On-line Windows Scheduling of Temporary Items

Wun-Tat Chan[1]⋆ and Prudence W.H. Wong[2]

[1] Department of Computer Science, University of Hong Kong, Hong Kong
wtchan@cs.hku.hk
[2] Department of Computer Science, University of Liverpool, UK
pwong@csc.liv.ac.uk

Abstract. In this paper we study on-line windows scheduling (WS) of temporary items. In a broadcasting system, there are some broadcast channels, each can broadcast one item in one time unit. Upon the arrival of an item with a window w, where w is a positive integer, the item has to be scheduled on a channel such that it will be broadcasted in the channel at least once every w time units until its departure. With the on-line input of these temporary items, the problem is to minimize the maximum number of channels used over all time. We give a 5-competitive on-line algorithm and show that there is a lower bound of $2-\epsilon$ for any $\epsilon > 0$ on the competitive ratio of any on-line algorithm.

1 Introduction

In this paper we study on-line windows scheduling (WS) [3, 4] of temporary items (i.e., items may depart). We are the first to consider temporary items in the problem. A temporary item may have *unknown duration*, i.e., the item departs at unpredictable time; or *known duration*, i.e., the departure time is known when the item arrives. In a broadcasting system, there are some broadcast channels, each can broadcast one item in one time unit. An item i comes with a window w_i, where w_i is a positive integer. When item i arrives, it has to be scheduled on one of the channels such that it will be broadcasted at least once every w_i time units. Reschedule of the item to other channels is not allowed. For example, suppose two items with windows equals to 2 and 3, respectively, arrive one after the other. Then we can schedule them in the same channel with the broadcast sequence $\langle 2, 3\rangle$ or $\langle 2, 2, 3\rangle$ repeatedly. The objective of the problem is to minimize the maximum number of channels used over all time.

WS can be applied to push-based broadcasting systems. In a push-based system, servers broadcast pages on broadcast channels to clients. Clients who wish to receive the pages will listen to the channels and select what information they want to receive. If the quality of service is measured by the average response time [1, 2], servers broadcast more popular pages more frequently. Yet, in some business model, the servers sell their service to information providers who request

⋆ This research was supported in part by Hong Kong RGC Grant HKU-5172/03E.

the pages to be broadcasted regularly [5, 11]. The frequency of broadcasting a page is proportional to the amount paid by the provider. The servers may receive additional requests by providers over time, while a provider may withdraw from the service at any time; this is modeled by temporary items. Another related application is video-on-demand systems: many requests of a popular movie are received over a short period of time. To reduce the server bandwidth requirement without sacrificing the response time, the pyramid broadcasting paradigm [14, 15] and its variants [12, 13] are adopted. The movie is partitioned into equal size segments, each can be broadcasted in one time unit, and segment i is broadcasted every i time units. Then any client can view the movie uninterruptedly by waiting for at most one time unit.

Related work: WS with permanent items (i.e., the items never depart) was first studied by Bar-Noy and Ladner [3], where they gave an off-line approximation algorithm which uses $H+O(\ln H)$ channels, for $H=\sum_i 1/w_i$. The NP-hardness of the problem was proved later [4]. An on-line algorithm was also given which uses $H+O(\sqrt{H})$ channels [4].

As pointed out in [4], WS is a restricted version of a bin packing problem, called the unit fraction bin packing (UFBP). UFBP is to pack all items, each with a width $1/w_i$ for some positive integer w_i, into minimum number of unit-capacity bins. It can be observed in the following that WS is a restricted version of UFBP. Suppose we have three items with w_i equal to 2, 3, and 6. They can be packed in the same bin because $1/2+1/3+1/6=1$. Yet we cannot schedule to broadcast them in the same channel [8]. It has been shown in the pinwheel problem[3] [7, 8, 10] that if the sum of $1/w_i$ is less than or equal to 3/4, the items can be scheduled on one channel [10], otherwise, it might not be possible to do so. Bar-Noy et al. [4] gave an off-line algorithm for UFBP that uses $H+1$ bins, where $H=\sum_i 1/w_i$. They also gave an on-line algorithm that uses $H+O(\sqrt{H})$ bins and a lower bound of $H+\Omega(\ln H)$.

