

Windows Scheduling of Arbitrary Length Jobs on Multiple 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, $I = \langle (w_1, \ell_1), (w_2, \ell_2), \dots, (w_n, \ell_n) \rangle$ of n pairs of positive integers that are associated with the jobs $1, 2, \dots, n$, respectively. The processing length of job i is ℓ_i slots where a slot is the processing time of one unit of length. The goal is to repeatedly and non-preemptively schedule all the jobs on the fewest possible machines such that the gap (window) between two consecutive beginnings of executions of job i is at most w_i slots. This problem arises in push broadcast systems in which data is transmitted on multiple 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 extended for arbitrary-length jobs mainly because the optimal number of machines might be arbitrarily larger than the generalized lower bound $W(I) = \sum_{i=1}^n (\ell_i/w_i)$. The main result of this paper is an 8-approximation algorithm for the WS problem with arbitrary lengths using new methods, different from those used for the unit-length case. The paper also presents another 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 cases and simulations show that it performs very well in practice.

*A preliminary version appeared in the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 56-65. 2005.

[†]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.

[§]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. vandegrift@up.edu.

1 Introduction

The *windows scheduling problem with arbitrary job length* for n jobs on multiple machines is defined as follows: Given is a sequence of n positive integer pairs $I = \langle (w_1, \ell_1), (w_2, \ell_2), \dots, (w_n, \ell_n) \rangle$ that are associated with the jobs $1, 2, \dots, n$, respectively. The processing length of job i is ℓ_i slots. For simplicity, we assume integer lengths and that each unit of length is processed in one slot of time. The goal is to repeatedly and non-preemptively schedule all the jobs on the fewest possible 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 $I = \langle (4, 2), (8, 4), (8, 2), (16, 4), (16, 4) \rangle$ and call the 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 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. The windows scheduling 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 fraction of the time.

Previous results for the windows scheduling problem considered either the case of one machine with unit-length jobs (the *pinwheel problem* [27, 28]), or the case of unit-length jobs with multiple machines ([5, 7]), or the case of one machine with arbitrary-length jobs (the *generalized pinwheel problem* [11, 19, 20]). This paper considers the generalized windows scheduling problem of multiple machines with arbitrary-length jobs.

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., [8, 22]), 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 modeled by the windows scheduling problem.

In *harmonic* windows scheduling (e.g., [6]), 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., [30, 29]) that are based on the concept of receiving data from multiple channels and buffering data for future playback. This concept was first developed in [38] 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 i th 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 [28] and the generalized pinwheel problem was considered in [11, 19, 20]. In these and other papers about the pinwheel problem, the focus was to characterize the instances that can be scheduled on one machine. For example, [13] 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]. This paper designed schedules using $opt + O(\ln(opt))$ machines, where opt is the number of machines used by an optimal solution.

In [34] 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 [36] an optimal solution for the chairman assignment problem was presented for a stronger fairness condition that depends on the prefixes of the schedule. These papers considered unit-length jobs on a single machine. Other work on periodic scheduling are [31, 14, 24]. In the generalized model, each job has an arbitrary length and it can be scheduled on multiple machines. For this case, [10] 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]. The case of unit-length jobs was addressed in [4] where constant approximation solutions were presented. These results were improved in [33] that presented a polynomial time approximation scheme. The arbitrary length case was considered in [32] where a constant approximation solution was presented.

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 they request. The unit-length case was considered in [9] and the general case of jobs with arbitrary lengths was considered in [12].

Closely related problems from the operation research and the communication networks areas include the machine maintenance problem [39, 3], the multi-item replenishment and other inventory problems [25, 35, 26], and the sensor resource management problem [19].

Windows scheduling with unit-length jobs is a special case of the *bin packing* problem (e.g., [16]) 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 while 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 [7].

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 [17, 18, 23]. These bounds can be achieved in the limit in the windows scheduling model [6]. Recently, our greedy algorithm for the windows scheduling problem with arbitrary job lengths (presented in Section 5) was used in an implementation of an algorithm for periodic broadcast of *variable bit rate movies* [15]. In this paper, the induced periodic broadcast problem is mapped into a windows scheduling problem, that is solved with the greedy algorithm. The result is a lossless and practical method, achieving low client delay and low bandwidth requirement.

