A General Buffer Scheme for the Windows Scheduling Problem

Amotz Bar-Noy[1], Jacob Christensen[2], Richard E. Ladner[2], and Tami Tamir[3]

[1] Computer and Information Science Department, Brooklyn College, 2900 Bedford Avenue Brooklyn, NY 11210
amotz@sci.brooklyn.cuny.edu
[2] Department of Computer Science and Engineering, Box 352350,University of Washington,Seattle, WA 98195
{jacoblc, ladner}@cs.washington.edu
[3] School of Computer Science, The Interdisciplinary Center, Herzliya, Israel
tami@idc.ac.il

Abstract. Broadcasting is an efficient alternative to unicast for delivering popular on-demand media requests. 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 channel 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.

S.E. Nikoletseas (Ed.): WEA 2005, LNCS 3503, pp. 139–151, 2005.
© Springer-Verlag Berlin Heidelberg 2005

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)* [5, 6]. In this paper, 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 new 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 (a schedule in short) 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. We say that a schedule is *perfect* if 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., [21, 15, 16, 18]). The following is a non-perfect 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 since 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 paper we demonstrate that for some instances such "flexible" schedules achieve better performance. Indeed, for the above example, 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 paper 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 non-perfect schedule on a single channel. This schedule was found by efficiently implementing the buffer scheme. Most previous techniques only produce perfect schedules.

1.1 Contributions

The main contribution of this paper 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 [5].

By using an exhaustive-search, we prove impossibility results and find the 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 dead-ends. 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 non-perfect 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 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. (i) *Dynamic (on-line) environment:* pages are arriving and departing on-line 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. (ii) *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. (iii) *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 of 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* [17] and *the chairman assignment problem* [20]. For both problems the Earliest Deadline First strategy was proven to be optimal. Our paper 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 [13, 14] for unit-length pages and was generalized to arbitrary length pages in [8, 12]. 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 [10, 11] 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 [5], 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 ([1]) or TeleText services ([2])) 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., [22, 3]).

WS is known to be NP-hard. In a way, this justifies the efforts of this paper. A proof for the case where i must be granted an exact w_i window is given in [4]. Another proof which is suitable also for the flexible case in which the schedule of i need not be perfect is given in [7].

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 feasible, 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 ℓ_i the *location* of 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 *non-scheduled* 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 advancement 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. 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 1. *When it does not fail, the buffers scheme produces a feasible schedule, and any feasible schedule for WS can be produced by an execution of the buffer scheme.*

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.

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. 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 ℓ_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 in order 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. *Any legal schedule for WS can be generated by the restricted buffer scheme.*

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 (*Largest Backward Move*), pages that can increase ϕ_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 ϕ_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 ℓ_i are selected. This rule is optimal for other periodic scheduling problems that care about average gaps (e.g., periodic scheduling [17] and the chairman assignment problem [20]). 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 $(w_i - \ell_i)$ 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: Put 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 $(w_i - \ell_i)$, break ties in favor of pages with smaller windows.

Optimality for Divisible-size Instances:

Definition 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, in a way that 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 3. *If an instance, $\mathcal{W}$, of WS is a divisible-size instance and $h \geq h_0(\mathcal{W})$, then LBM never fails.*

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 [5]. The algorithm BFI schedules the pages in non-decreasing 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 non-negative 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. In this extended abstract we report about two of them:
(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 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.

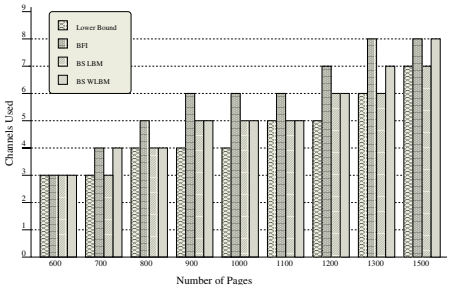

Fig. 1. Simulation results for random instances

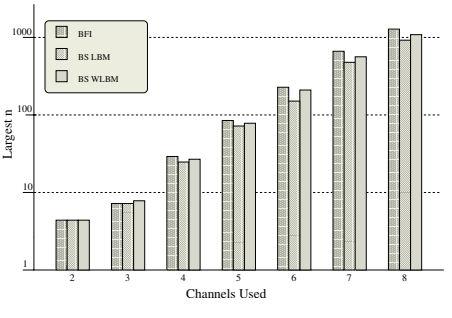

Fig. 2. Simulation results for harmonic instances

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 1, there is no schedule.