Coffman et al. [9] has studied the unit-capacity bin packing problem with temporary items, where the item width can be any real number less than or equal to 1. When the largest item width is 1, they showed that first-fit is 2.897-competitive and no on-line algorithm is better than 2.5-competitive. When the largest item width is less than $1/k$ for an integer k, they gave a general upper bound in terms of k for first-fit and a general lower bound in terms of k for any on-line algorithm. Note that the upper bound of UFBP with temporary items follows from the upper bound results of Coffman et al. but the lower bound does not.

Our contribution: Similar to other study on on-line problems, we measure the performance of an on-line algorithm in terms of competitive ratio (see [6] for a survey). In this paper we give the first upper and lower bounds on the competitive ratio of WS with temporary items. We observe that the competitive ratios depend on a value α which is the minimum of w_i. Precisely, we show that any

[3] The pinwheel problem is to determine whether the given items can be scheduled in one single channel and give the schedule if the answer is yes.

on-line algorithm has a competitive ratio at least $1 + 1/\alpha - \epsilon$ for any $\epsilon > 0$. In contrast to the case of permanent items in which there is an algorithm that uses at most $H + O(\sqrt{H})$ channels, the $1 + 1/\alpha - \epsilon$ lower bound demonstrates that WS with temporary items is "harder" than WS with permanent items. This lower bound applies to both known and unknown duration, yet the adversary for unknown duration is simpler as expected. As for upper bound, we give an on-line algorithm $\mathcal{W}$ that is 5-competitive when $\alpha = 1$, and $(2+2/(\alpha^* - 1))$-competitive when $\alpha \geq 2$, where α^* is the largest power of 2 that is smaller than or equal to α. These upper bounds also hold for both known and unknown duration.

Organization of the paper: The rest of the paper is organized as follows. In Section 2, we give some preliminaries. In Section 3, we present the lower bound for UFBP and WS. The lower bound for WS is based on the lower bound for UFBP. In Section 4, we present the on-line algorithm $\mathcal{W}$ for WS and analyze its performance. Finally, we give a conclusion in Section 5.

2 Preliminaries

In this section, we give a precise definition of the on-line problems of UFBP and WS with temporary items, and the necessary notations for further discussion.

On-line UFBP: The input is a sequence σ of items. Item i is represented as a 3-tuple $(a_i, d_i, 1/w_i)$, where a_i, d_i, and w_i are positive integers with a_i denoting the arrival time, d_i the departure time, and $1/w_i$ the width of item i. For unknown duration, d_i is not specified when item i arrives. All items are to be packed in unit-capacity bins. The objective of the problem is to minimize the maximum number of bins used over all time.

On-line WS: The input is a sequence σ of items for broadcast. Item i is represented as a 3-tuple (a_i, d_i, w_i), where a_i, d_i, and w_i are positive integers denoting the arrival time, the departure time, and the window of item i, respectively. We assume that $d_i \geq a_i + w_i - 1$, i.e., an item that arrives should be broadcasted at least once. For unknown duration, d_i is not specified when item i arrives. Each item takes one unit of time to broadcast, i.e., if an item is broadcasted in the beginning of time t, the client can receive the item at the end of time t. Item i must be scheduled on a broadcast channel between the time interval $[a_i, d_i]$ such that for any $a_i \leq t_1 \leq t_2 \leq d_i$ and $t_2 - t_1 + 1 \geq w_i$, item i should be broadcasted at least once in the sub-interval $[t_1, t_2]$. Note that in this definition, the first broadcast of item i must be within the time interval $[a_i, a_i + w_i - 1]$, i.e., there are at most $w_i - 1$ time units allowed between the arrival time and the first broadcast of an item. Similarly, we assume that in the unknown duration case, item i should announce its departure at or before time $d_i - w_i + 1$. The objective of the problem is to minimize the maximum number of channels used over all time. Define the *load* of a channel at time t to be the sum of $1/w_i$ for all item i scheduled in the channel that have arrived but not yet departed.

