Windows Scheduling of Arbitrary Length Jobs on Parallel Machines

Amotz Bar-Noy[*] Richard E. Ladner[†] Tami Tamir [‡] Tammy VanDeGrift[§]

ABSTRACT

The generalized windows scheduling problem for n jobs on multiple machines is defined as follows: Given is a sequence, $I = \langle(w_1, \ell_1), (w_2, \ell_2), \ldots, (w_n, \ell_n)\rangle$ of n pairs of positive integers that are associated with the jobs $1, 2, \ldots, n$, respectively. The processing length of job i is ℓ_i slots (a slot is the processing time of one length unit). The goal is to repeatedly and non-preemptively schedule all the jobs on the fewest possible parallel machines such that the gap (window) between two consecutive executions of the first slot of job i is at most w_i slots. This problem arises in push broadcast systems in which data is transmitted on parallel channels.

The problem is NP-hard even for unit-length jobs and a $(1+\varepsilon)$-approximation algorithm is known for this case by approximating the natural lower bound $W(I) = \sum_{i=1}^{n}(1/w_i)$. The techniques used for approximating unit-length jobs cannot be applied for arbitrary-length jobs mainly because the optimal number of machines might be arbitrarily larger than the generalized lower bound $W(I) = \sum_{i=1}^{n}(\ell_i/w_i)$. Our main result is an 8-approximation algorithm for the generalized windows scheduling problem using new methods, different from those used for the unit-length case. We also present an algorithm that uses $2(1+\varepsilon)W(I) + \log w_{max}$ machines and a greedy algorithm that is based on a new tree representation of schedules. The greedy algorithm is optimal for some special and simulations show that it performs very well in practice.

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

[†]Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195. ladner@cs.washington.edu. Research partially supported by NSF grant CCR-0098012.

[‡]School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. tami@idc.ac.il

[§]Electrical Engineering & Computer Science, University of Portland, 5000 N. Willamette Blvd., Portland, OR 97203. vandegri@up.edu

1. INTRODUCTION

The *generalized windows scheduling problem* for n jobs on parallel machines is defined as follows: Given is a sequence of n positive integer pairs $I = \langle(w_1, \ell_1), (w_2, \ell_2), \ldots, (w_n, \ell_n)\rangle$ that are associated with the jobs $1, 2, \ldots, n$, respectively. The processing length of job i is ℓ_i slots. For simplicity, we assume integer lengths and that each unit of length takes one slot of time in the schedule. The goal is to repeatedly and non-preemptively schedule all the jobs on the fewest possible parallel machines such that the gap (*window*) between any two consecutive executions of the first slot of job i is at most w_i slots.

Example: Let $I = \langle(4, 2), (8, 4), (8, 2), (16, 4), (16, 4)\rangle$ for jobs a, b, c, d, e, respectively. By calculating the total processing requirements of the jobs, it is clear that more than one machine is needed. A possible schedule using 2 machines is the following: $[a, a, c, c, a, a, *, *]$ on one machine and $[b, b, b, b, d, d, d, d, b, b, b, b, e, e, e, e]$ on the other. Schedules are represented by their periodic cycle where the "$*$" symbol stands for an idle slot. Observe that the window between any two appearances of any of any of the five jobs is always exactly as required.

The windows scheduling problem belongs to the class of *periodic scheduling problems* in which n jobs need to be scheduled infinitely often on m parallel machines (the number of machines is sometimes part of the input and not an optimization goal). Each job has a length and is associated with a frequency (or share) requirement. For example, a job might need to be executed one half of the time. The quality of a periodic schedule is measured by the actual frequencies in which the jobs are scheduled, and the regularity of the schedule regarding the gaps between consecutive executions of each job. This distinguishes periodic scheduling from traditional scheduling in which each job is executed only once and is associated with parameters like release time and deadline. The traditional optimization goal for periodic scheduling is an "average" type goal in which job i must be executed a specific fraction of the time. Our problem considers a different objective of a "max" type: the gap between any two consecutive executions of job i must be at most w_i which implies in particular that job i is executed at least ℓ_i/w_i of the time.

Previous results for the windows scheduling problem either assumed one machine with unit-length jobs (the *pinwheel problem* [21, 22]), unit-length jobs with multiple machines ([4, 6]), or one machine with arbitrary-length jobs (the *generalized pinwheel problem* [10, 16]). To the best of our knowledge, we are the first to consider the generalized windows scheduling problem.

1.1 Applications and Motivation

Periodic scheduling in general and windows scheduling in particular can be thought of as a scheduling problem for *push* broadcast systems (as opposed to *pull* broadcast systems). One example is the Broadcast Disks environment (e.g., [1]) where satellites broadcast popular information pages to clients. Another example is the TeleText environment (e.g., [2]) where running banners appear in some television networks. In such systems, there are clients and servers where the servers choose what information to push and in what frequency in order to optimize the quality of service for the clients. This optimization objective belongs to the average type periodic scheduling problems. In a more generalized model (e.g., [7, 19]), the servers "sell" their broadcasting service to various providers who supply content and request that the content be broadcast regularly. The regularity can be defined by a window that represents the maximum delay before a client receives a particular content. This is modelled by the windows scheduling problem.

In *harmonic* windows scheduling (e.g., [5]), jobs represent segments of movies. For $1 \leq i \leq n$, the window of segment i is $w_i = D + i$, where n is the number of equally sized segments of the movie and DL/n is the guaranteed startup delay for an uninterrupted playback of a movie of length L. Harmonic windows scheduling is the basis of many popular media delivery schemes (e.g., [24, 23]) that are based on the concept of receiving data from multiple channels and buffering data for future playback. This concept was first developed in [30] and was the subject of numerous papers in the last decade.

There is an interesting application of the generalized windows scheduling problem in compressed video delivery. When the video is compressed, frame (segment) i is associated with a length ℓ_i measured in time slots. Different compressed frames may have different lengths. Let L be the length of a decompressed frame also in time slots. The entire frame would have to be received before it could be decompressed and played back. Suppose the desired delay to play back the video is D slots. The first frame of length ℓ_1 must be entirely received in every window of size $w_1 = D$. The second frame of length ℓ_2 must be entirely received in every window of size $w_2 = D + L$. In general, the ith frame of length ℓ_i must be entirely received in every window of size $w_i = D + (i - 1)L$.

1.2 Related Work

The pinwheel problem was defined in [22] and the generalized pinwheel problem was considered in [10, 16]. In these and other papers about the pinwheel problem, the focus was to understand which inputs can be scheduled on one machine. For example, [12] optimized the bound on the value of $\sum_{i=1}^{n}(1/w_i)$ that guarantees a feasible schedule [12]. The windows scheduling problem was defined in [4] where schedules were defined that use $opt + O(\ln(opt))$ machines, where *opt* is the number of machines used by an optimal solution.

