# A General Buffer Scheme for the Windows Scheduling Problem

AMOTZ BAR-NOY
Brooklyn College, NY
RICHARD E. LADNER and JACOB CHRISTENSEN
University of Washington
and
TAMI TAMIR
The Interdisciplinary Center, Herzliya

Broadcasting is an efficient alternative to unicast for delivering popular on-demand media requests. Broadcasting schemes that are based on windows scheduling algorithms provide a way to satisfy all requests with both low bandwidth and low latency.

Consider a system of $n$ pages that need to be scheduled (transmitted) on identical channels an infinite number of times. Time is slotted, and it takes one time slot to transmit each page. In the *windows scheduling problem* (WS) each page $i$, $1 \leq i \leq n$, is associated with a *request window* $w_i$. In a feasible schedule for WS, page $i$ must be scheduled at least once in any window of $w_i$ time slots. The objective function is to minimize the number of channels required to schedule all the pages.

The main contribution of this paper is the design of a general *buffer scheme* for the windows scheduling problem such that *any* algorithm for WS follows this scheme. As a result, this scheme can serve as a tool to analyze and/or exhaust all possible WS-algorithms.

The buffer scheme is based on modelling the system as a nondeterministic finite state machine in which any directed cycle corresponds to a legal schedule and vice-versa. Since WS is NP-hard, we present some heuristics and pruning-rules for cycle detection that ensure reasonable cycle-search time.

By introducing various rules, the buffer scheme can be transformed into deterministic scheduling algorithms. We show that a simple page-selection rule for the buffer scheme provides an optimal schedule to WS for the case where all the $w_i$'s have divisible sizes, and other good schedules for some other general cases. By using an exhaustive-search, we prove impossibility results for other important instances.

We also show how to extend the buffer scheme to more generalized environments in which (*i*) pages are arriving and departing on-line, (*ii*) the window constraint has some *jitter*, and (*iii*) different pages might have different lengths.

---

## 1. INTRODUCTION

Currently, popular on-demand data on the Internet is provided in a unicast way, by requesting it from a server. Such systems are called *pull* systems. A very high demand over a short period of time may put stress on both server and network bandwidth. This stress can be alleviated by replicating data in mirrors or caches. An alternative approach to on-demand for popular data is a *push* system where the data is provided by periodic broadcast or multicast. Those desiring and authorized to receive the data simply wait, hopefully a short period of time, for the broadcast. Pushing has the advantage over pulling in that it requires less server and network bandwidth, as long as the demand is high. This approach to providing popular data has led to a very interesting problem. What are the best ways to partition the channel in a time multiplexed way to provide the service in a push system? This general question can be modelled mathematically in a number of ways. We choose a specific approach called *windows scheduling (WS)* [Bar-Noy and Ladner 2003; Bar-Noy, Ladner, and Tamir 2003]. In this article, we propose a new algorithmic technique called the *buffer scheme* that can be used to design algorithms to solve WS problems and several extensions of WS that cannot be solved using known algorithms. In addition, the buffer scheme can be used to prove impossibility results.

An instance to WS is a sequence $\mathcal{W} = \langle w_1, \ldots, w_n \rangle$ of $n$ request windows, and a set of $h$ identical channels. The window request $w_i$ is associated with a page $i$. Time is slotted, and it takes one time slot to transmit any page on any channel. The output is a feasible schedule in which for all $i$, the page $i$ must be scheduled (transmitted) on one of the $h$ channels at least once in any window of $w_i$ consecutive time slots. Equivalently, the requirement is that the *gap* between any two consecutive appearances of $i$ in the schedule is at most $w_i$ slots. We say that a schedule is *perfect* if for all $i$, the gap between any two consecutive appearances of $i$ in the schedule is a constant $w_i'$ for some $w_i' \leq w_i$.

The optimization problem associated with WS is to minimize the number of channels required to schedule all $n$ pages. Define $1/w_i$ as the *width* of page $i$ and let $h_0(\mathcal{W}) = \lceil \sum_i 1/w_i \rceil$. Then, $h_0(\mathcal{W})$ is an obvious lower bound on the minimum number of channels required for $\mathcal{W}$.

*Example I.*   An interesting example is that of *harmonic scheduling*, that is, scheduling sequences $\mathcal{H}_n = \langle 1, 2, \ldots n \rangle$ in a minimum number of channels. Harmonic windows scheduling is the basis of many popular media delivery schemes (e.g., [Viswanathan and Imielinski 1996; Hua and Sheu 2000; Juhn and Tseng 1997; Pâris, Carter, and Long 1999]). The following is a nonperfect schedule of 9 pages on 3 channels for the window sequence $\mathcal{H}_9 = \langle 1, 2, \ldots, 9 \rangle$.

$$\begin{bmatrix} 1 & 4 & 1 & 1 & 1 & 1 & 1 & 6 & 1 & 1 & 1 & 1 & \cdots \\ 2 & 1 & 2 & 5 & 2 & 4 & 2 & 5 & 2 & 4 & 2 & 5 & \cdots \\ 3 & 6 & 7 & 3 & 8 & 9 & 3 & 1 & 7 & 3 & 9 & 8 & \cdots \end{bmatrix}$$

Note that a page may be scheduled on different channels (e.g., 1 is scheduled on all three channels). Also, the gaps between any two consecutive appearances of $i$ need not be exactly $w_i$ or another *fixed* number (e.g., the actual window granted to 5 is 4 and the actual windows of 8 and 9 are sometimes 5 and sometimes 7). Even though this schedule is not "nicely" structured, it is feasible because it obeys the requirement that the maximal gap between any two appearances of $i$ is at most $w_i$ for any $i$. Using our buffer scheme in exhaustive search mode, we show that there is no schedule for $\mathcal{H}_{10} = \langle 1, 2, \ldots, 10 \rangle$ on three channels even though $\sum_{i=1}^{10} 1/i < 3$.

*Example II.*   In this article, we demonstrate that for some instances such "flexible" schedules achieve better performance. Indeed, for the instance in Example *I*, there exists a perfect feasible schedule on three channels. However, for the following instance this is not the case. Let $n = 5$ and $\mathcal{W} = \langle 3, 5, 8, 8, 8 \rangle$. We show in this article that there is no feasible perfect schedule of these 5 pages on a single channel. However,

$$[3, 5, 8_a, 3, 8_b, 5, 3, 8_c, 8_a, 3, 5, 8_b, 3, 8_c, 5, 3, 8_a, 8_b, 3, 5, 8_c, \ldots]$$

is a feasible nonperfect schedule on a single channel. This schedule was found by efficiently implementing the buffer scheme. Most previous techniques only produce perfect schedules, requiring two channels.