1.3 Contributions

We develop new approximation algorithms that are based on novel methods and techniques. We consider two special cases separately and combined: (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 general 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, where $W(I)$ is the total width of the jobs (formally, $W(I) = \sum_i(\ell_i/w_i)$), and w_{max} is the maximum window size of some job. Next, we present 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.

Our solutions do not use migrations. That is, a particular job is scheduled only on one machine. We note that if migrations are allowed (that is, different execution of a job might be on on different machines), then for some instances the optimal solution might required fewer machines.

1.4 Paper Organization

In Section 2 we motivate the study of approximation algorithms for the problem by proving that it is strongly NP-hard even if all the windows are powers of 2. We then study the thriftiness part and analyze the tradeoff between allocating the fewest possible number of time-slots to jobs, and using the minimal number of machines. Finally, we present algorithms for two special cases: instances with uniform lengths or uniform windows. These algorithms will serve as components in some of the approximation algorithms to be described later in the paper. In Section 3 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. In Section 4 we present two approximation algorithms: first, an algorithm that uses $2(1 + \varepsilon)W(I) + \log w_{max}$ machines, and then an 8-approximation algorithm. Finally, in Section 5, we introduce a tree representation of perfect schedules and a greedy algorithm that is based on this representation. The output of the greedy algorithm 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.

2 Preliminaries

2.1 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 . The total width of the jobs in an instance $I = \langle (w_1, \ell_1), (w_2, \ell_2), \dots, (w_n, \ell_n) \rangle$ is

$$W(I) = \sum_{i=1}^n (\ell_i/w_i) .$$

We consider only *non-preemptive* schedules in which the ℓ_i slots allocated to job i must be

successive.¹ We assume that $\ell_i \leq w_i$ for all $1 \leq i \leq n$. Otherwise, there is no feasible schedule of job i . Two special types of schedules are

Perfect schedules – 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.

Thrift schedules – 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 machine ([4, 19, 7]), we look for approximate solutions. A natural lower bound to the 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 $\lceil W(I) \rceil$ 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 [5]. The following example demonstrates that this lower bound can be arbitrarily far from the optimal solution for the windows scheduling problem with arbitrary job lengths.

Example: Let $I = \langle (r, 1), (r^2, r), \dots, (r^n, r^{n-1}) \rangle$ 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.

2.2 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, with arbitrary job lengths problem is strongly NP-hard even if all the w_i 's are powers of 2.

Theorem 2.1 *The windows scheduling problem with arbitrary job lengths 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 [21]. An instance of 3-partition is defined as follows.

Input: A set A of $3m$ elements, a number $B \in \mathbb{Z}^+$, and a size $s(x)$ for each $x \in A$ such that $B/4 < s(x) < B/2$ and $\sum_{x \in A} s(x) = mB$.

Output: Is there a partition of A into m disjoint sets, S_1, S_2, \dots, S_m , such that, $\sum_{x \in S_i} s(x) =$

¹Informally, the *preemptive* version is identical to windows scheduling of unit-length jobs by replacing each (w, ℓ) -job by ℓ $(w, 1)$ -jobs.

B for $1 \leq i \leq m$? Note that the above constraints on the element sizes imply that such a partition exists if and only if every S_i is composed of exactly three elements from A .

Given an instance of 3-partition, 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 a job with parameters $(kW, s(x))$ in I . In addition, I includes one job z with parameters $(W, W - B)$, and $k - m$ dummy jobs, d_1, d_2, \dots, d_{k-m} with parameters (kW, B) . Obviously the above construction can be done in polynomial time in n and the size of I is also polynomial in n .