In [27] periodic scheduling was defined to be a schedule where a job with window w is scheduled exactly once in every time interval of the form $[(k-1)w, kw]$ for any integer k. This is a typical fairness requirement for the average type periodic scheduling. In [28] an optimal solution for the chairman assignment problem was presented for a stronger fairness condition that depends on the prefix of the schedule. These papers considered unit-length jobs on a single machine. In the generalized model, each job has an arbitrary length and it can be scheduled on multiple machines. For this case, [9] proposed *Pfair* schedules in which the number of slots allocated to a job whose share request (measured by the fraction of time it should be processed) is f in any prefix of t time slots is either $\lfloor f \cdot t \rfloor$ or $\lceil f \cdot t \rceil$.

The broadcast disks problem, which is an average type periodic scheduling problem, was introduced in [1]. Unit-length jobs were addressed and constant approximation solutions were presented in [3]. These results were improved in [26] which presented a polynomial time approximation scheme. The arbitrary length case was considered and a constant approximation solution was presented in [25].

In perfect periodic schedules, each job has a fixed window size (a period) between consecutive executions. The objective is to minimize the maximum or average ratio between the granted period and the requested one. This problem differs from windows scheduling since jobs may get larger windows than their request. The unit-length case was considered in [8] and the general case of jobs with arbitrary lengths was considered in [11].

Windows scheduling with unit-length jobs is a special case of the *bin packing* problem (e.g., [13]) and one of our results takes advantage of this fact. In the *unit fractions bin packing* problem, the goal is to pack all the items in bins of unit size where the size of item i is $1/w_i$. In a way, bin packing is the fractional version of windows scheduling where windows scheduling imposes another restriction on the packing. The relationship between these two problems and their off-line and the on-line cases were considered in [6].

Recent work on video-on-demand systems has provided lower bounds on delay required to deliver media that also applies to the special case of windows scheduling as a media delivery scheme [14, 15, 20]. These bounds can be achieved in the limit in the windows scheduling model [5].

1.3 Contributions

Notations and definitions: Denote by w-*job* a job with window w and by (w, ℓ)-*job* a job with window w and length ℓ. An instance in which all the windows and all the lengths are powers of 2 is called a *power*-2 *instance*. The *width* of a (w, ℓ)-job is ℓ/w. $W(I) = \sum_{i=1}^{n}(\ell_i/w_i)$ is the total width of the jobs in $I = \langle(w_1, \ell_1), (w_2, \ell_2), \ldots, (w_n, \ell_n)\rangle$. We consider only *non-preemptive* schedules in which the ℓ_i slots allocated to job i must be successive. We note informally that the *preemptive* version is identical to windows scheduling of unit-length jobs by replacing each (w, ℓ)-job by ℓ $(w, 1)$-jobs. We assume that $\ell_i \leq w_i$ for all $1 \leq i \leq n$. Otherwise, the definition of schedules and windows should be modified. Two special types of schedules are *Perfect schedules* – in which for each job there exists some $w'_i \leq w_i$ such that the gap between any two executions of job i is *exactly* w'_i slots, and *Thrift schedules* – that are perfect schedules in which for all i, $w'_i = w_i$. For an instance I, let $OPT(I)$ denote the number of machines used in an optimal, not necessarily thrift, schedule, and let $OPT_T(I)$ denote the number of machines used in an optimal thrift schedule.

Since a restricted version of the optimal windows scheduling problem is NP-hard even for one channel ([3, 16, 6]), we look for approximate solutions. A natural lower bound to the generalized windows scheduling problem is the total width of the jobs. Since job i requires at least ℓ_i/w_i fraction of a machine, at least $W(I) = \sum_{i=1}^{n}(\ell_i/w_i)$ machines are required in order to accommodate all the jobs. For the unit-length case this lower bound is very close to the optimal solution and indeed $(1+\varepsilon)$-approximation solutions exist for small values of ε. The following example demonstrates that this lower bound can be arbitrarily far from the optimal solution for the generalized windows scheduling problem.

Example: Let $I = \langle(r, 1), (r^2, r), \ldots, (r^n, r^{n-1})\rangle$ be an instance consisting of n jobs where $r \geq 2$ is an integer. It is not hard to see that no two jobs can be executed on the same machine and therefore any feasible schedule must use at least n machines. On the other hand, each job demands $1/r$ of a machine for a total demand of n/r machines. Thus, the ratio between the optimal solution and this lower bound is at least r which can be arbitrarily large.

We develop new approximation algorithms that are based on novel methods and techniques. We consider two special cases: (i) In thrift schedules the gap between two executions of job i must be exactly w_i (in non-thrift schedules jobs may be scheduled more frequently). (ii) In power-2 instances, windows and lengths are powers of 2. This case can be optimally solved for unit-length jobs, but complex and interesting problems arise in the general case. In particular, the thriftiness paradox presented in this paper implies that even for power-2 instances it might be useful to schedule some jobs more frequently than their demand in order to use fewest machines.

For thrift schedules of power-2 instances, we present an optimal algorithm. This algorithm serves as the basis for an 8-approximation algorithm for the generalized windows scheduling problem after rounding both the windows and the lengths to power of 2 values. We also present an algorithm that uses $2(1+\varepsilon)W(I) + \log w_{max}$ machines and a greedy algorithm that is based on a tree representation of schedules. This greedy algorithm is evaluated by simulation, and performs very well in practice. A variant of this algorithm is optimal for thrift schedules of power-2 instances.

2. PRELIMINARIES

2.1 Hardness Proof

The windows scheduling problem for unit-length jobs is known to be NP-hard. For this case, an optimal algorithm exists when all the w_i's are powers of 2. By contrast the generalized problem is strongly NP-hard even if all the w_i's are powers of 2.

THEOREM 2.1. *The generalized windows scheduling problem is strongly NP-hard even if all the windows are powers of* 2.

PROOF. We show a reduction from 3-*partition*, which is strongly NP-hard [18]. An instance of 3-partition is defined as follows.
Input: A finite set A of $3m$ elements, a number $B \in Z^+$, and a size $s(x)$ for each $x \in A$, such that each $s(x)$ satisfies $B/4 < s(x) < B/2$ and such that $\sum_{x\in A} s(x) = B$.
Output: Is there a partition of A into m disjoint sets, $S_1, S_2, \ldots, S_m$, such that, for $1 \leq i \leq m$, $\sum_{x\in S_i} s(x) = B$? (Note that the above constraints on the element sizes imply that every such S_i must contains exactly three elements from A).