## 1.1 Contributions

The main contribution of this article is the design of a general *buffer scheme* for the windows scheduling problem. We show that *any* algorithm for WS follows this scheme. Thus, this scheme can serve as a tool to analyze all WS-algorithms. The buffer scheme is based on presenting the system as a nondeterministic finite state machine in which any directed cycle corresponds to a legal schedule and vice-versa. The state space is very large; therefore, we present some heuristics and pruning-rules to ensure reasonable cycle-search time.

By introducing various rules for the buffer scheme, it can be transformed into deterministic scheduling algorithms. We show that a simple greedy rule for the buffer scheme provides an optimal schedule to WS for the case where all the $w_i$'s have divisible sizes. Our theoretical results are accompanied by experiments. We implemented the deterministic buffer scheme with various page selection rules. The experiments show that for many instances the deterministic schemes

perform better than the known greedy WS algorithm presented in Bar-Noy and Ladner [2003].

By using an exhaustive-search, we prove impossibility results and find some best possible schedules. As mentioned earlier, we prove that there is no schedule of $\mathcal{H}_{10} = \langle 1, 2, \ldots, 10 \rangle$ on three channels. In addition, we find the best possible schedules for other important instances. Similar to branch and bound, the search is done efficiently thanks to heavy pruning of early detected deadends. The results achieved in the exhaustive-search experiments appear not to be achievable in any other way.

The main advantage of the buffer scheme is its ability to produce nonperfect schedules. Most of the known algorithms (with or without guaranteed performance) produce perfect schedules. However, in the WS problem and its applications, such a restriction is not required. We demonstrate that the earliest deadline first (EDF) strategy is not the best for WS even though it is optimal for similar problems. We develop some understanding that leads us to the design of the largest backward move (LBM) strategy that performs well in our simulations.

The basic windows scheduling problem can be generalized in several ways that cannot be handled by previous techniques that only produce perfect schedules. (1) *Dynamic (on-line) environment:* pages are arriving and departing online and the set of windows is not known in advance. Here the scheme is extended naturally emphasizing its advantage as a framework to algorithms as opposed to other greedy heuristics for the off-line setting that cannot be generalized with such an ease. (2) *Jitter windows:* each page is given by a pair of windows $(w_i', w_i)$, meaning that page $i$ needs to be scheduled *at least* once in any window of $w_i$ time slots and *at most* once in any window of $w_i'$ time slots. In the original definition, $w_i' = 1$. Here again, the generalization is natural. (3) *Different lengths:* pages might have different lengths. The buffer scheme can be generalized to produce high-quality schedules in these generalizations.

## 1.2 Prior Results and Related Work

The windows scheduling problem belongs to the class of *periodic scheduling* problems in which each page needs to be scheduled an infinite number of times. However, the optimization goal in the windows scheduling problem is of the "max" type whereas traditional optimization goals belong to the "average" type. That is, traditional objectives insist that each page $i$ would receive its required share $(1/w_i)$, even if some of the gaps could be larger than $w_i$. The issue is usually to optimize some fairness requirements that do not allow the gaps to be too different than $w_i$. Two examples are *periodic scheduling* [Liu and Layland 1973] and *the chairman assignment problem* [Tijdeman 1980]. For both problems the Earliest Deadline First strategy was proven to be optimal. Our article demonstrates that this is not the case for the windows scheduling problem.

The pinwheel problem is the windows scheduling problem with one channel. The problem was defined in Holte et al. [1989, 1992] for unit-length pages and was generalized to arbitrary length pages in Baruah and Lin [1998]; Feinberg

et al. [2002]. In these papers and other papers about the pinwheel problem the focus was to understand which inputs can be scheduled on one channel. In particular, the papers [Chan and Chin 1992,1993] optimized the bound on the value of $\sum_{i=1}^{n}(1/w_i)$ that guarantees a feasible schedule.

The windows scheduling problem was defined in [Bar-Noy and Ladner 2003], where it is shown how to construct perfect schedules that use $h_0(\mathcal{W}) + O(\log h_0(\mathcal{W}))$ channels. This asymptotic result is complemented with a practical greedy algorithm, but no approximation bound has been proved for it yet. Both the asymptotic and greedy algorithms produce only perfect schedules.

The general WS problem can be thought of as a scheduling problem for push broadcast systems (e.g., Broadcast Disks [Acharya, Franklin, and Zdonik 1995] or TeleText services [Ammar and Wong 1985]) where satellites broadcast popular information pages to clients. In such a system there are clients and servers. The server chooses what information to push in order to optimize the quality of service for the clients (mainly the response time). In a more generalized model, the servers are not the information providers. 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 size. Finally, various maintenance problems where considered with similar environments and optimization goals (e.g., [Wei and Liu 1983; Anily, Glass, and Hassin 1998]).

WS is known to be NP-hard. In a way, this justifies the efforts of this article. A proof for the case where $i$ must be granted an exact $w_i$ window is given in [Bar-Noy, Bhatia, Naor, and Schieber 2002]. Another proof that is suitable also for the flexible case in which the schedule of $i$ need not be perfect is given in Bar-Noy, Ladner, and Tamir [2007].

Another related problem is periodic scheduling with vacations, in which each page is associated with a profit, gained each time the page is transmitted, and the goal is to maximize the average profit per time unit. Pages can not be transmitted too often, each page there is a *vacation* parameter $w_i'$ and there must be a gap of at most $w_i'$ slots between two transmissions of page $i$. The paper by Sgall, Shachnai, and Tamir [2005] studies this problem and shows how the buffer scheme can be combined with an algorithm to find a cycle with maximal mean weight in a weighed graph to find optimal schedules.

## 2. THE GENERAL BUFFER SCHEME

In this section, we describe the buffer scheme and prove that for any instance of windows scheduling, any schedule can be generated by the buffer scheme. We then discuss how the buffer scheme can be simulated efficiently by early detection and pruning of dead-end states. Using these pruning rules, we establish an efficient implementation of the scheme that can exhaust all possible solutions. For big instances for which exhaustive search is not applicable, we suggest a greedy rule that produces a single execution of the scheme that "hopefully" generates a correct infinite schedule.