Both the above problems are on-line problems in the sense that at any time t, there is no information on the items arriving after t. Given a WS (UFBP, resp.)

algorithm $\mathcal{W}$, we analyze its performance using competitive analysis. Given a sequence σ of items, let $\mathcal{W}(\sigma, t)$ be the number of channels (bins, resp.) used by $\mathcal{W}$ at time t. We say that $\mathcal{W}$ is c-competitive if there exists a constant k such that for any input sequence σ, we have

$$\max_t \mathcal{W}(\sigma, t) \leq c \cdot \max_t \mathcal{O}(\sigma, t) + k,$$

where $\mathcal{O}$ is the optimal off-line algorithm of WS (UFBP, resp.)

Remarks: Notice that any on-line algorithm for WS with unknown duration always maintains the load of every channel to be at most 1 because it has to guarantee that all items scheduled on a channel can be broadcasted within their windows (even if none of the items depart). Therefore, we also restrict our attention to those off-line algorithms that maintain, at any time, the load of every channel to be at most 1.

3 Lower bound results

In this section we present lower bounds for UFBP and WS with temporary items. The lower bound for UFBP is easier to construct, based on which we can construct the lower bound for WS. Precisely, we show that for both UFBP and WS, no on-line algorithm is better than $(1 + 1/\alpha - \epsilon)$-competitive, for any $\epsilon > 0$ and $\alpha = \min_i\{w_i\}$, this is true for both known and unknown duration. Yet the adversary for known duration is more complicated as expected and is based on the adversary for unknown duration. We will give the details of the lower bound for UFBP, the lower bound for WS can be proved similarly and will be outlined at the end of this section.

Lower bound for UFBP We first consider the case of unknown duration. Given any on-line algorithm $\mathcal{A}$, we construct a sequence of items of widths either $1/y$ or $1/\alpha$ for some y and α such that y is much greater than α. The adversary works in three stages as follows.

1. Items with $w_i = y$ are released at time $a_i = i$ until at least y bins are used by $\mathcal{A}$ and the number of items m released is a multiple of y. Consider the minimum such m.
2. The adversary retains one item in each of any y occupied bins and let all the other $m - y$ items depart.
3. Finally, $\alpha(m - y)/y$ items with $w_i = \alpha$ are released.

Notice that at most y^2 items with $w_i = y$ are sufficient to force the on-line algorithm to use at least y bins in Stage 1. Thus, $m \leq y^2$. The following lemma gives a lower bound on the number of bins used by the on-line algorithm after Stage 3.

Lemma 1. *$\mathcal{A}$ uses at least $y + \max\{0, (m - y)/y - y(\alpha - 1)/\alpha\}$ bins.*

Proof. Notice that after Stage 2, at most $\alpha - 1$ items of width $1/\alpha$ can be packed in each of the y occupied bins. The total width of the items released in Stage 3 is equal to $(m-y)/y$. Therefore, if $(m-y)/y > y(\alpha-1)/\alpha$, $\mathcal{A}$ needs at least $(m-y)/y - y(\alpha-1)/\alpha$ additional new bins to pack all items released in Stage 3. Hence, the number of bins used is at least $y + \max\{0, (m-y)/y - y(\alpha-1)/\alpha\}$, and the lemma follows. $\square$

The following theorem shows the lower bound on the competitive ratio for UFBP with unknown duration. Roughly speaking, we establish the lower bound by showing that there is an off-line algorithm that uses m/y bins at any time and then compare the number of bins used by the on-line algorithm with m/y.

Theorem 1. *Any on-line algorithm for UFBP with unknown duration is at least* $(1 + 1/\alpha - \epsilon)$*-competitive for any* $\epsilon > 0$ *and* $\alpha = \min_i\{w_i\}$.

Proof. First we show that there is an off-line algorithm $\mathcal{O}$ that uses m/y bins at any time. In Stage 1, $\mathcal{O}$ packs the y non-departing items (i.e., those remain after Stage 2) in the same bin and the other $m-y$ items in another $(m-y)/y$ bins. In Stage 3, $\alpha(m-y)/y$ items of width $1/\alpha$ are released. These items are packed by $\mathcal{O}$ into the $(m-y)/y$ bins that become empty after Stage 2. Thus, the number of bins used by $\mathcal{O}$ at any time is $1 + (m-y)/y = m/y$.