Given an instance of 3-partition, we construct an input, I, for windows scheduling, such that all the windows in I are powers of 2 and I has a schedule on one machine if and only if A has a 3-partition.

Let $W > B$ be a power of 2, and let $k > m$ be a power of 2. For each $x \in A$ there is an item with parameters $(kW, s(x))$ in I. In addition, I includes one item z with parameters $(W, W - B)$, and $k - m$ dummy items, $d_1, d_2, \ldots, d_{k-m}$ with parameters (kW, B). If A has a partition then I has the following schedule: $[z, \mathcal{S}_1, z, \mathcal{S}_2, z, \ldots, \mathcal{S}_m, z, d_1, z, d_2, \ldots, z, d_{k-m}]$, where $\mathcal{S}_i$ is a consequent schedule of the items originated from S_i in arbitrary order. Since $\sum_{x\in S_i} s(x) = B$, the window of z is $\ell_z + B = W - B + B = W$, as needed. Also, the window of each of the other items is kW, since the total length of the items in this schedule is $\sum_{x\in A} s(x) + k\ell_z + (k - m)B = mB + k(W - B) + (k - m)B = kW$.

Note that $\sum_{i\in I} \ell_i/w_i = 1$, thus, any schedule of I on one machine must be thrift. In particular, if I has a schedule on one machine then the schedule of z must be with an exact W-window. This means that the k idle intervals of B slots between the first $k + 1$ executions of z induce a partition of A. Specifically, $k - m$ of these intervals are allocated to the dummy items and the remaining m intervals induce a partition. □

2.2 The Thriftiness Price

For an instance I, the *thriftiness price* is defined as the ratio $OPT_T(I)/OPT(I)$. We show that sometimes the thriftiness price can be very high. It means that thrift schedules allocate the fewest possible number of slots to jobs, but on the other hand, they might require many more machines. In fact, even for unit-length jobs the thriftiness price is not bounded. For a desired ratio r, consider an instance with r requests such that $w_i = p_i$ and $\ell_i = 1$ where $p_1, \ldots, p_r$ are r distinct primes greater than r. A thrift schedule must use r machines since jobs with relatively prime windows cannot be scheduled on the same machine. On the other hand, the simple round-robin schedule is a non-thrift schedule on one machine. It is feasible since the granted window for job i is r which is smaller than the required window w_i.

For power-2 windows and unit-length jobs, a known optimal algorithm for a thrift schedule ([4]) uses $OPT(I)$ machines, thus the thriftiness-price ratio is 1. Also, for power-2 instances, two polynomial time optimal algorithms that use $OPT_T(I)$ machines are given in this paper. However, even for power-2 instances, we can sometimes gain from scheduling a job with a window smaller than its demand. The problem of finding a schedule in $OPT(I)$ machines for power-2 instances remains open. Moreover, we do not know if the problem is NP-hard. The following example demonstrates this "paradox".

Paradox Example: Consider an instance consisting of one job $z = (4, 1)$ and five $(16, 2)$-jobs a, b, c, d, e. A perfect non-thrift schedule (of length 15) for this instance is: $[z, a, a, z, b, b, z, c, c, z, d, d, z, e, e]$. Job z is granted a window 3 and each of the other five jobs is granted window 15.

However, no 1-machine schedule in which the window size of z is 4 exists because only one $(16, 2)$-job can be scheduled between any two z's that are four slots apart. In any 16 consecutive slots we have four such holes and five $(16, 2)$-jobs to schedule.

The next two theorems show that the above example can be extended for any number of machines h, and that on the other hand, this 2-ratio (two machines instead of one machine in the example) is tight.

THEOREM 2.2. *If I is a power-2 instance, then $OPT_T(I) \leq 2 \cdot OPT(I)$.*

PROOF. Given a schedule of I on h machines, construct a thrift schedule of I on $2h$ machines. In particular, for the optimal schedule of I we get the statement of the theorem.

The construction is per-machine, that is, given a machine, M, that processes the set of jobs $I_M = \{w_i, \ell_i\} \subseteq I$, we show that the optimal thrift algorithm, $\mathcal{A}_T$, will schedule this set of jobs *thriftily* on at most two machines. Since the jobs of I_M are scheduled on a single machine it is known that $\sum_{i \in I_M} \ell_i/w_i \leq 1$, and also, for any job i in I_M, $\ell_i < w_{min}(I_M)$. In other words, the length of any job in I_M is less than the minimal window of a job in I_M. This is true since otherwise, these two jobs cannot be assigned to the same machine - as job i must be allocated ℓ_i consequent slots, leaving no slot for a job with w_{min} in this segment.

Consider the execution of $\mathcal{A}_T$ on I_M. Observe first that we will never have dedicated machines to a (w, w)-jobs. This is true since grouped jobs have the length of the longest job in the group, which is by the above, always less than w_{min}, and thus also less than the current considered window size. Thus, all the jobs will be packed in the last iteration. Let $w_{max} = 2^k w_{min}$. When moving from iteration i to iteration $i+1$, $\mathcal{A}_T$ might add a dummy job of window $w_{max}/2^i$ and length at most $w_{min}/2$ (by the above, this is the maximal possible length of any job). The width of this additional job is $(w_{min}/2)/(2^{k-i} w_{min}) = 1/2^{k-i+1}$. Therefore, along the whole execution, as i is increased from 0 to $k-1$, the total width added by dummy jobs is at most $1/2^{k+1} + \ldots + 1/8 + 1/4 < 1$. Since these are the only dummy jobs added, and $\mathcal{A}_T$ packs optimally all the jobs of the last iteration (all having window w_{min}), the total number of machines used is it most $\lceil H(I_M) + H(\text{dummy jobs}) \rceil \leq \lceil 1 + 1 \rceil \leq 2$. $\Box$

THEOREM 2.3. *For any integer h, there exists a power-2 instance I such that $OPT(I) = h$ and $OPT_T(I) = 2h$.*

PROOF. For any $i = 0, 4, 8, 4k, \ldots$, define the instance I_i consisting of six jobs: a single $(2^{i+2}, 2^i)$-job, denoted z_i, and five $(2^{i+4}, 2^{i+1})$-jobs, denoted a_i, b_i, c_i, d_i, e_i. For example, $I_0 = \{(4, 1), (16, 2), (16, 2), (16, 2), (16, 2), (16, 2)\}$ which is the instance from the paradox example above, and $I_4 = \{(64, 16), (256, 32), (256, 32), (256, 32), (256, 32), (256, 32)\}$.

For a given h, the instance I_h^* consists of a union of any h different instances from the above set of instances (say, the first h). An important observation is that jobs from I_i and I_j for $i > j$ cannot be scheduled on the same machine. This is true since the length of any job in I_i is at least the window of any job in I_j.