### 2.1 Overview of the Scheme

Let $\mathcal{W} = \langle w_1, \ldots, w_n \rangle$ and number of channels $h$ be an instance of the windows scheduling problem. Let $w^* = \max_i \{w_i\}$. We represent the pages state using

a set of buffers, $B_1, B_2, \ldots, B_{w^*}$. Each page is located in some buffer. A page located in $B_j$ must be transmitted during the next $j$ slots. Initially, buffer $B_j$ includes all the pages with $w_i = j$. We denote by $b_j$ the number of pages in $B_j$ and by $\ell_i$ the *location* of page $i$ (i.e., $i \in B_{\ell_i}$).

In each iteration, the scheme schedules at most $h$ pages on the $h$ channels. By definition, the pages of $B_1$ must be scheduled. In addition, the scheme selects at most $h - b_1$ additional pages from other buffers to be scheduled in this iteration. The way these pages are selected is the crucial part of the scheme and is discussed later. After selecting the pages to be scheduled, the scheme updates the content of the buffers.

- For all $j > 1$, all the *nonscheduled* pages located in $B_j$ are moved to $B_{j-1}$.
- Each scheduled page, $i$, is placed in $B_{w_i}$—to ensure that the next schedule of $i$ will be during the next $w_i$ slots.

This description implies that the space complexity of the buffer scheme depends on $w^*$. However, by using a data structure that is "page-oriented," the buffer scheme can be implemented in space $O(n)$.

From the pages' point of view, a page is first located as far as possible ($w_i$ slots) from a deadline (represented by $B_1$), it then gets closer and closer to the deadline and can be selected to be transmitted in any time during this approachment toward the deadline. With no specific rule for selecting which of the $h - b_1$ pages should be scheduled, the buffer scheme behaves like a nondeterministic finite state machine with a very large state space, where a state is simply an assignment of pages to buffers.

In running the buffer scheme nondeterministically, it fails if in some time point $b_1 > h$. The scheme is successful if it produces an infinite schedule. Note that since the number of states is finite ($w_1 \cdot w_2 \cdot \ldots \cdot w_n$), this is equivalent to having two time slots $t_1, t_2$ such that the states at $t_1$ and $t_2$ are identical. Given these two time slots, the page-selection sequence between $t_1$ and $t_2$ can be repeated forever to obtain an infinite schedule.

THEOREM 2.1. *When it does not fail, the buffer scheme produces a feasible schedule.*

PROOF. In each iteration, the buffer scheme schedules at most $h$ pages; since there are $h$ channels, this is fine. Consider any page $i$. Recall that initially and whenever $i$ is scheduled it is located in $B_{w_i}$. Since $i$ advances one buffer toward $B_1$ in each time slot, and since $i$ is selected from $B_1$ or earlier during this advancement, it is guaranteed that the gap between any two selections of $i$ is at most $w_i$ slots. □

THEOREM 2.2. *Any feasible schedule for WS can be generated by the buffers scheme.*

PROOF. Given a schedule of $\mathcal{W}$, let $S_t$ be the set of pages scheduled at time slot $t$. Consider the buffer scheme execution that selects at time slot $t$ the pages of $S_t$. We show that this is a possible valid (nondeterministic) execution of the scheme. First, for any $t$, $|S_t| \leq h$, thus, the number of selected pages does not

exceed $h$. Next, we show that all the pages of $B_1$ are selected. Recall that the initial location of $i$ is $B_{w_i}$. Let $i \in B_1$ at time $t$. By definition of the buffer scheme, it must be that the previous schedule of $i$ took place at time $t - w_i - 1$ or that this is the $(w_i - 1)$th slot of the schedule and $i$ has not been scheduled yet. Given that the schedule we follow is feasible, it must be that $i \in S_t$, and therefore there exists one nondeterministic execution that selects the same set of $h - b_1$ or less additional pages to obtain the set $S_t$.  □

*Remark.* In our simulations and in the page-selection rules we suggest, no channel is "idle" in the execution; that is, exactly $h$ pages are scheduled in each time slot. It is important to observe that this no-idle policy is superior over scheduling policies that allow idles. Informally, if there exists a schedule of $\mathcal{W}$ in which some channels are idle, then a nonidle schedule also exists. This is true, since we can always fill idle slots by scheduling arbitrary pages. In buffer scheme terms, by doing this each additional selected page is moved by the scheme from its current location, $\ell_i$, to $B_{w_i}$. Since $\ell_i \leq w_i$, this only increases the distance of $i$ from $B_1$.

## 2.2 Page Selection Criteria and Dead-ends Detection

As mentioned above, the buffer scheme fails if at some time point $b_1 > h$, that is, more than $h$ pages must be scheduled in the next time slot. However, we can establish other, more tight, dead-end conditions. Then, by trying to avoid these dead-ends, we can establish "good" page selection criteria. In this section, we present a tight dead-end criteria, and describe how to greedily select pages in each time slot in a way that delays (and hopefully avoids) a dead-end state. Given a state of the buffers, let $c(i, j)$ denote the number of times $i$ must be scheduled during the next $j$ slots in any feasible schedule.

CLAIM 2.3. *For any $i, j$,*

$$c(i, j) \geq \begin{cases} 0 & \text{if } j < \ell_i \\ 1 + \left\lfloor \frac{j - \ell_i}{w_i} \right\rfloor & \text{if } j \geq \ell_i \end{cases}$$

PROOF. If $j < \ell_i$, that is, if $i$ is located beyond the first $j$ buffers, we do not need to schedule $i$ at all during the next $j$ slots. If $j \geq \ell_i$, then we must schedule $i$ once during the next $\ell_j$ slots. After this schedule, $i$ will be located in $B_{w_i}$. Note that for any $t$, given that $i \in B_{w_i}$, we must schedule $i$ at least $\lfloor t/w_i \rfloor$ times during the next $t$ slots. In our case, we have $t = j - \ell_i$, since this is the minimal number of slots that remains after the first schedule of $i$.  □

For example, if $\ell_i = 1$, $w_i = 3$ and $j = 11$, then $c(i, j) = 4$. This implies that $i$ must be scheduled at least 4 times during the next 11 slots: once in the next slot, and three more times in the remaining 10 slots. Let $c(j)$ denote the total number of page schedules the system must provide in the next $j$ slots. By definition, $c(j) = \sum_{i=1}^{n} c(i, j)$. By definition, $jh$ is the number of available page schedules in the next $j$ slots. Let $f(j) = jh - c(j)$ denote the *freedom level* existing in the next $j$ slots.