Then, we analyze the number of bins used by any on-line algorithm $\mathcal{A}$. There are two cases to consider. (1) Suppose that $(m-y)/y \le y(\alpha-1)/\alpha$. By Lemma 1, no new bins are needed in Stage 3 and the maximum number of bins used by $\mathcal{A}$ at any time is y. By simple mathematics, the inequality can be rearranged as $y^2/m \ge 1/(1 + 1/y - 1/\alpha)$. Therefore, the ratio of the maximum number of bins used by $\mathcal{A}$ to that used by $\mathcal{O}$ is $y/(m/y)$, which is at least $1/(1+1/y-1/\alpha) \ge 1 + 1/\alpha$ for $y \ge (\alpha+1)/(\alpha+1/\alpha-1)$. (2) Suppose that $(m-y)/y > y(\alpha-1)/\alpha$. The maximum number of bins used by $\mathcal{A}$ at any time is $y + (m-y)/y - y(\alpha-1)/\alpha = m/y + y(1/\alpha - 1/y) \ge m/y + (m/y)(1/\alpha - 1/y)$. The latter inequality holds because $m \le y^2$. Therefore, the ratio of the maximum number of bins used by $\mathcal{A}$ to that used by $\mathcal{O}$ is at least $1 + 1/\alpha - 1/y$. By letting $\epsilon = 1/y$, the ratios in both case are at least $1 + 1/\alpha - \epsilon$, thus, the theorem holds. $\square$

We then modify the adversary such that it also works for known duration. The major issue is that we have to determine the departure time of the items when they arrive such that at the end of Stage 2, the on-line algorithm uses y bins, each containing exactly one item of width $1/y$. Then, using a similar argument in Theorem 1, we can prove the same lower bound for known duration. Roughly speaking, the departure time of the item i, for $i \ge 2$, depends on how the on-line algorithm pack item $(i-1)$, in particular, whether item $(i-1)$ is packed in an empty bin or not. The details are as follows.

To set the departure time d_i of item i arriving at time i, we need to consider how the on-line algorithm $\mathcal{A}$ packs the items arrived so far. We capture this information by two values β_{i-1} and γ_{i-1} (to be defined). For any time t, let I_t be the set of items arrived at or before t and $F_t \subseteq I_t$ be the set of items that $\mathcal{A}$ opens an empty bin for. We define $\beta_t = \min_{i \in F_t}\{d_i\}$ and $\gamma_t = \max_{i \in I_t - F_t}\{d_i\}$.

Now, we describe how to set the departure times. Firstly, we set $d_1 = L$ for some large constant L to be defined later. Note that $\beta_1 = L$ and $\gamma_1 = 0$. Next, we set $d_2 = (\beta_1 + \gamma_1)/2 = L/2$. If $\mathcal{A}$ packs item 2 in the same bin as item 1, we have $\beta_2 = L$ and $\gamma_2 = L/2$. Otherwise, we have $\beta_2 = L/2$ and $\gamma_2 = 0$. In general, we set $d_i = (\beta_{i-1} + \gamma_{i-1})/2$. Notice that Stage 2 ends at time γ_m and Stage 3 starts at time $\gamma_m + 1$. For the adversary to work, we need to ensure that d_i is an integer and $d_i > m$ for all $1 \leq i \leq m$, which can be proved to hold if $L \geq 2^{y+\lceil logm+1 \rceil}$. The details will be given in the full paper.

The following lemma gives an invariant about the departure times.

Lemma 2. *For any time $1 \leq t \leq \gamma_m$, we have $\beta_t > \gamma_t$.*

Proof. We first consider the case where $1 \leq t \leq m$. We prove the lemma by a stronger claim that (1) if item t is packed in an empty bin by $\mathcal{A}$, then $\beta_t = d_t$, otherwise, $\gamma_t = d_t$; and (2) $\beta_t > \gamma_t$. We prove the claim by an induction on t. The claim holds for $t = 1$ because $d_1 = \beta_1 = L$ and $\gamma_1 = 0$. Assume the claim holds for some $t < k \leq m$. At time k, item k arrives. Since $d_k = (\beta_{k-1} + \gamma_{k-1})/2$ and $\beta_{k-1} > \gamma_{k-1}$, we have $\gamma_{k-1} < d_k < \beta_{k-1}$. If item k is packed in an empty bin, then $\beta_k = \min\{d_k, \beta_{k-1}\} = d_k$ and $\gamma_k = \gamma_{k-1}$. Otherwise, $\gamma_k = \max\{d_k, \gamma_{k-1}\} = d_k$ and $\beta_k = \beta_{k-1}$. For both cases, we have $\beta_k > \gamma_k$, thus, the claim holds.

