left(1 - \frac{1}{w'}\right) + \frac{1}{w'}$$
$$= h + h' - 1 - \frac{h'-2}{w'} > h + h' - 2 \ .$$

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 $N_{\mathsf{AFD}}(\sigma) \leq N_{\mathsf{OPTB}}(\sigma) + 1$. The above analysis is tight, as demonstrated by the non-increasing sequence $\sigma = \langle \frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4} \rangle$. The optimal solution uses two bins, the first contains $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{4}\}$ and the second contains $\{\frac{1}{3}, \frac{1}{3}, \frac{1}{4}\}$. 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, $N_{\mathsf{AFD}}(\sigma) \geq N_{\mathsf{OPTB}}(\sigma) + 1$.

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 $2H(\sigma)$ (when σ consists of items of width $1/2 + \varepsilon$).

4 On-line UFBP

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 4.1. $H(\sigma) \geq h_0$. □

Claim 4.2. *The total free space in all of the bins of $\mathcal{B}$ is at least* $\Theta(\ln(w))$. □

Claim 4.3. $N_{\mathcal{B}}(\sigma) < w^2 \ln(w)$. □

Consequently, for this sequence, the total item width is at most $N_{\mathcal{B}}(\sigma) - \Theta(\ln(w)) \leq w^2 \ln(w) - \Theta(\ln(w))$, and thus, $H(\sigma) \leq w^2 \ln(w) - \Theta(\ln(w))$. Since $\ln(N_{\mathcal{B}}(\sigma)) \leq \ln(w^2 \ln(w)) = \Theta(\ln(w))$, it follows that $N_{\mathcal{B}}(\sigma) \geq H(\sigma) + \Omega(\ln(H(\sigma)))$. □

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

Corollary 4.1. *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 Any-fit and Next-Fit. In this section we consider simple on-line algorithms for UFBP. In particular, we analyze the performance of the *Next-fit* and the *Any-fit* algorithms 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.

Any Fit (AF)**:** An item is packed in any one of the bins that has enough free space for it.

In fact, *Any fit* is a class of algorithms, differ by the way the actual bin in which the item is packed is selected. In *First-fit* the item is packed in the first bin that has enough free space for its width. Other Any-fit algorithms might prefer the *fullest* or the *emptiest* bin that can accommodate the new item. Our analysis shows that for the UFBP problem all AF algorithms have the same worst-case performance regardless of the selection rule.

The following results state the exact competitive ratios of NF and AF. That is, for $\mathcal{A} \in \{\mathsf{NF}, \mathsf{AF}\}$, (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 4.2. $\rho(\mathsf{NF}) = 2$. □

THEOREM 4.3. $\rho(\mathsf{AF}) = \frac{6}{5}$.

Proof. We first show $\rho(\mathsf{AF}) \geq \frac{6}{5}$. 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. AF 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 AF is at least $\frac{6x}{5x} = \frac{6}{5}$. For the upper bound, assume that there exists a sequence σ of items for which $\rho_\sigma(\mathsf{AF}) > (6/5)$. Then there exists a bin whose capacity is less than $5/6$ and a point in the sequence after which the width of any item is greater than $1/6$. That is, after this point, a width could have only the values: $1/2, 1/3, 1/4, 1/5$. Since no linear combination of these 4 values totals a value that is less than $5/6$ and greater than $4/5$, it follows that there is no additional bin whose capacity is less than $5/6$. Therefore, AF allocates at most $(6/5)H(S) + 1$ bins to the sequence σ. Thus, for any $\varepsilon > 0$ and for any long enough sequences $\rho(\mathsf{AF}) \leq \frac{6}{5} + \varepsilon$. □

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.

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

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 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.