If for some $j$, $f(j) < 0$ then a dead-end state is reached. If $f(j) = 0$, then only pages from the first $j$ buffers must be scheduled in the next $j$ slots. If $f(j) > 0$, then some freedom exists in the way the pages are selected. That is, $c(j)$ pages must be selected from the first $j$ buffers, and the remaining $f(j)$ pages can come from any buffer. In particular, for $j = 1$, only the pages in $B_1$ are considered, thus this rule generalizes the obvious condition for $B_1$.

Importantly, it is possible to know how many pages must be selected from the first $j$ buffers *in the next slot*. For any $j$, the system can provide, at most, $(j-1)h$ page-schedules during any $j-1$ slots. Thus, at least $n(j) = c(j)-(j-1)h$ pages from the first $j$ buffers must be selected in the next slot to avoid a dead-end. Again, this condition generalizes the condition for $B_1$.

## 2.3 Delaying Dead-ends and Deterministic Rules

We present a greedy way to select the pages to be scheduled based on the parameters $c(j)$ and $n(j)$ that are calculated during the selection process. Let $s$ denote the number of pages selected so far in the current iteration. Initially, $j = 1$ and $s = 0$. As long as $s < h$, continue selecting pages as follows. For each $j$, if $n(j) > h$ the selection process fails. If $n(j) = s$, there are no constraints due to $B_j$ (since $s$ pages have already been selected from the first $j$ buffers) and the selection proceeds to $j + 1$. Otherwise ($s < n(j) \leq h$), select from the first $j$ buffers $n(j)-s$ pages that were not selected yet, and proceed to $j+1$. Note that this scheme is still nondeterministic because we have not yet specified exactly which pages are scheduled. We call this scheme the *restricted buffer scheme*.

THEOREM 2.4.    *Any legal schedule for WS can be generated by the restricted buffer scheme.*

PROOF.    The proof is identical to the proof of Theorem 2.2. Since we are given an infinite schedule, it is guaranteed that a page selection which follows this schedule never leads to a dead-end state. ☐

We now give some deterministic rules for deciding exactly which pages to schedule in a restricted buffer scheme. In applying the restricted buffer scheme, it must determine, given a specific $k$ and $j$, which $k$ pages from the first $j$ buffers are to be scheduled in the next time slot. Naturally, high priority is given to pages whose transmission will reduce the most the *load* on the channels.

This load can be measured by a potential function based on the locations of the pages. We suggest two greedy selection rules, each of them maximizes a different potential function. Our first greedy rule is suitable for the potential function $\phi_1 = \sum_i \ell_i$. Our second greedy rule is suitable for the potential function $\phi_2 = \sum_i \ell_i/w_i$. These two approaches are realized by the following rules:

1.  Select pages for which $w_i - \ell_i$ is maximal.
2.  Select pages for which $(w_i - \ell_i)/w_i$ is maximal.

In the first rule, denoted LBM, pages that can increase $\phi_1$ the most are selected. In LBM, pages that will move the most are scheduled first. In the second rule, denoted WLBM (*weighted LBM*), the pages that increase $\phi_2$ the most are selected. Each of these rules can be applied when ties are broken in favor of

pages associated with smaller windows or larger windows. Our simulations reveal that breaking ties in favor of pages with small windows performs better for almost all inputs. On the other hand, we cannot crown any of these two rules as the ultimate winner. For the first rule, we show that it is optimal for a large set of instances, even without the dead-end detection of the restricted buffer scheme. The second rule performs better on large harmonic instances. For both rules, the simulations give good results (see Section 3).

In our simulations, a third natural greedy rule is considered, *Earliest Deadline First*, in which the pages with minimal $\ell_i$ are selected. This rule is optimal for other periodic scheduling problems that care about average gaps (e.g., periodic scheduling [Liu and Layland 1973] and the chairman assignment problem [Tijdeman 1980]). However, in our problem, this rule performs poorly. This can be explained by the fact that deadlines are well considered by the dead-end detection mechanism of the restricted buffer scheme. The role of the additional page selection is to reduce future load on the channels.

## 2.4 The LBM Selection Rule

Let LBM be the buffer scheme with the greedy rule that prefers pages with large $(w_i - \ell_i)$ value and breaks ties in favor of pages with smaller windows. We show that LBM is optimal for a large set of instances even without the dead-end detection mechanism of the restricted buffer scheme. Without dead-end detection, LBM runs as follows:

1. Initialization: Place $i$ in buffer $B_{w_i}$ for all $1 \leq i \leq n$.
2. In each time slot:
   a. If $b_1 > h$ then terminate with a failure.
   b. Otherwise, schedule all the pages from $B_1$.
   c. If $h > b_1$, select $h - b_1$ additional pages with the largest $(w_i - \ell_i)$ value, break ties in favor of pages with smaller windows.

### 2.4.1 *Optimality for Divisible-size Instances*

*Definition 2.1.* An instance $\mathcal{W}$ of WS is a divisible-size instance, if $w_{i+1}$ divides $w_i$ in the sorted sequence of windows $w_1 \geq \cdots \geq w_i \geq w_{i+1} \geq \cdots \geq w_n$ for all $1 \leq i < n$.

For example, an instance in which all the windows are powers of 2 is a divisible-size instance. The divisible-size constraint is not unreasonable. For example, pages could be advertising slots which are only offered in windows that are powers of 2, similar to the way magazines sell space only in certain fractions, 1/2 page, 1/4 page, and so on. The following theorem proves that LBM is optimal for divisible-size instances.

THEOREM 2.5. *If an instance, $\mathcal{W}$, of WS is a divisible-size instance and $h \geq h_0(\mathcal{W})$, then LBM never fails.*

PROOF. Let $\mathcal{W} = \langle w_1, w_2, \ldots, w_n \rangle$ where for all $i$ and $j$, $w_i$ divides $w_j$ or vice versa. We show that LBM never has more than $h$ pages in $B_1$. Assume the opposite and let $t$ be the first time in which there are more than $h$ pages in $B_1$.

Let $J_k$ be any page with the largest window request in $B_1$ at time $t$. Since $J_k$ reaches $B_1$, it must be that $t \geq w_k - 1$. By combining the following two Lemmas, we obtain the statement of the Theorem.