Assume now that A has a partition. It follows that I has the following schedule:

$$[z, \mathcal{S}_1, z, \mathcal{S}_2, z, \dots, \mathcal{S}_m, z, d_1, z, d_2, \dots, z, d_{k-m}]$$

where \mathcal{S}_i is a schedule in arbitrary order of the jobs associated with the elements from S_i . 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

$$\sum_{x \in A} s(x) + k\ell_z + (k - m)B = mB + k(W - B) + (k - m)B = kW .$$

Now assume that there exists a schedule of I on one machine. Note that

$$W(I) = \sum_{x \in A} \frac{s(x)}{kW} + \frac{W - B}{W} + \frac{(k - m)B}{kW} = \frac{mB}{kW} + 1 - \frac{B}{W} + \frac{kB}{kW} - \frac{mB}{kW} = 1 .$$

Thus, any schedule of I on one machine must be thrifty. In particular, if I has a schedule on one machine then the schedule of job z must be with an exact W -window. This means that m out of the k gaps of B slots between the k executions of job z induce a partition of A where the other $k - m$ gaps are allocated to the dummy jobs. Note that the schedule is cyclic and therefore there are k gaps between the k executions of job z . ■

2.3 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 although thrifty schedules allocate the fewest possible number of slots to jobs, 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 jobs such that $w_i = p_i$ and $\ell_i = 1$ where p_1, \dots, p_r are r distinct primes greater than r . A thrifty 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 not-necessarily thrifty 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 thrifty schedule ([5]) uses $OPT(I)$ machines, and therefore the thriftiness-price ratio is 1 for this case. 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 with $OPT(I)$ machines for power-2 instances remains open. Moreover, we do not know if the problem is NP-hard. The following example demonstrates this “paradox”.

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 of size 3 and each of the other five jobs is granted a window of size 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 there are four such gaps but five $(16, 2)$ -jobs to schedule.

The next two theorems show that the above example can be extended to 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. The proof of Theorem 2.2 is given at the end of Section 3 - as it uses the optimal algorithm that is presented in that section.

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

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 $i = 0, 4, 8, 4k, \dots$, 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,

$$I_0 = \langle (4, 1), (16, 2), (16, 2), (16, 2), (16, 2), (16, 2) \rangle$$

which is the instance from the paradox example above, and

$$I_4 = \langle (64, 16), (256, 32), (256, 32), (256, 32), (256, 32), (256, 32) \rangle.$$

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 $i \neq j$ cannot be scheduled on the same machine. This is true since, assuming w.l.o.g that $i > j$, 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 $5(2^i + 2^{i+1}) < 2^{i+4}$. ■

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, there are four such gaps but five $(2^{i+4}, 2^{i+1})$ -jobs to schedule. Thus, an additional machine must be used. ■

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. ■

2.4 Uniform Lengths or Uniform Windows

If all the jobs have the same length then we show that 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 any $(k\ell, \ell)$ -job by a $(k, 1)$ -job.

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 $(k, 1)$ -job. Then any schedule of I induces a schedule of I' and vice-versa. In particular, $OPT(I) = OPT(I')$.*

Proof:

Given I , consider first the instance I'' obtained from I by replacing each $(k\ell + r, \ell)$ -job ($0 \leq r < \ell$) by a $(k\ell, \ell)$ -job. In words, each window is rounded down to the nearest multiple of ℓ . Since all the windows in I are not smaller than the windows in I'' , any schedule of I'' is feasible for I , in particular this implies that $OPT(I) \leq OPT(I'')$. For the other direction, we show that any schedule of I can be converted to a schedule of I'' without increasing the number of machines. Consider a schedule of I . Partition this schedule into “slices” of ℓ slots. If for all jobs, all the executions of a job are in a single complete slice, it means that the actual windows are multiples of ℓ and therefore this schedule is feasible also for I'' . Else consider the first time in the schedule in which some job, j , is not “aligned” in a single slice. That is, j is processed during the last x slots of some slice and during the first $\ell - x$ slots of the next slice. Since all jobs have length ℓ , and this is the first time that some job is not aligned in a slice, it must be that the machine is idle in the $\ell - x$ slots before the process of j begins. It is possible to *shift* the whole schedule on this machine $\ell - x$ slots to the left. This action does not hurt the feasibility of the schedule of j or any other job following it since gaps only decrease. Repeating this process, it is possible to keep delaying the first time in which some job is not aligned in a single slice until all jobs are scheduled in a window that is a multiple of ℓ . Since windows only decrease in this process, the resulting schedule is a feasible schedule of I'' .