For any $m < t \leq \gamma_m$, there is no item released at time t. Therefore, we have $\beta_t = \beta_m > \gamma_m = \gamma_t$. Combining with the above claim, the lemma follows. □

Using the invariant of Lemma 2 and a similar argument as Theorem 1, we have the following theorem.

Theorem 2. *Any on-line algorithm for UFBP with known duration is at least $(1 + 1/\alpha - \epsilon)$-competitive for any $\epsilon > 0$ and $\alpha = \min_i\{w_i\}$.*

Proof. Notice that with known duration, we can have an off-line algorithm using the same number of bins as the one given in the proof of Theorem 1. As a result, we only need to consider the number of bins used by any on-line algorithm. By Lemma 2, we can see that at time $\gamma_m + 1$ each of the y occupied bins has exactly one item of width $1/y$. By Lemma 1, the on-line algorithm uses at least $y + \max\{0, (m-y)/y - y(a-1)/\alpha\}$ bins. Following the same argument as in Theorem 1, we can prove that every on-line algorithm for UFBP with known duration has competitive ratio at least $1+1/\alpha-\epsilon$ for any $\epsilon > 0$ and $\alpha = \min_i\{w_i\}$. □

Lower bound for WS To prove the lower bound for WS, we modify the adversary for UFBP such that a bin is considered as a channel and an item of width $1/w$ becomes an item of window w. Then, we have an adversary for the on-line WS. Based on the modified adversary, we can derive the following theorem.

Theorem 3. *Any on-line algorithms for WS is at least $(1+1/\alpha-\epsilon)$-competitive for any $\epsilon > 0$ and $\alpha = \min_i\{w_i\}$. This holds for both known and unknown duration cases.*

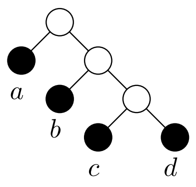

Fig. 1. The schedule corresponding to this broadcast tree is $\langle a\,b\,a\,c\,a\,b\,a\,d \rangle$.

4 Upper bound results

In this section, we give an on-line algorithm $\mathcal{W}$ for the WS problem and analyze its performance. We focus on the unknown duration case, the result carries for the known duration case. Roughly speaking, the on-line algorithm $\mathcal{W}$ rounds each item window w down to w' which is a power of two (e.g., we round the window of 7 down to 4) and then schedule the item according to w'. Note that if any item is broadcasted at least once in every interval of w', it is broadcasted at least once in every interval of w. As a result, we can first focus on scheduling items with windows that are powers of two.

4.1 Broadcast trees - representation of schedules

Similar to the work by Bar-Noy et al. [4], we represent a broadcast schedule on m channels by a forest of m binary trees. The schedule on each channel is represented by a binary tree in which all nodes have exactly zero or two children. We call this binary tree a *broadcast tree*. Given any broadcast tree, the schedule is to alternately broadcast an item from the left and right subtrees; items from each subtree T is selected alternately from the left and right subtrees of T in a recursive manner. For example, in Figure 1, the broadcast tree represents a schedule that alternates between the item a and the items on the right subtree; when selecting the right subtree, we alternate between the item b and items on the right subtree recursively. The corresponding schedule is $\langle a\,b\,a\,c\,a\,b\,a\,d \rangle$. Notice that a leaf at depth d represents an item scheduled with window $w = 2^d$.