LEMMA 2.6.    *If $t$ is the first time that $J_k$ reaches $B_1$, there are at most $h - 1$ additional pages in $B_1$.*

PROOF.    We show that otherwise $\sum_i 1/w_i > h$, that is, $h < h_0(\mathcal{W})$, contradicting the theorem conditions. Observe first that all the pages scheduled so far were granted their exact window. This is true since $k$ was not scheduled yet even-though it was always preferred by greedy over pages that were already scheduled (because $k$ was further from its initial location compared to any page that was already scheduled).

Let $I$ be the set of pages in the instance. Consider $S \subset I$, the set of pages with $w \leq w_k$. The page $k$ itself is not in $S$. We first show that $\sum_{i \in S} 1/w_i = h$. Recall that $\mathcal{W}$ is divisible, thus, for each page $i \in S$, there exists some integer $n_i \geq 1$ such that $w_k = n_i w_i$. Recall that we break ties in favor of small windows so no page from $I \setminus S$ (having window larger than $w_k$) was already scheduled. Since $k$ was not scheduled yet, and since its initial location was $w_k$, the next slot is the $w_k$th slot of the whole schedule. Assume that we are able to schedule all the pages from $B_1 \setminus \{k\}$ on the next slot. We know that all the pages scheduled so far were granted their exact window. In particular, during the first $w_k$ slots of the schedule each of the pages $i \in S$ is scheduled *exactly* $n_i = w_k/w_i$ times. Since there are $h$ channels with no idles, and since all the slots are allocated to pages from $S$, the total number of schedules of pages from $S$ during the first $w_k$ slots of the schedule is $hw_k$. Thus, $\sum_{i \in S} n_i = hw_k$, which implies that $\sum_{i \in S} 1/w_i = h$. Since at least $k \notin S$, we get that for the whole instance $\sum_{i \in I} 1/w_i \geq h + 1/w_k > h$.    □

LEMMA 2.7.    *If $t$ is the $m$th time, $m > 1$, in which $J_k$ reaches $B_1$, there are at most $h - 1$ additional pages in $B_1$.*

PROOF.    Let $P$ be the set of pages in $B_1$ at time $t$. Recall that page $k$ has the widest window request among the pages in $B_1$. Also, since $k$ reaches $B_1$, it must be that $t \geq w_k - 1$. Be definition of the second case implies that all the pages of $P$ were already scheduled at least once.

We show that at time $t - w_k$ all the pages of $P$ were scheduled simultaneously, contradicting the fact that there are only $h$ channels (or contradicting our choice of $t$ as the earliest time in which $B_1$ includes more than $h$ pages, since this was already the case at time $t - w_k - 1$).

Since $k \in B_1$ at time $t$, it must be that $k$ was located in $B_{w_k}$ at time $t - w_k + 1$, meaning that it was scheduled at time $t - w_k$ as we need. Let $j \in P$ be any other page in $P$. Since $\mathcal{W}$ is divisible, $w_k = n_j w_j$ for some integer $n_j \geq 1$. Assume that page $j$ was not scheduled at time $t - w_k$ and let $t - w_k + x$ ($1 \leq x \leq w_j$) be the first time after $t - w_k$ in which page $j$ is scheduled. By the algorithm, during the $w_k - x$ slots preceding $t$, page $k$ was preferred over page $j$ (since $k$ is farer from its initial location compared to $j$). Thus, $j$ is granted an exact $w_j$-window during this time interval. This means that $j$ was scheduled at times $t - w_k + x$, $t - w_k + x + w_j$, ..., $t - w_k + x + (n_j - 1)w_j = t - w_j + x$. Since $1 \leq x \leq$
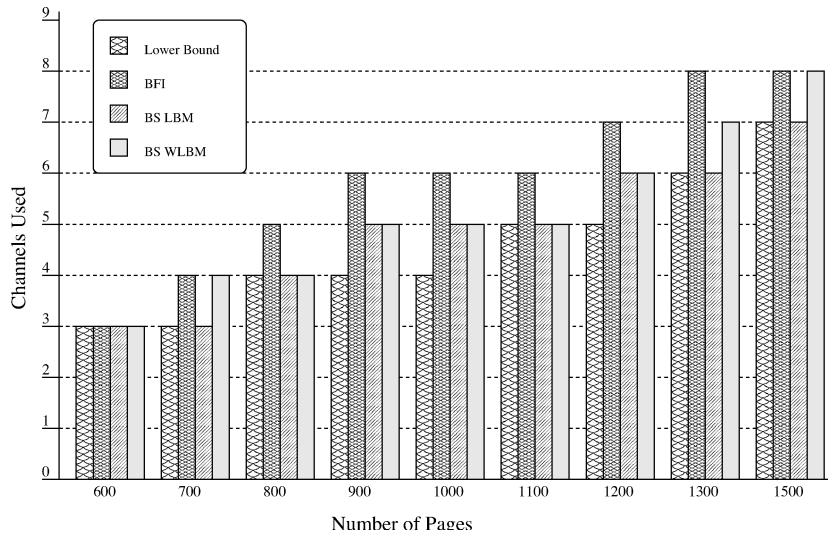
Fig. 1.    Simulation results for random instances.

$w_j$, we conclude that page $j$ was scheduled during the interval $[t - w_j + 1, t]$ contradicting the assumption that page $j$ is in $B_1$ at time $t$.    ☐

## 3. DETERMINISTIC RULES EXPERIMENTS

We simulated the buffer scheme with the deterministic page-selection rules given in Section 2.3. The performance of the buffer scheme, measured by the number of channels required to schedule the pages, was compared for each instance, $\mathcal{W}$, with the lower bound $h_0(\mathcal{W})$ and with the number of channels required by the greedy algorithm, Best-Fit Increasing (*BFI*), given in Bar-Noy and Ladner [2003]. The algorithm BFI schedules the pages in nondecreasing order of their window request. Page $i$ with window request $w_i$ is assigned to a channel that can allocate to it a window $w_i'$ such that $w_i - w_i'$ is nonnegative and minimal. In other words, when scheduling the next page, BFI tries to minimize the lost width $(1/w_i' - 1/w_i)$. Note that BFI produces only perfect schedules.

In our simulations, we considered several classes of instances. The following two are of special interest.

(i) Random—Sequences generated randomly, $w_i$ is chosen randomly in $2, \ldots, 500$ according to the following distribution. Let $S = \sum_{i=2}^{500} i$ then the probability of choosing $w_i = i$ is $i/S$. The simulation results for random instances are shown in Figure 1. The same set of randomly chosen pages was scheduled by the greedy BFI algorithm, by the buffer scheme using the LBM rule and by the buffer scheme using the weighted LBM rule. It can be seen that the buffer scheme always performs better, or not worse, than the greedy algorithm. Also, the buffer scheme is always within one channel from the lower bound (given by $h_0(\mathcal{W})$).