LEMMA 4.1. *For any sequence* σ *and* $k > 0$,

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

Proof. 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 capacity could be small, and the total number of bins used is

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

□

Let $h = \sqrt{H(\sigma)}$. For simplicity, we assume that h is an integer. Otherwise, we round h to the nearest integer, 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 4.2. *For any sequence σ,*

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

When $H(\sigma)$ is not known in advance, we dynamically increase 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 $H' \le 1$, use $\mathcal{B}^*_1$ (regular first-fit), when $1 < H' \le 4$, shift to $\mathcal{B}^*_2$, and in general, when $(k-1)^2 < H' \le k^2$, 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 4.4. *For any sequence σ,*

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

Proof. Let s_k denote the subset of σ of items that were packed while executing $\mathcal{B}^*_k$. It follows that the total width of items in s_1 is less than $1+\frac{1}{2}$, the total width of items in s_2 is less than $2^2 + \frac{1}{2} - 1^2 = 3$, and in general, the total size of items in s_k 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 4.1, 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. 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 capacity (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

$$\begin{aligned} & h + \sum_{k=1}^{h} \frac{k+1}{k} \sum_{i \in s_k} \frac{1}{w_i} < h + \sum_{k=1}^{h} \frac{k+1}{k} 2k \\ = \quad & h + \sum_{k=1}^{h} 2k + \sum_{k=1}^{h} 2 = h^2 + 4h = H + 4\sqrt{H}. \end{aligned}$$

$\square$

5 On-line WS

In this Section we consider on-line algorithms for the WS problem. We start with a set of optimal algorithms for specific instances. These algorithms form the building blocks of our concluding algorithm, $\mathcal{W}^*_{dyn}$ that has the same performance as Algorithm $\mathcal{B}^*_{dyn}$ for UFBP.

5.1 An Optimal Algorithm for Powers of 2 Windows.

Assume that the sequence σ contains only windows that are powers of 2. That is, $w_i = 2^{v_i}$ for an integer $v_i \ge 1$ for $1 \le i \le n$.

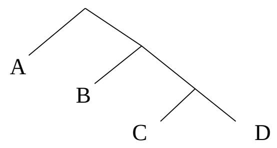

Figure 1: Tree representation of the cyclic schedule $[A, B, A, C, A, B, A, D]$.

We represent each channel schedule by a binary tree, in which all the internal nodes have degree 2. The tree leaves represent the scheduled items. To construct the schedule from a tree, alternate between 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 binary trees, each representing one channel.

We denote by *open* a leaf that is not assigned yet to any request, and by *active* a tree that is not fully utilized yet, that is, a tree with at least one open leaf, representing 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.

Algorithm $\mathcal{W}_1$: Let $w_i = 2^{v_i}$ be the next request in σ. If there is an open leaf whose label is 2^{v_i}, schedule the request on that leaf. Otherwise, let $u_i < v_i$ be the minimal such that an open leaf whose label is 2^{u_i} exists. If no such u_i exists, open a new active tree with one open leaf whose label is $2^0 = 1$. Let ℓ

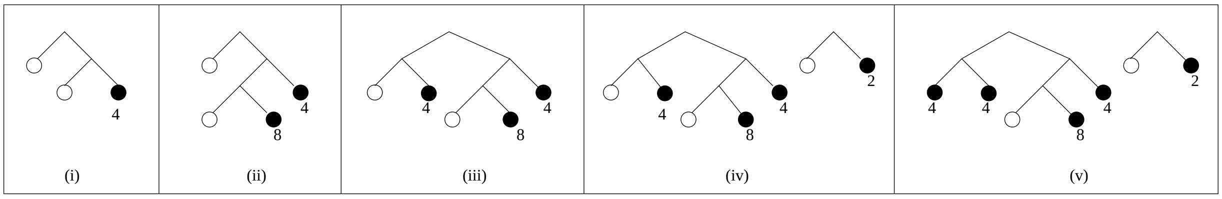

Figure 2: Algorithm evolution for $\sigma = \langle 4, 8, 4, 2, 4 \rangle$. White circles denote open leaves

be the selected leaf and let T be the tree containing the leaf ℓ. Append to T a lace binary sub-tree of depth $v_i - u_i$ whose root is ℓ and schedule request i on one of the two leaves with depth v_i in T. See an example in Figure 2.