To ease the discussion of the on-line algorithm $\mathcal{W}$, we give some definitions on broadcast trees. We say that a leaf is *open* if there is no item assigned to the leaf, otherwise, it is *closed*. An open leaf is represented by an unfilled circle and closed by filled circle. We label a leaf at depth d by 2^d. See Figure 2 (a) for an example. The *load* of a broadcast tree is defined to be the sum of reciprocal of the labels of all closed leaves in the tree. A *lace binary tree* of height h is a binary tree in which for each depth from 1 to $h-1$, there is a single leaf and for depth h, there are two leaves. Note that there are more than one different lace binary tree of height $h > 2$. The tree in Figure 2 (c) is a lace binary tree of height 3.

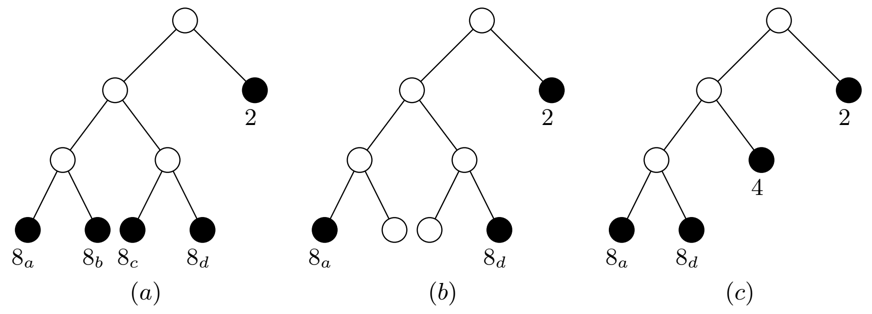

Fig. 2. Appropriate rearrangement is necessary after departure of items. (a) Broadcast tree for the sequence $\{8_a, 8_b, 8_c, 8_d, 2\}$. (b) Items 8_b and 8_c depart; a new item 4 cannot be included into the tree. (c) The item 4 can be included if the tree is restructured.

4.2 The on-line algorithm $\mathcal{W}$

The on-line algorithm by Bar-Noy et al. expands an existing broadcast tree or opens a new broadcast tree when items arrive. Our on-line algorithm $\mathcal{W}$ handles newly arrived items similarly. Yet when items depart, we have to rearrange a broadcast tree wisely to make sure that new items can still be inserted to a broadcast tree as long as the tree is not very full. We illustrate the need for proper rearrangement in the following example. Suppose there are 4 items of window 8 and 1 item of window 2 arriving and the broadcast tree is as shown in Figure 2 (a). Later the second and third items of window 8 depart and another item of window 4 arrives. Figure 2 (b) shows the broadcast tree after the items depart if we do not rearrange the tree properly; we cannot include the item of window 4 into the tree. Yet if we rearrange the tree as in Figure 2 (c), we are able to include the new item into the tree.

Now, we describe the on-line algorithm $\mathcal{W}$. The on-line algorithm $\mathcal{W}$ attempts to maintain an invariant on the structure of the broadcast trees: for every broadcast tree, there is at most one open leaf at each depth. To keep this invariant, the on-line algorithm $\mathcal{W}$ modifies the structure according to the arrival and departure of items as follows.

Arrival: When an item with window w arrives, round it down to the nearest power of 2, say $w' = 2^v$. If there is an open leaf at depth v in some broadcast tree, schedule the item to that leaf. Otherwise, let $u < v$ be the maximum value such that there is an open leaf at depth u in some broadcast tree. If no such u exists, open a new tree with one open leaf (in this case $u = 0$). Let ℓ be the leaf selected. Append a lace subtree of height $v - u$ with all leaves open to the tree to replace ℓ and then schedule the new item to one of the open leaves at depth v in the resulting tree. See Figure 3 (a)-(d) for examples.

Departure: When an item at depth v of a broadcast tree is going to depart in the next 2^v time, the corresponding leaf ℓ will become open. If there is no