(ii) Harmonic—$\mathcal{H}_n = \langle 1, 2, \ldots, n \rangle$. The simulation results for Harmonic instances is shown in Figure 2. For each number of channels $h = 2, \ldots, 8$ and for
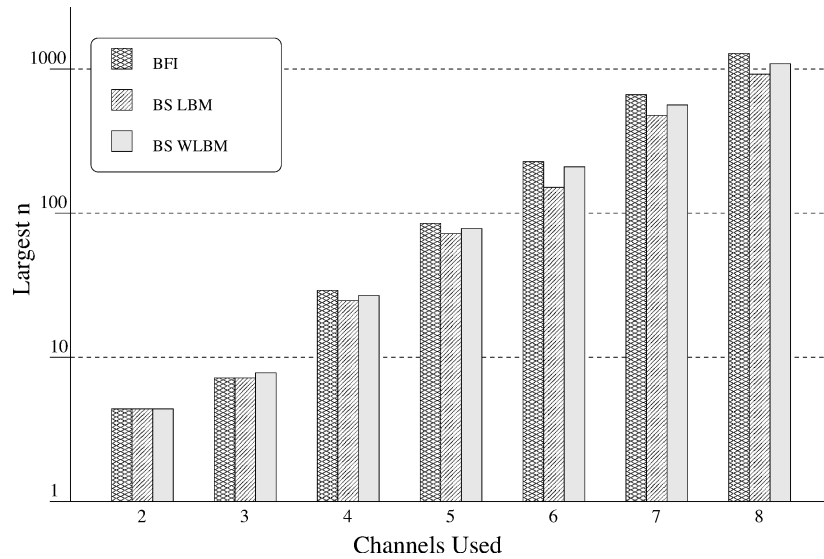
Fig. 2. Simulation results for harmonic instances.

each rule, the maximal $n$ such that $\mathcal{H}_n$ is scheduled successfully is presented. For these instances, the algorithm BFI performs better than any of the deterministic rules of the buffer scheme. The differences though are not significant. In particular, for any harmonic sequence, none of the rules failed on $h_0(\mathcal{W}) + 1$ channels.

## 4. THE EXHAUSTIVE BUFFER SCHEME

In this section, we demonstrate the usefulness of the buffer scheme for practical cases for which it is possible to run an efficient implementation of the scheme that exhausts all possible solutions. Dead-end detection is integrated in the search. It enables early pruning of dead-end states and ensures reasonable cycle-search time. We use the scheme to find the best schedules for some instances and to prove non-trivial impossibility results for other instances.

To obtain our results, we reduce the problem of finding a schedule based on the buffer scheme to the problem of detecting a directed cycle in a finite directed graph. This problem can be solved using standard depth first search (DFS). Consider the directed graph $G$ in which each vertex represents a possible state of the buffers, and there is an edge from $v_1$ to $v_2$ if and only if it is possible to move from the state represented by $v_1$ to the state represented by $v_2$ in one time slot - that is, by scheduling $h$ pages (including all the pages of $B_1$) and updating the resulting page locations as needed. Note that $G$ is finite, since the number of pages is finite and each page has a finite number of potential locations. Now use a standard DFS to detect if there is a directed cycle. If a cycle is detected, then this cycle induces an infinite schedule. If no directed cycle exists, by Theorem 2.2, there is no schedule.

Table I. Some Best Possible Schedules for
Small Number of Segments

| # of Segments | Best Range | Delay |
|---|---|---|
| 5 | 4..8 | $4/5 = 0.8$ |
| 6 | 5..10 | $5/6 = 0.833$ |
| 7 | 5..11 | $5/7 = 0.714$ |
| 8 | 6..13 | $6/8 = 0.75$ |

*Windows Scheduling for Broadcasting Schemes.* The buffer scheme can find for small values of $n$ the minimal $d$ such that there exists a schedule of the instance $\mathcal{W} = \langle d, \ldots, d + n - 1 \rangle$ on $h$ channels. These instances are of special interest for the media-on-demand application since a schedule of $\mathcal{W}$ would imply a broadcasting scheme for $h$ channels with delay guaranteed at most $d/n$ of the media length (using the shifting technique presented in Bar-Noy, Ladner, and Tamir [2003]). In this scheme, the transmission is partitioned into $n$ segments. The trade-off is between the number of segments and the delay. Table I summarizes our simulation results for $n = 5, 6, 7, 8$ segments and a single channel. For each $5 \leq n \leq 8$, we performed an efficient exhaustive search over all possible executions of the buffer scheme. While for some values of $n$ the optimal schedules are perfect and can be generated by simple greedy heuristics, for other values of $n$, the nonperfect schedules produced by the buffer scheme are the only known schedules. This indicates that for some values of $n$ and $d$ the best schedule is not perfect. No existing technique can produce such schedules.

To illustrate that optimal schedules might be non-structured, we present the optimal one-channel schedule for $\langle 5, \ldots, 11 \rangle$. One can verify that this is indeed a feasible schedule. No specific selection rule was applied to produce this schedule, it was generated by exhaustive search over the non-deterministic execution of the buffer scheme:

$[10, 9, 7, 5, 8, 6, 9, 11, 5, 7, 10, 6, 8, 5, 11, 9, 7, 6, 5, 8, 10, 6, 7, 5, 9, 11, 6, 8, 5, 7,$
$10, 9, 6, 5, 7, 8, 11, 5, 6]$.

*Impossibility Results.* Using the buffer scheme, we were able to solve an open problem from Bar-Noy and Ladner [2003] by proving that no schedule exists on three channels for the instance $\mathcal{H}_{10} = \langle 1, \ldots, 10 \rangle$ even though $\sum_{i=1}^{10} 1/i < 3$. Using the early detection of dead-ends we able to reduce the search proving impossibility from 3,628,800 states to only 60,000 states. Using similar techniques we determined that there are no one channel schedules for any of the sequences $\langle 3..7 \rangle$, $\langle 4..9 \rangle$, $\langle 4..10 \rangle$, and $\langle 5..12 \rangle$. This means that the ranges given in the Table I are optimal.

*Arbitrary Instances.* Most of the previous algorithms suggested for WS produce perfect schedules. The buffer scheme removes this constraint. We demonstrate this by the following, one out of many, example. Consider the instance $\mathcal{W} = \langle 3, 5, 8, 8, 8 \rangle$. We first show that there is no perfect schedule for $\mathcal{W}$ on a single channel.