LEMMA 5.1. *During the execution of $\mathcal{W}_1$, the forest 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 is only one open leaf in depth zero (the root of the first tree). When scheduling a new request, $\mathcal{W}_1$ either closes an open leaf or replaces an open leaf with a lace tree. Since this replacement is performed on the lowest open leaf, there are no open leaves in lower depths. By the definition of a lace binary tree, each added leaf is the single one in its depth. Finally, there are two leaves in the lowest level but one of them is allocated to the new request. $\square$

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

Proof. 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) $\sum_{i\in\sigma} 1/w_i$ 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 $j > 0$. Let d denote the maximum depth of any active tree. Then the total bandwidth of the open leaves in all the first h opened trees is at most $\sum_{j=1}^{d} 1/2^{v_i+j}$ 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. $\square$

5.2 An Optimal Algorithm for Windows of the form $c2^{v_i}$. We generalize algorithm $\mathcal{W}_1$ to work for instances in which there exists an integer c such that all the w_i's are of the form $c2^{v_i}$ for an integer $v_i \geq 0$ for $1 \leq i \leq n$. We note that in a similar way, we can construct optimal algorithms for any instance with divisible window sizes. In such sequences, there exists constants c_1, c_2, such that for all $1 \leq i \leq n$, window i is of size $c_1 c_2^{v_i}$ for some integer v_i. Details are omitted since we do not use this fact in the rest of the paper.

Let $c \geq 1$ be an odd number. In Algorithm $\mathcal{W}_c$, each channel schedule is represented by a tree whose root degree is c, and each of the c subtrees is a binary tree. To construct the schedule from a tree, in a round-robin fashion, schedule a request from each of the c binary subtrees. In each binary subtree the selection of the next request is done as described in Section 5.1. It follows that a leaf, ℓ, whose depth in the tree is $d(\ell)$ represents an item scheduled with window size $c2^{d(\ell)-1}$. The whole schedule is represented by a forest of such trees, each representing one channel. Let the label of a leaf ℓ of depth $d(\ell)$ be $c2^{d(\ell)-1}$.

Algorithm $\mathcal{W}_c$: Let $w_i = 2^{v_i}$ be the next request in σ. If there is an open leaf whose label is $c2^{v_i}$, schedule the request on that leaf. Otherwise, let $u_i < v_i$ be the minimal such that an open leaf whose label is $c2^{u_i}$ exists. If no such u_i exists, open a new active tree with c open leaves each with a label c. Let ℓ be the selected leaf and let T be the tree containing the leaf ℓ. Append to T a lace binary sub-tree of depth $v_i - u_i$ whose root is ℓ and schedule request i on one of the two leaves with depth $v_i + 1$ in T.

The proof of the next lemma is a generalization of the proof of Lemma 5.2.

LEMMA 5.3. *For any sequence σ in which for all $i, w_i = c2^{v_i}$ we have $N_{\mathcal{W}_c}(\sigma) = H(\sigma)$.* $\square$

5.3 An $H(\sigma)+O(\sqrt{H(\sigma)})$ Algorithm for Arbitrary Windows. For arbitrary windows, 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.

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 resulting request.

Algorithm $\mathcal{W}_k^*$: Maintain k sets of channels: $C_1, \ldots, C_k$. On the channel-set C_j, schedule requests whose window is rounded down to $(2j-1)2^{v_i}$. Let w_i be the window 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 $C_{\frac{c+1}{2}}$.

Lemma 5.4. *For any sequence σ and $k > 0$,*

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

Proof. We first show that when Algorithm $\mathcal{W}_k^*$ is applied, a request with window w is rounded to a request with window w' such that w' is "close" to w. Then we analyze the performance of $\mathcal{W}_k^*$ on the rounded windows.