CLAIM 2.4. *For any i, there is a non-thrift schedule of I_i on one machine.*

PROOF. The following is a non-thrift perfect schedule for $a_i, b_i, c_i, d_i, e_i, z_i$: $[z_i, a_i, z_i, b_i, z_i, c_i, z_i, d_i, z_i, e_i]$, where each appearance of z_i is for 2^i slots and each appearance of one of the other five jobs a_i, b_i, c_i, d_i, e_i is for 2^{i+1} slots. The window of z_i is therefore $2^i + 2^{i+1} < 2^{i+2}$, and the window of the other five jobs a_i, b_i, c_i, d_i, e_i is $5(2^i + 2^{i+1}) < 2^{i+4}$. $\Box$

CLAIM 2.5. *For any i, there is no thrift schedule of I_i on one machine.*

PROOF. Note that only one $(2^{i+4}, 2^{i+1})$-job can be scheduled between any two consecutive schedules of $z = (2^{i+2}, 2^i)$. In any 2^{i+4} consecutive slots we have four such holes and five $(2^{i+4}, 2^{i+1})$-jobs to schedule. Thus, an additional machine must be used. $\Box$

Combining the above claims with the fact that jobs from different I_i's cannot be scheduled on the same machine, yields the 2-ratio.

2.3 Uniform Lengths or Uniform Windows

If all the jobs have the same length then the problem can be reduced to the unit-length case. Let ℓ be the uniform length of all jobs. If all windows are multiples of ℓ then it is possible to replace all $(k\ell, \ell)$-jobs by $(k, 1)$-jobs. The resulting instance, I', has unit lengths and any schedule of it can be "stretched" by a factor of ℓ to produce a schedule of the original instance. Also, each schedule of the original instance induces a schedule of I'. For arbitrary windows, it is possible to round down a window of size $k\ell + r$, $r < \ell$ to be $k\ell$ without hurting the solution. A stretching and shifting argument is used to prove the following:

THEOREM 2.6. *Let I be an instance in which all jobs have the same length ℓ. Let I' be the instance obtained from I by replacing each $(k\ell + r, \ell)$-job $(0 \leq r < \ell)$ by a $(k\ell, \ell)$-job. Then $OPT(I) = OPT(I')$.*

If all jobs have the same window, w, then the problem is reduced to *Bin-packing with discrete sizes*. Formally, items of sizes in $\{1/w, 2/w, \ldots, w/w\}$ are to be packed in a minimal number of bins of size 1, where a (w, ℓ)-job is represented by an item of size ℓ/w. Using the known APTAS for bin-packing [29], we get an APTAS for this special case of windows scheduling.

3. OPTIMAL THRIFT SCHEDULE OF POWER-2 INSTANCES

We present an optimal algorithm for thrift schedules of instances in which all the w_i's and ℓ_i's are powers of 2. The algorithm, denoted $\mathcal{A}_T$, schedules all the jobs in a minimal number of parallel machines. Let w_{min} and w_{max} denote the minimal and maximal windows in I. The algorithm produces a schedule of length w_{max} (to be repeated cyclically). We use the following property of thrift schedules of jobs with power-of-2 windows:

CLAIM 3.1. *In any thrift schedule, for each of the machines, if a w-job is scheduled on slot x, then slot $x + w/2$ on this machine is idle or allocated to a job having window at least w.*

PROOF. Consider a w_i-job with $w_i < w$. We show that job i cannot be scheduled on slot $x + w/2$. Since all windows are powers of 2, $w_i = w/2^j$ for some $j > 0$. Thus, if job i is scheduled on slot $x + w/2$, it must be scheduled also on slot x, which is occupied by the w-job. $\Box$

Algorithm $\mathcal{A}_T$: Assume $w_{max} = 2^k w_{min}$. The algorithm proceeds in three phases.

Phase 1: The first phase of the algorithm consists of k iterations. In the first iteration, the algorithm considers the w_{max}-jobs. Some of these jobs are scheduled and the rest are replaced by $(w_{max}/2)$-jobs. The set of non-scheduled jobs (original jobs and the newly created $(w_{max}/2)$-jobs), are moved to the next iteration. Generally, let I_i denote the set of jobs that are not scheduled before iteration i, where i goes from 0 to $k-1$, in particular, $I_0 = I$. In iteration i, $\mathcal{A}_T$ schedules some of the $(w_{max}/2^i)$-jobs on h_i machines, and replaces the rest of the $(w_{max}/2^i)$-jobs by $(w_{max}/2^{i+1})$-jobs. This way, in I_i, all the jobs have window at most $w_{max}/2^i$.

We now describe the way I_{i+1} is built from I_i. Let M be the set of jobs having the maximal window, $w = w_{max}/2^i$, in I_i. $\mathcal{A}_T$ first schedules each (w, w)-job on a dedicated machine. Let h_i be the number of these jobs. From the remaining jobs, $\mathcal{A}_T$ constructs the instance I_{i+1} as follows: Sort the jobs of M such that $\ell_1 \geq \ell_2 \geq \ldots$. Let j be such that $\ell_1 = \ell_2 + \ell_3 + \ldots + \ell_j$. If no such j exists, it must be that $\ell_1 > \ell_2 + \ell_2 + \ldots + \ell_{|M|}$ (because each ℓ_i is a power of 2) and all the jobs of M are replaced by one $(w/2, \ell_1)$-job. If such a j exists, $\mathcal{A}_T$ replaces the j jobs with one $(w/2, \ell_1)$-job, and continues in the same way with the rest of M. In addition, all the jobs of I_i having window smaller than w are moved to I_{i+1}.

Phase 2: Recall that in I_i all the jobs have windows at most $w_{max}/2^i$, thus, all jobs in I_k have windows at most $w_{max}/2^k = w_{min}$. In other words, I_k consists of w_{min}-jobs. In the second phase of the algorithm, $\mathcal{A}_T$ schedules I_k *optimally* on $h' = \lceil \sum_{(w,\ell)\in I_k} \ell / w_{min} \rceil$ machines by partitioning them into h' sets such that the total length of the jobs in each set is at most w_{min}. Since ℓ is a power of 2 and $\ell \leq w_{min}$ for all $(w, \ell) \in I_k$, such a partition exists and can be found by any 'any fit' algorithm that considers the jobs in non-increasing order of their lengths. Given such a partition, $\mathcal{A}_T$ allocates one machine to each of the h' sets and schedules the jobs of each set sequentially and thriftily on this machine. The length of this optimal schedule of I_k is w_{min}.