We next show a correspondence between a schedule of I'' and I' . Any schedule of I' , with unit-length jobs, can be “stretched” by a factor of ℓ to produce a schedule for I'' . Also, each schedule of I'' can be “shrunk” to induce a schedule for I' (given the above method, the schedule is aligned to slices of ℓ slots each). ■

The above theorem implies that any approximation algorithm for unit-length suits also instances in which all the jobs have the same length. Another special case is when all jobs have the same window, w . In this case the problem reduces to *Bin-packing with discrete sizes*. Formally, items of sizes in $\{1/w, 2/w, \dots, w/w\}$ 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 result for bin-packing [37], we get the following.

Corollary 2.7 *There exists an APTAS for windows scheduling with uniform windows.*

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 on a minimal number of 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 of a power-2 instance, 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. ■

The above property motivates the main idea in our algorithm: the jobs having window w are partitioned into ‘paired-groups’, such that the jobs of each group are processed $w/2$ slots apart from their paired group. Our algorithm finds these paired groups, and converts each pair into a job having window $w/2$. If such a pairing does not exist, idle slots are inevitable.

Algorithm \mathcal{A}_T : Let $w_{max} = 2^k w_{min}$. The algorithm proceeds in three phases. An overview of these phases is given in Figure 1.

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 J 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 J such that $\ell_1 \geq \ell_2 \geq \dots$. Let j be such that $\ell_1 = \ell_2 + \ell_3 + \dots + \ell_j$. If no such j exists, it must be that $\ell_1 > \ell_2 + \ell_2 + \dots + \ell_{|J|}$ (because each ℓ_i is a power of 2) and all the jobs of J are replaced by a single $(w/2, \ell_1)$ -job. If such a j exists, \mathcal{A}_T replaces the j jobs with a single $(w/2, \ell_1)$ -job, and continues in the same way with the rest of J . 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 , $i = k, k - 1, \dots, 1$, \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 I: 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

Let $k = \log_2 w_{max}/w_{min}$.

$I_0 = I$

1. For $i = 0$ to $k - 1$ do:

Let $w = w_{max}/2^i$.

1.1 schedule each (w, w) -job from I_i on a dedicated machine.

1.2 build from the remaining jobs of I_i an instances I_{i+1} with maximal window $w/2$.

2. Schedule I_k greedily on $h' = \lceil \sum_{j \in I_k} \ell_j / w_{min} \rceil$ machines.

3. For $i = k$ down to 1 do:

3.1 Double the schedule of I_i

3.2 Replace each $(w_{max}/2^i)$ -job originated from I_{i-1} by the $(w_{max}/2^{i-1})$ -jobs composing it.

Figure 1: The Algorithm \mathcal{A}_T

$I_2 = \langle a = (4, 1), b' = (4, 2), c' = (4, 1) \rangle$ 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 constructs 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 II: Consider the instance $I = \langle a = (4, 2), b = (8, 2), c = (8, 2), d = (16, 8), e = (16, 4), f = (16, 2), g = (16, 1) \rangle$. It has $w_{max} = 16$. First, \mathcal{A}_T replaces d, e, f and g by $d' = (8, 8)$. That is, $I_1 = \langle a = (4, 2), b = (8, 2), c = (8, 2), d' = (8, 8) \rangle$. 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 $[d, d, d, d, d, d, d, d, e, e, e, e, f, f, g, *]$. These two machines together form a schedule of $I_0 = I$.

3.1 Analysis of \mathcal{A}_T

We show that \mathcal{A}_T is optimal for power-2 instances. Recall that for $i = 0, \dots, k - 1$, h_i is the number of machines allocated in iteration i to $(w_{max}/2^i, w_{max}/2^i)$ -jobs. Also, h' is the number of machines used to schedule I_k . The total number of machines used by \mathcal{A}_T to schedule I is therefore

$$n_T(I) = \sum_{i=0}^{k-1} h_i + h'.$$

We bound the value of h_i using the following lemma.

Lemma 3.2 For all i , $0 \leq i \leq k - 1$, $h_i \leq OPT_T(I_i) - OPT_T(I_{i+1})$.

Proof: We show 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 $OPT_T(I_{i+1}) \leq OPT_T(I_i) - h_i$.