Claim 5.1. *For any integer w and any $k \geq 0$, there exists an integer $w' < w$ such that (i) $\frac{w}{w'} \leq \frac{k+1}{k}$, and (ii) $\exists c \in \{1, 3, \ldots, 2k-1\}$ and $\exists v$ such that $w' = c2^v$.* □

Let w'_i denote the rounded window of request i. Let σ' denote the sequence of the rounded windows, and let σ'_c denote the subset of σ' of the requests whose windows are rounded to $c2^v$ for some integer $v \geq 0$. By Lemma 5.3, Algorithms $\mathcal{W}_c$ uses $H(\sigma'_c) = \left\lceil \sum_{i \in \sigma'_c} 1/w'_i \right\rceil \leq 1 + \sum_{i \in \sigma'_c} 1/w'_i$ channels to schedule all the requests in σ'_c. Summing over all the k channel-sets, we get that all the requests are scheduled on at most $\sum_{i \in \sigma'} 1/w'_i + k$ channels. By Claim 5.1 $\sum_{i \in \sigma'} 1/w'_i \leq \frac{k+1}{k} \sum_{i \in \sigma} 1/w_i$. Thus, $N_{\mathcal{W}_k^*}(\sigma) \leq \frac{k}{k+1} H(\sigma) + k$. □

Let $h = \sqrt{H(\sigma)}$. 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 $\mathcal{W}_h^*$.

Lemma 5.5. *For any sequence σ,*

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

When $H(\sigma)$ is not known in advance, we dynamically increase 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 5.1. *For any sequence σ,*

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

Proof. Recall that $h = \sqrt{H(\sigma)}$. That is, Algorithm $\mathcal{W}_h^*$ is the last algorithm executed by $\mathcal{W}_{dyn}^*$. Let w'_i denote the rounded window of request i. Let σ' denote the sequence of the rounded windows. As in the proof of Lemma 5.4, we have that $\mathcal{W}_{dyn}^*$ uses at most $h + \sum_{i \in \sigma'} 1/w'_i$ channels. We now bound the bandwidth lost due to rounding. The idea is that, indeed, prefixes of σ has a smaller range of rounding possibilities, however, this loss is proportional to the total bandwidth request of the prefix. In particular, the bulk of the requests has all the h rounding possibilities .

Let s_c denote the subset of σ of requests arriving while executing $\mathcal{W}_c^*$. As in the proof of Theorem 4.4, we have that the total bandwidth request of s_k is at most $2k$. By Claim 5.1, when executing $\mathcal{W}_k^*$, we round the requests such that $w'_i \geq \frac{k}{k+1} w_i$. Thus,

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

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

$$h + \sum_{i \in \sigma'} \frac{1}{w'_i} = h + \sum_{k=1}^{h} 2(k+1) = h^2 + 4h = H + 4\sqrt{H}$$

□

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 1 in Section 1.3. The following problems remain open.

1. For off-line UFBP, we know a solution which is optimal up to an additive term of 1. Is this problem NP-hard?

2. 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?

3. 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?

4. 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?

5. All of our algorithm for WS do not 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 Comm.*, 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. In *Proc. of the 13-th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 433–442, 2002.

[5] A. Bar-Noy, R. E. Ladner, and T. Tamir. Scheduling techniques for media-on-demand. In *Proc. 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*, 9(5):175–186, 2003.

[7] 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 J. Discrete Math.*, 13:384–402, 2000.

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

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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 J. of Comput.*, 3:256–278, 1974.

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

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

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

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

[19] R. Tijdeman. The chairman assignment problem. *Discrete Mathematics*, 32:323–330, 1980.

[20] A. van Vliet. On the asymptotic worst case behavior of harmonic fit. *J. of Algs.*, 20:113–136, 1996.

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

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