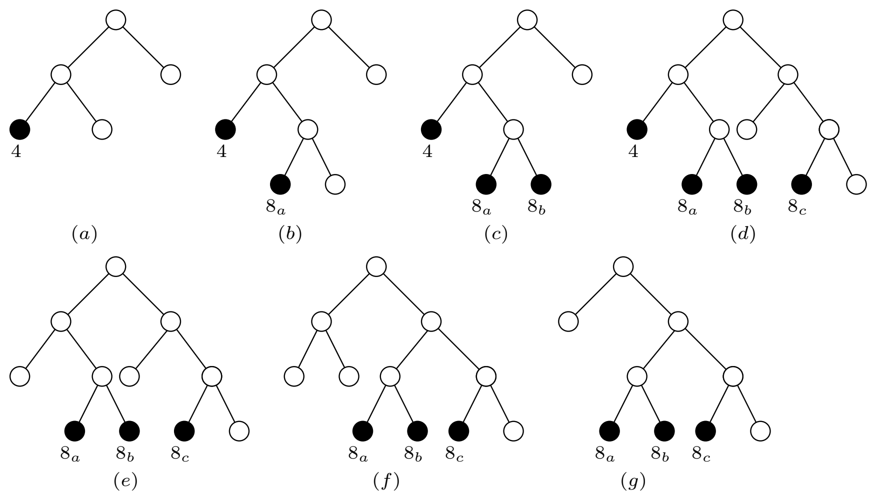

Fig. 3. (a)-(d) Evolution of the broadcast tree for the sequence $\{4, 8_a, 8_b, 8_c\}$. Note that the lace binary trees inserted are chosen for better drawing effect only. (e)-(g) Evolution of the broadcast tree when the item 4 departs.

other open leaf at depth v of the tree, no restructuring is needed. Otherwise, there is exactly one other open leaf f. Let ℓ' be the node, a closed leaf or an internal node, that shares the same parent p with ℓ in the tree. Detach the subtree rooted at ℓ' and replace f by this subtree. Remove ℓ and make p an open leaf. Then we repeat the restructuring similarly at depth $v - 1$ and upper depth, if necessary, by considering p as the new open leaf. See Figure 3 (e)-(f) for an example.

Remarks: When we transit from the old to the new broadcast schedule, the interval between the broadcast of an item that has been moved is altered. To make sure that the interval is still smaller than or equal to its window size, we will broadcast the item once more at the original time slot (which becomes an empty slot in the new schedule), if necessary. This can be proved that the remedy guarantees the broadcast interval to be at most the window size of the moved item, and the transition lasts for at most 2^v time units. The details will be given in the full paper.

4.3 Analysis of the on-line algorithm $\mathcal{W}$

In this section, we analyze the performance of the on-line algorithm $\mathcal{W}$. Roughly speaking, we show that when the on-line algorithm $\mathcal{W}$ opens a new broadcast tree for an item, the load in each existing broadcast tree is reasonably close to 1. Since the optimal off-line algorithm can schedule in each channel items with load at most 1, we can show that the number of channels used by the on-line algorithm $\mathcal{W}$ is at most a constant times that is used by the optimal off-line algorithm. The first step is to show in Lemmas 3 and 4 that a broadcast tree is reasonably full whenever we cannot schedule a new item in the tree.

Lemma 3. *For every broadcast tree, there is at most one open leaf at each depth at any time.*

Proof. The proof is by induction on time. Initially, there is only one open leaf at depth zero (the root of the first broadcast tree). When a new item arrives, we either close an open leaf or replace an open leaf ℓ with a lace tree. Note that ℓ is chosen in a way that its depth is the largest among all candidate open leaves. That means there is no open leaf on each depth corresponding to the added lace tree. Furthermore, the lace tree only contains one single leaf in each depth except the bottommost one in which one of the two open leaves is assigned to the new item. As a result, adding a new item to a broadcast tree keeps the invariant.

When an item departs, a closed leaf becomes open. If there is no other open leaf at the same depth, the number of open leave is then one. If there is another open leaf at the same depth, the broadcast tree will be restructured so that the two open leaves are attached to the same parent. The two open leaves will be removed and then the number of open leaves will become zero. However, the parent of the two open leaves is made open, thus increase the number of open leaves at that depth. The restructuring procedure is repeated, if necessary, and thus keeping at most one open leaf at other depth. □

Lemma 4. *If a new item with window 2^v cannot be scheduled in a broadcast tree T, the load of T is at least $1 - 1/2^v$.*

Proof. By the definition of the on-line algorithm $\mathcal{W}$, there is no open leaf at depth $v' \leq v$. By Lemma 3, there is at most one open leaf at depth greater than v. Notice that a broadcast tree with all leaves closed has a load of 1. Therefore, the total load of T is at least $1 - \sum_{u>v} 1/2^u \geq 1 - 1/2^v$. □

Based on the above two lemmas, we can derive the competitive ratio of the on-line algorithm $\mathcal{W}$ as follows.