Let S_i be a schedule of I_i on h machines. Let $w = w_{max}/2^i$. In the move from I_i to I_{i+1} , w -jobs are eliminated. Some of them, the (w, w) -jobs, are scheduled on dedicated machines. According to the algorithm, there are h_i such jobs. Since each such (w, w) -job has width = 1, it must have a dedicated machine in S_i . Therefore, all other jobs are scheduled in S_i on $h - h_i$ machines. The following is an algorithm that produces a schedule, S_{i+1} , of I_{i+1} on $h - h_i$ machines.

First, all the jobs of I_i with window smaller than w appear also in I_{i+1} and their schedule in S_i induces their schedule on S_{i+1} . When constructing I_{i+1} , w -jobs of I_i with width < 1 , are replaced by $(w/2)$ -jobs. Consider the longest $(w/2)$ -job in I_{i+1} that is originated from w -jobs of I_i , specifically, from a (w, ℓ_1) -job, j_1 , and a set B_1 of w -jobs such that $\sum_{j \in B_1} \ell_j \leq \ell_1$. Consider the schedule of j_1 in I_i . By Claim 3.1, the $(w/2)$ -apart slots are idle or allocated to w -jobs. Thus, these slots can be used to schedule in S_{i+1} the jobs of B_1 . If other w -jobs were scheduled in these slots in S_i , the other jobs can be scheduled in the slots allocated to the jobs of B_1 in I_i . It is always possible to move these jobs into the slots of the jobs of B_1 since all the lengths are powers of 2 and the jobs of B_1 are the longest among the w -jobs of I_i (according to the way \mathcal{A}_T groups them with j_1). Continue in the same way with the next largest grouped job in I_{i+1} and schedule it in S_{i+1} according to the schedule of the longest job in the group, until all the grouped jobs of I_{i+1} are scheduled in the slots that were allocated to the w -jobs composing them. ■

Combining the above Claim with the observation that I_k is packed optimally by \mathcal{A}_T (in other words $OPT_T(I_k) = h'$), we get that

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

Therefore,

Theorem 3.3 *For any power-2 instance \mathcal{A}_T schedules I on $n_T(I) = OPT_T(I)$ machines.*

Proof of Theorem 2.2: Let I be a power-2 instance, we show that $OPT_T(I) \leq 2 \cdot OPT(I)$. Given any schedule for I on h machines, construct a thrift schedule for I on $2h$ machines. In particular, for the optimal schedule for I , we get the statement of the theorem.

The construction is per-machine, that is, given a machine, M , on which the set of jobs $I_M \subseteq I$ is scheduled, we show that the algorithm \mathcal{A}_T schedules 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 $W(I_M) \leq 1$, and also, for any job (w, ℓ) in I_M , $\ell < w_{min}(I_M)$. 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 (w, ℓ) must be allocated ℓ 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 the algorithm never dedicate machines to (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 $1/2^{k+1} + \dots + 1/8 + 1/4 < 1$. 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 at most $\lceil W(I_M) + W(\text{dummy jobs}) \rceil \leq \lceil 1 + 1 \rceil \leq 2$. ■

4 Approximation Algorithms for Arbitrary Instances

4.1 An Algorithm that Uses $2(1 + \varepsilon)W(I) + \log w_{max}$ Machines

Consider the following algorithm that is based on partitioning the instance into subsets of jobs and scheduling each subset independently. First, as a preprocessing, one machine is dedicated to each job whose width is more than $1/2$. Next, for the remainder of the instance, reduce each window w_i to the nearest power of 2. Let S_u be the subset of the instance whose windows were rounded to 2^u . Clearly, a schedule of the rounded instance is also a feasible schedule of the original instance because jobs are processed at least as frequently as required. The rounding process creates at most $\log w_{max}$ different instances, $S_1, S_2, \dots, S_{\log w_{max}}$ each having jobs with *uniform windows*.

As explained in Section 2.4, each sub-instance S_u can be scheduled separately on a disjoint set of machines by any algorithm for the bin-packing problem. In particular, by Corollary 2.7 one can use the asymptotic PTAS of Vega and Leuker [37] that uses at most $(1 + \varepsilon)OPT(I) + 1$ bins to pack an instance I .