Phase 3: During the third phase of the algorithm, after scheduling optimally I_k, $\mathcal{A}_T$ backtracks to schedule the original set of jobs, I. This phase consists of k iterations. In iteration i, $i = k, k-1, \ldots, 1$, $\mathcal{A}_T$ moves from a schedule of I_i of length $w_{max}/2^i$ to a valid thrift schedule of I_{i-1} of length $w_{max}/2^{i-1}$. Given a schedule of length $w_{max}/2^i$ of I_i, repeat it to get a schedule of double length. The jobs with window smaller than $w_{max}/2^{i-1}$ were not modified in the move from I_{i-1} to I_i and therefore they are legally scheduled. In the doubled schedule, every $(w_{max}/2^i)$-job appears twice. Some of the $(w_{max}/2^i)$-jobs in I_i originate from $(w_{max}/2^{i-1})$-jobs in I_{i-1}. Each such job of length ℓ originates from one $(w_{max}/2^{i-1}, \ell)$-job, j, and a set, B_j of $(w_{max}/2^{i-1})$-jobs with total length at most ℓ. In the double-length schedule, replace the first appearance of this job by j and the second appearance by B_j and some idle slots such that the total length of B_j and the idle slots is ℓ. This process is done for each of the grouped $(w_{max}/2^i)$-jobs and for each machine in the schedule of I_i. The resulting schedule is a feasible thrift schedule of I_{i-1} of length $w_{max}/2^{i-1}$.

Example execution of $\mathcal{A}_T$ without dummy jobs: Consider the instance $I = \langle a = (4,1), b = (8,2), c = (8,1), d = (8,1), e = (16,2), f = (16,2), g = (16,16)\rangle$. It has $w_{max} = 16$. $\mathcal{A}_T$ first dedicates one machine to job g. Next, it replaces e and f by $e' = (8,2)$. The remaining instance is $I_1 = \langle a = (4,1), b = (8,2), c = (8,1), d = (8,1), e' = (8,2)\rangle$ in which $w = w_{max}/2 = 8$. In the second iteration, $\mathcal{A}_T$ replaces b and e' by $b' = (4,2)$, and c and d by $c' = (4,1)$. The remaining instance is $I_2 = \langle a = (4,1), b' = (4,2), c' = (4,1)\rangle$ in which $w = w_{max}/4 = w_{min} = 4$. That is, all the jobs have $w = 4$ and $\sum_{(4,\ell)\in I_2} \ell/4 = 1$. $\mathcal{A}_T$ now constructs the 1-machine schedule $[a, c', b', b']$. Next, it retrieves a schedule of I from the schedule of I_2. This is done by 'opening' the groups, first to get a schedule of I_1: $[a, c, b, b, a, d, e', e']$ and again, to get the final schedule $[a, c, b, b, a, d, e, e, a, c, b, b, a, d, f, f]$. Together with the machine that processes g, this is an optimal two-machine schedule of I.

Example execution of $\mathcal{A}_T$ with dummy jobs: Consider the instance $I = \langle a = (2,1), b = (4,2)\rangle$. It has $w_{max} = 4$, and no additional 4-job exists to be grouped with b. Thus, b is replaced by $b' = (2,2)$ to get $I_1 = \langle a = (2,1), b' = (2,2)\rangle$. All the jobs have a 2-window. $(2+1)/2 < \lceil (2+1)/2 \rceil = 2$, so one dummy job, $d = (2,1)$, is added and $\mathcal{A}_T$ partitions I_1 into two sets, each to be scheduled on one machine. Specifically, one machine for b' and one for a and d. Next, to get a schedule of I, $\mathcal{A}_T$ opens the b' group to get $[b, b, *, *]$ ($*$ denotes idle), and replaces the dummy job by idle slots, to get $[a, *, a, *]$ on the second machine.

Example 3: Consider the instance $I = \langle a = (4,2), b = (8,2), c = (8,2), d = (8,4), e = (8,4)\rangle$. It has $w_{max} = 8$. First, $\mathcal{A}_T$ replaces d and e by $d' = (4,4)$. That is, $I_1 = \langle a = (4,2), b = (8,2), c = (8,2), d' = (4,4)\rangle$. One machine is dedicated to the new job d' and $\mathcal{A}_T$ continues with a, b, c. The jobs b and c are replaced by $b' = (4,2)$. Thus, $I_2 = \langle a = (4,2), b' = (4,2)\rangle$. Now all the jobs have the same window $w = 4$. and an optimal schedule is $[a, a, b', b']$. Next, $\mathcal{A}_T$ doubles this schedule to get the schedule $[a, a, b, b, a, a, c, c]$ of I_1 and doubles the schedule of the d'-machine to get the schedule $[d, d, d, d, e, e, e, e]$. These two machines together form a schedule of $I_0 = I$.

THEOREM 3.2. *For any power-2 instance, I, $\mathcal{A}_T$ schedules I on $OPT_T(I)$ machines.*

Proof sketch: Based on Claim 3.1, it can be shown that for all i, $0 \leq i \leq k-1$, if I_i has a feasible schedule on h machines, then I_{i+1} has a feasible schedule on $h - h_i$ machines. In particular this implies that $OPT_T(I_{i+1}) \leq OPT_T(I_i) - h_i$. Recall that h_i is the number of machines dedicated in iteration i to $(w_{max}/2^i, w_{max}/2^i)$-jobs. Combine the above with the observation that I_k is packed optimally (formally, $OPT_T(I_k) = h'$), we get that the number of bins used by $\mathcal{A}_T$ is

$$\begin{aligned}\sum_{i=0}^{k-1} h_i + h' &\leq \sum_{i=0}^{k-1} (OPT_T(I_i) - OPT_T(I_{i+1})) + h' \\ &= OPT_T(I_0) - OPT_T(I_k) + h' \\ &= OPT_T(I_0) \ .\end{aligned}$$

∎

4. APPROXIMATION ALGORITHMS FOR ARBITRARY INSTANCES

Given a general input, we can reduce each window w_i to the nearest power of 2, and schedule separately (as a bin-packing problem, see Section 2.3) all the jobs whose windows were rounded to 2^u. The rounding produces $\log w_{max}$ uniform-window instances. When using an APTAS for the induced bin-packing problem [29], this approach gives a general algorithm that uses $2(1+\varepsilon)W(I) + \log w_{max}$ machines. We omit the details. We now present our main result: an 8-approximation algorithm.