CLAIM 4.1. *There is no perfect schedule for the instance* $\mathcal{W} = \langle 3, 5, 8, 8, 8 \rangle$ *on a single channel.*

PROOF. We denote the pages by their window size. Note that in any perfect schedule 3 cannot be granted a window of 2, since $1/2 + 1/5 + 3/8 > 1$. Thus, 3 must be allocated an exact 3-window. Since $gcd(3, 4) = gcd(3, 5) = 1$, the page 5 must also be allocated an exact 3-window. We are left with three pages that require together $3/8$ of the width, while only $1/3 (< 3/8)$ of the width is still available. □

The exhaustive search and the deterministic buffer scheme with LBM produce the following nonperfect schedule for $\mathcal{W}$:

$$[3, 5, 8_a, 3, 8_b, 5, 3, 8_c, 8_a, 3, 5, 8_b, 3, 8_c, 5, 3, 8_a, 8_b, 3, 5, 8_c, \ldots].$$

We could not find any special pattern or structure in this schedule, suggesting that the only nonmanual way to produce it is by using the buffer scheme.

## 5. EXTENSIONS TO OTHER MODELS

We show how the buffer scheme paradigm can be extended to more general environments. As opposed to other known heuristics for WS, the first two extensions are simple and natural. The extension below are suitable to the objective of minimizing the number of channels. However, with small modifications the buffer scheme can be used to other periodic scheduling problems and objectives. For example, as detailed by Sgall, Shachnai, and Tamir [2005], the buffer scheme combined with an algorithm for finding a cycle with minimum average edge weight, can be used to solve the problem of scheduling with vacations, which is a max-revenue type problem.

*Dynamic Window Scheduling.* In the dynamic (on-line) version of WS, pages arrive and depart over time [Chan and Wong 2004]. This can be supported by the buffer scheme as follows: (1) Any arriving page with window $w_i$ is placed upon arrival in $B_{w_i}$. (2) Any departing page is removed from its current location. The number $h$ of active channels can be adjusted according to the current load. That is, add a new channel whenever the current total width is larger than some threshold (to be determined by the scheme), and release some active channels whenever the current total load is smaller than some threshold.

*Window Scheduling with Jitter.* In this model, each page is associated with a pair of window sizes $(w_i', w_i)$ meaning that $i$ needs to be scheduled *at least* once in any window of $w_i$ time slots, and *at most* once in any window of $w_i'$ time slots. That is, the gap between consecutive appearances of $i$ in the schedule must be between $w_i'$ and $w_i$. In the original WS, $w_i' = 1$ for all $1 \leq i \leq n$. In another extreme, in which $w_i' = w_i$, only perfect schedules are feasible and the gap between any two appearances of $i$ in the schedule is exactly $w_i$. To support such instances with a buffer scheme, we modify the page-selection rules as follows: (1) After scheduling $i$, put it in buffer $B_{w_i}$. (2) Page $i$ can be selected for scheduling only if it is currently located in one of the buffers $B_1, B_2, \ldots, B_{w_i - w_i' + 1}$. This

ensures that at least $w_i'$ slots have passed since the last time $i$ was scheduled. The first selection of $i$ can be from any buffer.

*Pages with Different Lengths.*    In this model, each page is associated with a window $w_i$ and with a length $p_i$. Page $i$ needs to be allocated at least $p_i$ transmission slots in any window of $w_i$ slots. Clearly, $p_i \leq w_i$ for all $1 \leq i \leq n$, otherwise it is impossible to schedule this page. We consider *nonpreemptive* windows scheduling in which for any $i$, the $p_i$ slots allocated to $i$ must be successive. In other words, $i$ must be scheduled nonpreemptively on the channels and the gap between any two beginnings of schedules is at most $w_i$.[1] To support pages with different lengths, each $i$ is represented as a chain of $p_i$ page-segments of length 1. Formally, $i$ is represented by one *locomotive* $i_1$, and $p_i - 1$ *wagons* $i_2, \ldots, i_{p_i}$. We modify the page selection rules in a way that guarantees that all segments of $i$ will be scheduled one after the other. In general, the locomotives are scheduled like in the unit-length scheme and a wagon always follows its locomotive. Specifically,

1. Initially, the $p_i$ segments composing $i$ are located in the $p_i$ buffers $B_{w_i}, B_{w_i+1}, B_{w_i+p_i-1}$. This ensures that at most $w_i$ time slots can elapse before the next schedule of the locomotive, and that all the wagons will follow.
2. When transmitted, every segment of $i$ is moved to buffer $B_{w_i}$.
3. When $h > b_1$ and additional executions can be selected for the next slot, we can only select locomotives, and only those belonging to pages that are not currently processed.[2] For each selected locomotive, $i_1$, we move all the $p_i - 1$ wagons of $i$ into $B_1, B_2, \ldots, B_{p_i-1}$.
4. The calculation of the load on the channels for dead-end detection is done as in the unit-length case. We consider the load generated by all the page segments.

The following theorems extend Theorems 2.1 and 2.2 for instances with arbitrary lengths.

THEOREM 5.1.    *When it does not fail, the buffers scheme produces a feasible schedule for windows scheduling of pages with arbitrary lengths.*

PROOF.    In each iteration, we schedule at most $h$ page segments, since there are $h$ channels this is fine. Consider any page $i$. Recall that initially the segments of $i$ are located in the $p_i$ buffers $B_{w_i}, B_{w_i+1}, B_{w_i+p_i-1}$. For any $i$, in each time slot in which $i_1$ is not selected, it moves one buffer toward $B_1$. All the wagons of $i$ also move one buffer toward $B_1$. After at most $w_i$ time slots $i_1$ must be selected to be scheduled. By rule 3, the remaining wagons of $i$ are now moved to $B_1, \ldots, B_{p_i-1}$, to guarantee that $i$ is scheduled nonpreemptively on $p_i$

---

[1] It is known [Bar-Noy, Ladner, Tamir, and VANDEGRIFT 2005] that *preemptive* WS is equivalent to WS of unit-length pages. Thus, we consider here only the more restricted problem of nonpreemptive scheduling.
[2] We assume here that the next processing of a page cannot start during its current processing. When such parallelism is allowed, we can remove this assumption by temporary duplicating the wagons that still need to be executed.