Theorem 4.1 *The above algorithm uses at most $2(1 + \varepsilon)W(I) + \log w_{max}$ machines.*

Proof: Consider first the jobs with width larger than $1/2$ - each machine dedicated to such a job is busy at least half of the time, thus, a 2-ratio between the number of machines and the total width processed is preserved for this sub-instance. Next, for any subset of jobs S_u , let S'_u be the set of rounded jobs in S_u . Since the rounding might increase the width of each job by a factor of less than 2, it holds that $W(S'_u) < 2W(S_u)$. The approximation scheme for bin packing then schedules (packs) the jobs of S'_u on at most $(1 + \varepsilon)OPT(S'_u) + 1$ machines (as explained in Section 2.4). Clearly $OPT(S'_u) \geq W(S'_u)$, thus at most $2(1 + \varepsilon)OPT(S_u) + 1$ machines are used to schedule S_u . All together, for the $\log w_{max}$ different subsets and for the wide jobs, at most $2(1 + \varepsilon)W(I) + \log w_{max}$ machines are used. ■

4.2 An 8-Approximation Algorithm

In this Section we present an 8-approximation algorithm for arbitrary instances, the factor of 8 is a result of three components, each contributing a factor of at most 2.

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 $OPT_T(J'')$, and a (w, w) -job must be allocated a dedicated machine in a schedules of J' as well as on any schedule of J'' , we can assume w.l.o.g. that such jobs do not exist.

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

Proof: Given a thrift schedule of J' on h machines, construct a thrift schedule of J'' on $2h$ machines. In particular, for the optimal schedule of J' we get the statement of the theorem.

The construction is per-machine, that is, given *one* machine that processes thriftly a set of jobs $S' = \{w_i, \ell_i\} \subseteq J'$, construct a two-machine schedule of the corresponding set $S = \{w_i, 2\ell_i\} \subseteq J''$ of jobs. If S' consists of a single (w, w) -job, then the corresponding (w, w) -job in S can be scheduled on a single machine.

Consider an execution of the optimal thrift algorithm, \mathcal{A}_T , on S' . Denote this execution $\mathcal{A}_T(S')$. Since \mathcal{A}_T is optimal and it is given that S' is scheduled on a single machine, \mathcal{A}_T ends up with a one-machine schedule for S' . Consider an execution of \mathcal{A}_T on S . Denote this

execution $\mathcal{A}_T(S)$. Recall that \mathcal{A}_T is based on grouping jobs having the same window to a single job with a half-window. The grouping is done in non-increasing order of job's length, and since this order is identical in S and S' , as long as there are no dedicated machines (that are dedicated to (w, w) -jobs), the grouping in $\mathcal{A}_T(S)$ and $\mathcal{A}_T(S')$ is identical. That is, if in $\mathcal{A}_T(S')$ a (w, ℓ_1) -job is grouped with a set B' of w -jobs to get a $(w/2, \ell_1)$ -job, then in $\mathcal{A}_T(S)$, a $(w, 2\ell_1)$ -job is grouped with the set B of w -jobs, where B is the set of double-length jobs corresponding to B' .

Claim 4.3 *No dedicated machines are allocated in $\mathcal{A}_T(S')$.*

Proof: Assume that for some w , a (w, w) -job is created in $\mathcal{A}_T(S')$. Since the final schedule created by $\mathcal{A}_T(S')$ is on a single machine, all the jobs of S' are packed in this (w, w) -job. However, only $2w$ -jobs are grouped when creating the (w, w) -job, which implies that before this iteration *all* jobs have the same $2w$ -window and $\mathcal{A}_T(S')$ should have packed them greedily without grouping. A contradiction. ■