Let I be an arbitrary instance. Let J' be the instance obtained from I by rounding the lengths down to powers of 2 and rounding the windows up to powers of 2. Clearly, J' is a power-2 instance. Note that J' is *easier* than I. In other words, every schedule for I induces a valid schedule for J', by allocating to each job of J' the slots allocated to the corresponding job in I. In particular, $OPT(J') \leq OPT(I)$. Let J be the power-2 instance obtained from J' by replacing each (w, ℓ)-job by a $(w/2, 2\ell)$-job. If $w/2 < 2\ell$ then the (w, ℓ)-job of J' contributes a (w, w)-job to J'. Note that each (w, ℓ)-job in I is represented in J by a (w', ℓ')-job such that $w' \leq w$ and $\ell' \geq \ell$, therefore, the instance I is *easier* than J meaning that every schedule for J induces a valid schedule for I. This is also valid for the jobs with $w/2 < 2\ell$: being replaced by a (w, w)-job, each such job will be allocated a machine - which is clearly sufficient.

Algorithm $\mathcal{A}$: Execute the algorithm $\mathcal{A}_T$, which is optimal for power-2 instances, to find an optimal thrift schedule of J, the hardest instance among the three. The resulting schedule induces a valid (perfect but not necessarily thrift) schedule of I.

In order to analyze the approximation ratio of $\mathcal{A}$, we first bound the cost of doubling the job lengths and the cost of dividing all windows by 2 in a power-2 instance.

The Cost of Doubling the Lengths: For a power-2 instance J', consider the instance J'' obtained from J' by replacing each (w, ℓ)-job by a $(w, 2\ell)$-job. In other words, each job in J' contributes to J'' a job with the same window and a doubled length. Clearly, J'' is also a power-2 instance. Note that if $w = \ell$, then a non-feasible $(w, 2w)$-job is created. To avoid this problem, a (w, w)-job in J' contributes to J'' one identical (w, w)-job. However, since we give an upper bound on $OPT_T(J'')$, we can assume w.l.o.g. that such jobs do not exist.

Claim 4.1. *$OPT_T(J'') \leq 2 \cdot OPT_T(J')$.*

Proof sketch: Given a thrift schedule of J' on h machines, construct a thrift schedule of J'' on $2h$ machines. Note that for the optimal schedule of J' we get the statement of the claim. The construction is per-machine, that is, given *one* machine that processes thriftily a subset of jobs of J', it must be that $\mathcal{A}_T$ uses a single machine to schedule those jobs. It can be shown that $\mathcal{A}_T$ uses at most two machines in order to schedule the corresponding subset of jobs of J''. If J' consists of a single (w, w)-job, then the corresponding (w, w)-job in J'' is scheduled on a single machine. ■

The Cost of Dividing the Windows by 2: For a power-2 instance J', consider the instance J'' obtained from J' by replacing each (w, ℓ)-job by a $(w/2, \ell)$-job. In other words, each job in J' contributes to J'' a job with the half-size window and the same length. By definition, J'' is also a power-2 instance. Note that if $w = \ell$, then a non-feasible $(w/2, w)$-job is created. To avoid this problem, a (w, w)-job in J' contributes to J'' one identical (w, w)-job. In fact, since we look for an upper bound on $OPT_T(J'')$, we can assume without loss of generality that such jobs do not exist. In addition, since the only possible 1-jobs are $(1, 1)$-jobs, the above exception includes also 1-jobs, and therefore J'' is well-defined.

Claim 4.2. *$OPT_T(J'') \leq 2 \cdot OPT_T(J')$.*

Proof sketch: Let $J'' = \langle (w_i, \ell_i) : 1 \leq i \leq n \rangle$ and $J' = \langle (2w_i, \ell_i) : 1 \leq i \leq n \rangle$. Consider the instance $K = \langle (2w_i, 2\ell_i) : 1 \leq i \leq n \rangle$. By Claim 4.1, $OPT_T(K) \leq 2 \cdot OPT_T(J')$. We show that $OPT_T(J'') \leq OPT_T(K)$. Given a thrift schedule of K, the idea is to construct a schedule of J'' by compressing it by a factor of 2. Assume that time slots are indexed $1, 2, \ldots$. It can be shown that there exists an optimal schedule of K in which all schedules of all jobs begin in odd-indexed slots. Given such a schedule of K, for every $t \geq 1$, the slots $2t-1, 2t$ either process the same (even-length) job, or are both idle. Therefore, it is possible to compress this schedule, by taking just the odd slot out of each such pair. The resulting instance is a schedule of J''. ■

Theorem 4.3. *For any instance I, $\mathcal{A}$ schedules I on at most $8 \cdot OPT(I)$ machines.*

Proof sketch: Combine Claims 4.1 and 4.2, to get $OPT_T(J) \leq 4OPT_T(J')$. An additional factor of 2 is due to the thrift price for power-2 instances (Theorem 2.2). That is, $OPT_T(J) \leq 4OPT_T(J') \leq 8OPT(J')$, Finally, since J' is an easier instance than I we have, $OTP_T(J) \leq 8OPT(I)$. ■

5. A PRACTICAL ALGORITHM

We present a greedy algorithm for the generalized windows scheduling problem, the output of which is a perfect, but not necessarily thrift, schedule. For arbitrary instances, the algorithm is evaluated by a simulation, according to which it performs very close to the optimal (see Section 5.3). A variant of this algorithm for thrift schedules of power-2 instances is proved to be optimal. Due to space limitation we do not detail this variant, we only note that the optimal schedules it produces are not necessarily identical to those produced by $\mathcal{A}_T$. In overview, the greedy algorithm is similar to other *fit* packing algorithms. The algorithm sorts the jobs according to some deterministic rule (breaking ties arbitrarily) and then jobs are scheduled one after the other according to the sorted order. Each job is scheduled on one of the already open machines that can process it, and in the case there is no such machine, a new machine is added, and the job is scheduled on it. The machine selection rule for generalized window scheduling is more involved than is usually found in solving other problems (like bin-packing) with a similar strategy. In particular, after the machine is selected, it is determined in which slots the job will be scheduled. In the following we use *directed trees* to represent the state of the machines and describe the algorithm formally.

5.1 Tree Representation of Perfect Schedules

Each machine is represented by a directed tree. Every node in the tree is labelled with a window w and a length ℓ, representing a periodic (w, ℓ)-schedule on the machine. Each leaf might be *closed* or *open*. A closed (w, ℓ)-leaf is associated with a (w', ℓ')-job scheduled on this machine. In this case $w \leq w'$ and $\ell \geq \ell'$. An open (w, ℓ)-leaf is associated with a (w, ℓ)-periodic idle of the machine (an idle of ℓ slots repeated with window w). For example, the tree in Figure 1 represents a schedule of the instance $I = \{a = (4, 1), b = (8, 1), c = (8, 1), d = (8, 2), e = (16, 2), f = (16, 2).\}$.