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 [6]). In this scheme, the transmission is partitioned into n segments. The trade-off is between the number of segments and the delay. Table 1 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 non-perfect 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 $\langle 5, \ldots, 11 \rangle$. No specific selection rule was ap-

Table 1. 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	$6/8 = 0.75$

plied 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 [5] 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 1 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 $\mathcal{W} = \langle 3, 5, 8, 8, 8 \rangle$. Using the fact that $gcd(3, 8) = gcd(3, 5) = 1$, it can be shown that there is no perfect schedule for $\mathcal{W}$ on a single channel. The exhaustive search and the deterministic buffer scheme with LBM produce the following non-perfect 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 non-manual 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.

Dynamic Window Scheduling: In the dynamic (on-line) version of WS, pages arrive and depart over time [9]. This can be supported by the buffer scheme as follows: (*i*) Any arriving page with window w_i is placed upon arrival in B_{w_i}. (*ii*) 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 the other 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: (*i*) After scheduling i, put it in buffer B_{w_i}. (*ii*) 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 *non-preemptive* windows scheduling in which for any i, the p_i slots allocated to i must be successive. In other words, i must be scheduled non-preemptively 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. Due to lack of space we do not give here the full details of how these page segments are selected one after the other.

References

1. S. Acharya, M. J. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *IEEE Personal Communications*, Vol. 2, No. 6, 50-60, 1995.
2. M. H. Ammar and J. W.Wong. The design of teletext broadcast cycles. *Performance Evaluation*, Vol. 5, No. 4, 235-242, 1985.
3. S. Anily, C. A. Glass, and R. Hassin. The scheduling of maintenance service. *Discrete Applied Mathematics,* Vol. 82, 27-42, 1998.
4. A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling. *Mathematics of Operations Research,* Vol. 27, No. 3, 518-544, 2002.
5. A. Bar-Noy and R. E. Ladner. Windows scheduling problems for broadcast systems. *SIAM Journal on Computing (SICOMP),* Vol. 32, No. 4, 1091-1113, 2003.
6. A. Bar-Noy, R. E. Ladner, and T. Tamir. Scheduling techniques for media-on-demand. *Proc. of the 14-th SODA,* 791-800, 2003.
7. A. Bar-Noy, R. E. Ladner, and T. Tamir. Windows scheduling as a restricted bin-packing problem. *Proc. of the 15-th SODA,* 217-226, 2004.
8. S. K. Baruah S-S. Lin. Pfair Scheduling of Generalized Pinwheel Task Systems *IEEE Trans. on Comp.,* Vol. 47, 812–816, 1998.
9. W. T. Chan and P. W. H. Wong, On-line Windows Scheduling of Temporary Items, *Proc. of the 15th ISAAC*, 259-270, 2004.
10. M. Y. Chan and F. Chin. General schedulers for the pinwheel problem based on double-integer reduction. *IEEE Trans. on Computers,* Vol. 41, 755–768, 1992.
11. M. Y. Chan and F. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica,* Vol. 9, 425–462, 1993.

[1] In a work in progress about WS with arbitrary length pages, we show that *preemptive* WS is equivalent to WS of unit-length pages. Thus, we consider here only the more restricted problem of non-preemptive scheduling.

12. E. A. Feinberg, M. Bender, M. Curry, D. Huang, T. Koutsoudis, and J. Bernstein. Sensor resource management for an airborne early warning radar. In *Proceedings of SPIE The International Society of Optical Engineering,* 145–156, 2002.
13. R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A real-time scheduling problem. In *Proc. of the 22-nd Hawaii International Conf. on System Sciences,* 693–702, 1989.
14. R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel. Pinwheel scheduling with two distinct numbers. *Theoretical Computer Science,* Vol. 100, 105–135, 1992.
15. K. A. Hua and S. Sheu. An efficient periodic broadcast technique for digital video libraries. *Multimedia Tools and Applications*. Vol. 10, 157-177, 2000.
16. L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, Vol. 43, No. 3, 268-271, 1997.
17. C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM,* Vol. 20, No. 1, 46-61, 1973.
18. J.F. Pâris, S. W. Carter, and D. D. E. Long. A hybrid broadcasting protocol for video on demand. *Proc. of the IS&T/SPIE Conference on Multimedia Computing and Networking*, 317-326, 1999.
19. J.F. Pâris. A broadcasting protocol for video-on-demand using optional partial preloading. *XIth International Conference on Computing*, Vol. I, 319-329, 2002.
20. R. Tijdeman. The chairman assignment problem. *Discrete Mathematics,* Vol. 32, 323-330, 1980.
21. S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *ACM Multimedia Systems Journal*, Vol. 4, No. 3, 197-208, 1996.
22. W. Wei and C. Liu. On a periodic maintenance problem. *Operations Res. Letters,* Vol. 2, 90-93, 1983.