Theorem 4. *The on-line algorithm $\mathcal{W}$ is $2 + 2/(\alpha^* - 1)$-competitive when $\alpha^* \geq 2$, where α^* is the largest power of 2 smaller than or equal to $\min_i\{w_i\}$. This holds for both known and unknown duration cases.*

Proof. Suppose the on-line algorithm $\mathcal{W}$ opens a new broadcast tree for a new item with window w and $w' = 2^v$ is the corresponding round down window. For any broadcast tree T in the forest, by Lemma 4, the total load of T is at least $1 - 1/2^v$. Thus, the total load of the corresponding channel is at least $(1 - 1/2^v)/2 = 1/2 - 1/2^{v+1}$ (because of the round down to nearest power of 2).

Let m be the number of channels used by the on-line algorithm $\mathcal{W}$. Then the total load of items, and hence the number of channels used by the optimal off-line algorithm is at least $(m-1)(1/2 - 1/2^{v+1})$. Therefore, by taking an appropriate additive constant, the competitive ratio is at most $1/(1/2 - 1/2^{v+1}) = 2 + 2/(2^v - 1) \geq 2 + 2/(\alpha^* - 1)$. Thus, the theorem follows. □

Theorem 5. *The on-line algorithm $\mathcal{W}$ is 5-competitive when $\min_i\{w_i\} = 1$. This holds for both known and unknown duration cases.*

Proof. At any instance, suppose the on-line algorithm $\mathcal{W}$ uses n channels for items with window 1 and m channels for items with other window values. Then the optimal off-line algorithm must use at least n channels at this instance. Furthermore, consider the moment when the on-line algorithm $\mathcal{W}$ opens the m-th channels for item with window $w > 1$, using a similar argument as in Theorem 4, the optimal off-line algorithm needs at least $(m-1)/4$ channels (set $\alpha^* = 2$ in the formula in Theorem 4). In other words, the optimal off-line algorithm uses at least $\max\{n, (m-1)/4\}$ channels while the on-line algorithm $\mathcal{W}$ uses $m+n$ channels. The worst case ratio is obtained when $n = (m-1)/4$; by taking appropriate additive constant, the competitive ratio is at most 5 and the theorem follows. □

5 Conclusion

In this paper we study on-line windows scheduling of temporary items. We have given a 5-competitive on-line algorithm and showed that there is a lower bound of $2-\epsilon$ for any $\epsilon > 0$ on the competitive ratio of any on-line algorithm. An immediate open question is whether we can close the gap. Another interesting problem is to schedule as many items as possible when the number of channels is fixed; this involves admission control.

References

[1] S. Acharya, M. 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 and R. E. Ladner. Windows scheduling problems for broadcast systems. *SIAM Journal on Computing*, 32(4):1091–1113, 2003. (A preliminary version appears in SODA 2002.).

[4] A. Bar-Noy, R. E. Ladner, and T. Tamir. Windows scheduling as a restricted version of bin packing. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 224–233, 2004.

[5] A. Bar-Noy, J. Naor, and B. Schieber. Pushing dependent data in clients-providers-servers systems. *Wireless Network*, 9(5):421–430, 2003.

[6] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

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

[8] M. Y. Chan and F. Y. L. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9(5):425–625, 1993.

[9] E. G. Coffman, M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM Journal on Computing*, 1983.

[10] P. C. Fishburn and J. C. Lagarias. Pinwheel scheduling: Achievable densities. *Algorithmica*, 34(1):14–38, 2002.

[11] V. Gondhalekar, R. Jain, and J. Werth. Scheduling on airdisks: Efficient access to personalized information services via periodic wireless data broadcast. In *Proceedings of IEEE International Conference on Communications*, volume 3, pages 1276–1280, 1997.

[12] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proceedings of ACM SIGCOMM conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 89–100, 1997.

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

[14] S. Viswanathan and T. Imielinski. Pyramid broadcasting for video-on-demand service. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 66–77, 1995.

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