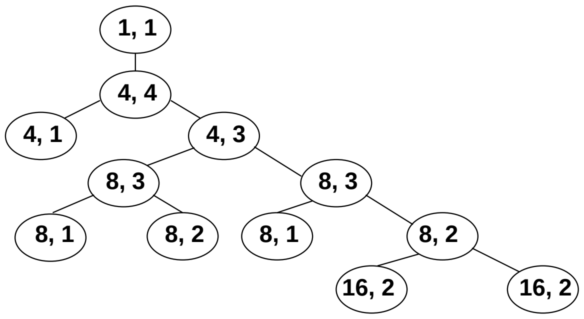

Figure 1: Tree representation of a 1-machine schedule

Initially, the machine is idle. The associated tree has a single node - a $(1, 1)$-leaf, meaning we can schedule a job with window 1 and length 1 on this machine. An open (w, ℓ)-leaf can be split into multiple leaves as follows:

1. Split into k open (wk, ℓ)-leaves. For example, a $(4, 2)$-leaf can split into three $(12, 2)$-leaves.

2. Split into k leaves $\langle(w, \ell_1), (w, \ell_2), \dots, (w, \ell_k)\rangle$ such that $\sum_{i=1}^{k} \ell_i = \ell$. For example, a $(12, 8)$-leaf can split into $(12, 5), (12, 2)$ and $(12, 1)$.

Also, for any w, a $(1, 1)$-leaf (the root of the tree) can be replaced by a (w, w)-leaf.

These rules imply a straightforward deterministic mapping of trees into a schedule. The schedule is defined recursively. The base case is the (w, w)-root representing an idle schedule of length w. A (w, ℓ)-node that splits into k (wk, ℓ)-children represents a round-robin schedule on the children schedules, each allocated ℓ slots in any window of wk slots. A (w, ℓ)-node that splits into k nodes $\{(w, \ell_1), (w, \ell_2), \dots, (w, \ell_k)\}$ such that $\sum_{i=1}^{k} \ell_i = \ell$ represents a round-robin schedule on the children schedules, where child i is allocated ℓ_i slots in every window of w slots. The schedule represented by the tree in Figure 1 is $[a, b, d, d, a, c, e, e, a, b, d, d, a, c, f, f]$.

5.2 The Greedy Algorithm

In the first stage of the algorithm the jobs are sorted in non-decreasing order by their window size, that is, $w_1 \leq w_2 \leq \dots \leq w_n$. Jobs having the same window and different lengths are sorted in non-increasing order by their lengths. That is, the w-jobs are sorted such that $\ell_1 \geq \ell_2 \geq \dots$. For every two jobs (w_1, ℓ_1) and (w_2, ℓ_2), the first job comes before the second one if $w_1 < w_2$ or $w_1 = w_2$ and $\ell_1 \geq \ell_2$. After sorting, the algorithm schedules the jobs one after the other according to the sorted order. Let (w, ℓ) be the next job to be scheduled. A (w, ℓ)-job can be scheduled on any (w', x)-leaf such that $w' \leq w$ and $x \geq \ell$. Moreover, if $w' = kw''$ and $w'' \leq w$, a (w', x)-leaf can split into k (w'', x)-leaves, and one of them will be used. In both cases (split or not), if $x > \ell$ the (v, x)-leaf on which (w, ℓ) is scheduled, splits to a closed (v, ℓ)-leaf that is allocated to the job and to an open $(v, x - \ell)$-leaf.

Scheduling Rule: The algorithm schedules the next (w, ℓ)-job on a leaf (v, x) that minimizes the lost bandwidth (given by $1/v - 1/w$). Ties are broken in favor of leaves (v, x) with minimal $x \geq \ell$.

5.3 Simulation Results

The implementation consists of two parts: the algorithm and the creation of an instance.

Algorithm: The implementation follows directly from the algorithm specification. The implementation employs search to find the fewest trees (machines) necessary to schedule the instance. The maximum number of trees necessary is the number of jobs n in the instance while the minimum number of trees is 1. We use binary search in this range to find the fewest machines necessary.

Instance Generation: In order to test the greedy algorithm, we generated random instances. Let H be the optimal number of trees for a given instance I. We generated instances with known H values to allow comparisons to the optimal.

Each instance is generated from a forest of H separate $(1, 1)$ roots, with each root generating an independent tree. Given a leaf node in a tree, the implementation randomly selects (with equal probability) to split it into k (prime number chosen randomly) children nodes, or to mark the node as frozen, prohibiting any future splits, or to split it into two children while conserving the window size in the children. The implementation uses a threshold value to terminate tree creation, and these leaves become jobs in instance I.

The optimal number of trees for instance I is exactly the number of trees, H, used to create the instance. We call these *non-perturbed* instances since the jobs in I are exactly those generated in the tree creation process. To create instances I with less consistent window sizes, we perturb the window sizes of nodes by increasing them slightly. We denote these as *perturbed* instances. In order to ensure that H does not decrease, we set a limit on the differences between the original width ℓ/w and the new width ℓ/w'. Specifically, the new value w' can be between w and $1.125w$ (to keep modifications small) as long as the total difference in width for all jobs remains under 1.0 ($\sum_I (\ell/w - \ell/w') < 1.0$).

Experimental Results: We ran the greedy algorithm and the three variations on 20 non-perturbed instances and 20 perturbed instances for H between 5 and 100 (stepping by 5). The three variations sort the jobs within an instance in different ways before using the scheduling routine of the greedy algorithm. The first variation, called Demand, sorts the jobs according to their widths ℓ/w. The Most demanding jobs (larger ℓ/w) are scheduled first. Ties are broken in favor of small w. The second variation, called Length, sorts jobs by length, with longer jobs scheduled first, and ties are broken in favor of small w. In the final variation, Online, the jobs are shuffled randomly and scheduled in the resulting order.