consecutive slots. Recall that a scheduled wagon is located in $B_{w_i}$—exactly one time slot after the preceding wagon of this page. Thus, all the page-segments of $i$ advance at the same rate toward $B_1$ and will be selected to run at most $w_i$ slots after their previous schedule. □

THEOREM 5.2. *Any feasible schedule for windows scheduling of pages with arbitrary lengths can be generated by the buffer scheme.*

PROOF. The proof is identical to the proof of Theorem 2.2. Since we can follow in each slot the selection of a known legal schedule, it is guaranteed that in the resulting buffer-scheme each page is scheduled in a nonpreemptive manner. □

## 6. CONCLUSIONS

In this article, we showed how any algorithm for the windows scheduling problem can be described as an implementation of the buffer scheme with appropriate scheduling rules. Using a mechanism to detect dead-ends it is possible to efficiently implement the buffer scheme (note that a polynomial time implementation is impossible since the problem is NP-Hard). As a result, we generated the best known schedules for practical instances and proved their optimality via an exhaustive search. We also used the exhaustive search to prove non-trivial impossibility results.

The main advantage of the buffer scheme is its ability to produce nonperfect schedules. Most of the known algorithms (with or without guaranteed performance) produce perfect schedules. However, in the windows scheduling problem and its applications such a restriction is not mandatory. Another evident advantage of the buffer scheme is the ease with which we are able to extend it to more general environments. In particular, the generalization to dynamic instances is promising since previous solutions suffer from being tailored to a static instance of windows. Finally, one of the most important messages of this paper is that the earliest deadline first (EDF) strategy is not the best for the "max" type of periodic scheduling. This is in contrast to its success for the "average" type problems (see [Liu and Layland 1973; Tijdeman 1980]). We developed some understanding that led us to the design of the largest backward move (LBM) strategy.

Some problems remain open, including

- By examining the simulation results of LBM, we strongly believe that better guaranteed bounds for this assignment rule exist. In particular, based on our simulations, we believe that the following is true: for any request set $\mathcal{W}$, LBM never fails on $h_0(\mathcal{W}) + c$ channels, for some small constant $c$.
- We plan to further explore the buffer scheme to come up with better schedules for practical instances. It is interesting to see if there exists a "simple" rule that would outperform LBM.

REFERENCES

ACHARYA, S., FRANKLIN, M. J., AND ZDONIK S. 1995. Dissemination-based data delivery using broadcast disks. *IEEE Personal Communications* 2, 6, 50–60.

AMMAR, M. H. AND WONG, J. W. 1985. The design of teletext broadcast cycles. *Performance Evaluation* 5, 4, 235–242.

ANILY, S., GLASS, C. A., AND HASSIN, R. 1998. The scheduling of maintenance service. *Discrete Applied Mathematics 82*, 27–42.

BAR-NOY, A., BHATIA, R., NAOR, J., AND SCHIEBER, B. 2002. Minimizing service and operation costs of periodic scheduling. *Mathematics of Operations Research 27*, 3, 518–544.

BAR-NOY, A. AND LADNER, R. E. 2003. Windows scheduling problems for broadcast systems. *SIAM Journal on Computing (SICOMP) 32*, 4, 1091–1113.

BAR-NOY, A., LADNER, R. E., AND TAMIR, T. 2003. Scheduling techniques for media-on-demand. In *Proceedings of the 14th SODA,* 791–800.

BAR-NOY, A., LADNER, R. E., AND TAMIR, T. 2007. Windows scheduling as a restricted bin-packing problem. *ACM Trans. Algorithms 3*, 3.

BAR-NOY, A., LADNER, R. E. TAMIR, T., AND VANDEGRIFT, T. 2005. Windows scheduling of arbitrary length jobs on parallel machines. In *Proceedings of the 17th ACM Symposium on Parallel Algorithms and Architectures (SPAA).*

BARUAH, S. K. AND LIN, S.-S. 1998. Pfair scheduling of generalized pinwheel task systems. *IEEE Trans. Comp. 47*, 812–816.

CHAN W. T. AND WONG, P. W. H. 2004. On-line windows scheduling of temporary items. In *Proceedings of the 15th ISAAC*, 259–270.

CHAN M. Y. AND CHIN, F. 1992. General schedulers for the pinwheel problem based on double-integer reduction. *IEEE Trans. Computers 41,* 755–768.

CHAN M. Y. AND CHIN F. 1993. Schedulers for larger classes of pinwheel instances. *Algorithmica, 9*, 425–462.

FEINBERG, E. A., BENDER, M., CURRY, M., HUANG, D., KOUTSOUDIS, T., AND BERNSTEIN, J. 2002. Sensor resource management for an airborne early warning radar. In *Proceedings of SPIE the International Society of Optical Engineering,* 145–156.

HOLTE, R., MOK, A., ROSIER, L., TULCHINSKY, I., AND VARVEL, D. 1989. The pinwheel: A real-time scheduling problem. In *Proceedings of the 22nd Hawaii International Conference on System Sciences,* 693–702.

HOLTE, R., ROSIER, L., TULCHINSKY, I., AND VARVEL, D. 1992. Pinwheel scheduling with two distinct numbers. *Theoretical Computer Science 100*, 105–135.

HUA K. A. AND SHEU, S. 2000. An efficient periodic broadcast technique for digital video libraries. *Multimedia Tools and Applications 10*, 157–177.

JUHN, L. AND TSENG, L. 1997. Harmonic broadcasting for video-on-demand service. *IEEE Trans. on Broadcasting 43*, 3, 268–271.

LIU C. L. AND LAYLAND, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20*, 1, 46–61.

PÂRIS, J. F., CARTER, S. W., AND LONG, D. D. E. 1999. A hybrid broadcasting protocol for video on demand. In *Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking*, 317–326.

PÂRIS, J. F. 2002. A broadcasting protocol for video-on-demand using optional partial preloading. In *Proceedings of the 11th International Conference on Computing I*, 319–329.

SGALL, J., SHACHNAI, H., AND TAMIR, T. 2005. Fairness-free periodic scheduling with vacations. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA),* 592–603.

TIJDEMAN, R. 1980. The chairman assignment problem. *Discrete Mathematics 32*, 323–330.

VISWANATHAN, S. AND IMIELINSKI, T. 1996. Metropolitan area video-on-demand service using pyramid broadcasting. *ACM Multimedia Systems Journal 4*, 3, 197–208.

WEI W. AND LIU, C. 1983. On a periodic maintenance problem. *Operations Res. Letters, 2*, 90–93.