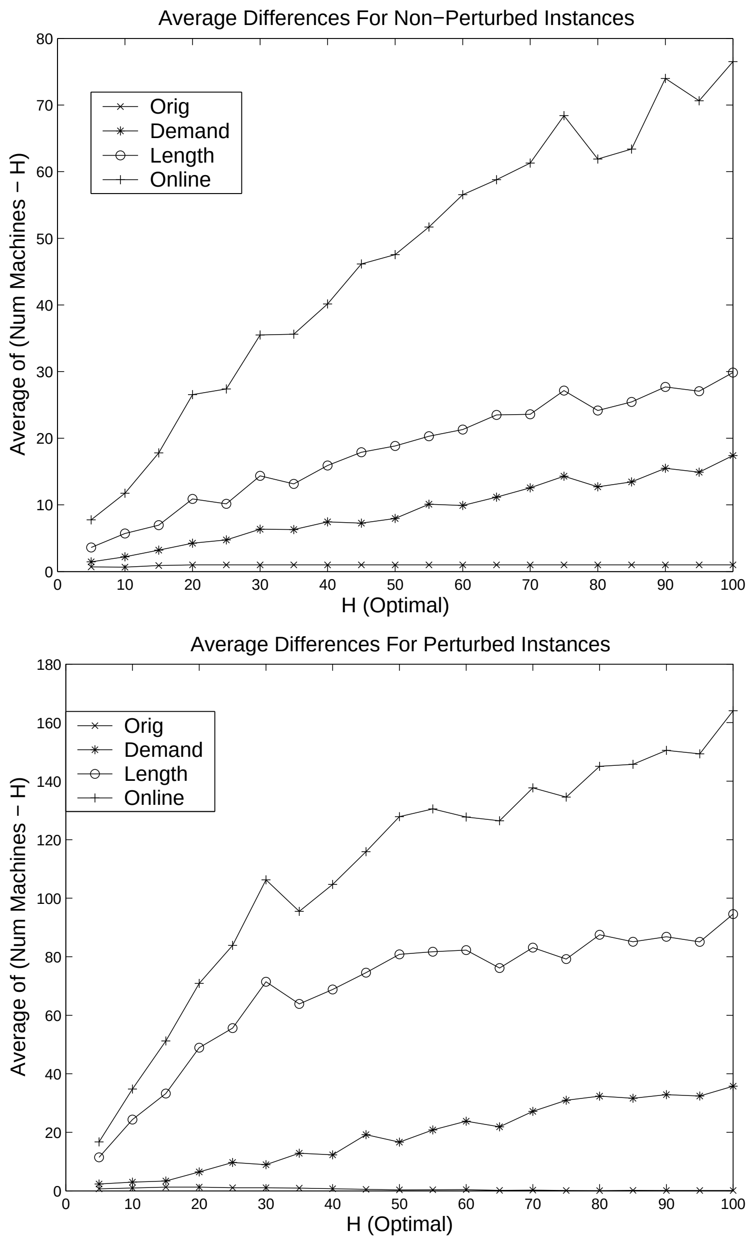

Figure 2: For non-perturbed (top) and perturbed (bottom) instances: Shows the average difference (20 runs per H) in number of machines used and the optimal number of machines (H).

The top of figure 2 shows the results for all four algorithms (Orig, Demand, Length, Online) for non-perturbed instances. The average differences between the number of machines scheduled and H are shown. In every experimental run, the original greedy algorithm used the fewest machines of all four variations and used H or $H + 1$ machines. The results are similar for perturbed instances as shown in Figure 2 (bottom), with the original greedy algorithm using the fewest machines. For the Demand, Length, and Online versions, the average difference for each H is roughly twice the non-perturbed results. The original algorithm used between H and $H + 3$ machines in every experimental run. For H greater than 30 the original greedy is usually optimal.

6. OPEN PROBLEMS

Can the approximation factor 8 be improved? Alternatively, is there a $C > 1$ for which a C-approximation is NP-hard? These questions also apply to the special cases of thrift schedules and windows scheduling of power-2 instances. The optimal algorithm presented in this paper for power-2 instances is for thrift schedules. Is it NP-hard to find a non-thrift optimal schedule for these instances? As shown in the thriftiness paradox, the optimal schedule is not necessarily thrift.

The greedy algorithm performs very well in practice. Is there any theoretical bound on its performance? Are there better natural algorithms for practical instances?

All of our solutions and previous solutions to the original windows scheduling do not use migrations. That is, a particular job is scheduled only on one machine. It is open to see if solutions with migrations perform better.

Thrift schedules and the generalized windows scheduling are special cases of a general problem in which jobs may be scheduled with some *jitter*. That is, job i is associated with jitter parameters j_i^{ub} and j_i^{lb} and the window between any two consecutive executions of job i must be no smaller than $w_i - j_i^{lb}$ and no larger than $w_i + j_i^{ub}$. In a thrift schedule both jitter parameters equal 0 and in the generalized windows scheduling problem $j_i^{ub} = 0$ and $j_i^{lb} = w_i - 1$. This generalization is motivated by maintenance problems in which jobs cannot get the service too often.

7. REFERENCES

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

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

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

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

[5] A. Bar-Noy, R. E. Ladner, and T. Tamir. Scheduling techniques for media-on-demand. In *Proceedings of the 14-th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 791–800, 2003.

[6] A. Bar-Noy, R. E. Ladner, and T. Tamir. Windows scheduling as a restricted version of bin packing. In *Proceedings of the 15-th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 217–226, 2004.

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

[8] A. Bar-Noy, A. Nisgav, and B. Patt-Shamir. Nearly optimal perfectly-periodic schedules. *Distributed Computing,* 15(4):207–220, 2002.

[9] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: a notion of fairness in resource allocation. *Algorithmica,* 15:600–625, 1996.

[10] S. K. Baruah S-S. Lin. Pfair Scheduling of Generalized Pinwheel Task Systems *IEEE Transactions on Computers,* 47(7):812–816, 1998.

[11] Z. Brakerski, A. Nisgav, and B. Patt-Shamir. General perfectly periodic scheduling. In *Proceedings of the 21-st ACM Symposium on Principles of Distributed Computing (PODC),* 163–172, 2002.

[12] Y. Chan and F. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica,* 9:425–462, 1993.

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

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

[15] W. S Evans, D. G. Kirkpatrick. Optimally scheduling video-on-demand to minimize delay when server and receiver bandwidth may differ. In *Proc. of the 15-th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1041–1049, 2004.

[16] 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.

[17] E. A. Feinberg and M. Curry. Online scheduling: generalized pinwheel problem. In *Proceedings of the 42-nd IEEE Conference on Decision and Control (CDC),* 4333–4338, 2003.

[18] M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

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

[20] L. Gao, J. Kurose, and D. Towsley. Efficient schemes for broadcasting popular videos. *Multimedia Systems*, 8(4): 284–294, 2002.

[21] 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 Conference on System Sciences,* 693–702, 1989.

[22] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel. Pinwheel scheduling with two distinct numbers. *Theoretical Computer Science,* 100:105–135, 1992.

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

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

[25] C. Kenyon and N. Schabanel. The data broadcast problem with non-uniform transmission times. *Algorithmica,* 35(2):146–175, 2003.

[26] C. Kenyon, N. Schabanel, and N. E. Young. Polynomial-time approximation scheme for data broadcast. In *Proc. of the 32-nd ACM Symposium on Theory of Computing (STOC)*, 659–666, 2000.

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

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

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

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