Claim 4.4 *No dedicated machines are allocated in $\mathcal{A}_T(S)$.*

Proof: Assume that for some w , a (w, w) -job is created in $\mathcal{A}_T(S)$. It was created by grouping a $(2w, w)$ -job, J_1 , with a set of $2w$ -jobs. Since the same grouping is done in $\mathcal{A}_T(S)$ and $\mathcal{A}_T(S')$, a $(w, w/2)$ -job is created in $\mathcal{A}_T(S')$. In the next iteration of $\mathcal{A}_T(S')$, this $(w, w/2)$ -job will be grouped with other w -jobs to create a $(w/2, w/2)$ -job. By Claim 4.3, there are no dedicated machines in the $\mathcal{A}_T(S')$, therefore a $(w/2, w/2)$ -jobs can not be created. It must be that $w_{min}(S') = w$. Since the window values in S and S' are the same (only the lengths are different), it must be that $w_{min}(S) = w$ as well. Thus, a (w, w) -job can be created in $\mathcal{A}_T(S)$ only before the last iteration - in which all jobs have the same w_{min} -window, and are scheduled greedily. ■

Let S_{last}, S'_{last} denote the instance of w_{min} -jobs to be packed in the last iteration of $\mathcal{A}_T(S), \mathcal{A}_T(S')$, respectively. Consider $\mathcal{A}_T(S')$. Since no dedicated machines are used, all the jobs are represented in S'_{last} and are scheduled optimally on $\lceil W(S'_{last}) \rceil$ machines. Given that only one machine is used, it must be that $W(S'_{last}) \leq 1$. Since as long as no dedicated machines are allocated, the grouping in S and S' is exactly the same, it must be that $W(S_{last}) = 2W(S'_{last}) \leq 2$. Therefore, in $\mathcal{A}_T(S)$, at most $\lceil W(S_{last}) \rceil \leq 2$ machines are used to schedule all the jobs of S . ■

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.

Lemma 4.5 $OPT_T(J'') \leq 2 \cdot OPT_T(J')$.

Proof: Let $J'' = \{w_i, \ell_i\}$ and $J' = \{2w_i, \ell_i\}$. Consider the instance $K = \{2w_i, 2\ell_i\}$. By Lemma 4.2, $OPT_T(K) \leq 2 \cdot OPT_T(J')$. 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 $1, 2, \dots$. The following property will be used in our construction:

Claim 4.6 *There exists an optimal schedule of K in which all schedules of all jobs begin in an odd-indexed slots.*

Proof: Given an optimal schedule of K , scan it from left to right. Denote by *odd-aligned interval* consecutive slots in which all schedules of all jobs begin in an odd-index slot, and by *even-aligned interval* consecutive slots in which all schedules of all jobs begin in an even-index slot. Note that since all the sizes of windows and lengths in K are even, and since the schedule is thrift, for each job j , all the schedules of j are either in odd- or even-aligned intervals. Also, there must be at least one idle slot before every even-aligned interval. It is therefore possible to shift by one slot to the left every even-aligned interval. The schedule is still thrift and valid since for every non-aligned job, *all* schedules of this jobs were moved. In the resulting schedule there are only odd-aligned intervals, in other words, all schedules of all jobs begin in an odd-indexed slots. ■

Given a schedule of K in which all schedules of all jobs begin in an odd-index slot, for every $t \geq 1$, the slots $2t - 1, 2t$ process the same (even-length) job, or are both idle. It is therefore possible to compress this schedule, by taking just the odd slot out of each such pair. The resulting instance is a schedule of J'' . ■

Analysis of Algorithm \mathcal{A} : We can now summarize the analysis of algorithms \mathcal{A} :

Theorem 4.7 *For any instance I , \mathcal{A} schedules I on at most $8 \cdot OPT(I)$ machines.*

Proof: Recall that for a given instance I , the algorithms constructs two instances: J' – obtained from I by rounding the lengths down to powers of 2 and rounding the windows up to powers of 2, and J – obtained from J' by replacing each (w, ℓ) -job by a $(w/2, 2\ell)$ -job. It then runs \mathcal{A}_T on J . Combine Lemmas 4.2 and 4.5, to get $OPT_T(J) \leq 4OPT_T(J')$. An additional factor of 2 is due to the thrift price of 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, $OPT_T(J) \leq 8OPT(I)$. ■

5 A Practical Algorithm

We present a greedy algorithm for the windows scheduling problem with arbitrary job lengths, 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).

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 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 2 represents a schedule of the instance $I = \{a = (4, 1), b = (8, 2), c = (8, 1), d = (8, 1), e = (16, 2), f = (16, 2)\}$.

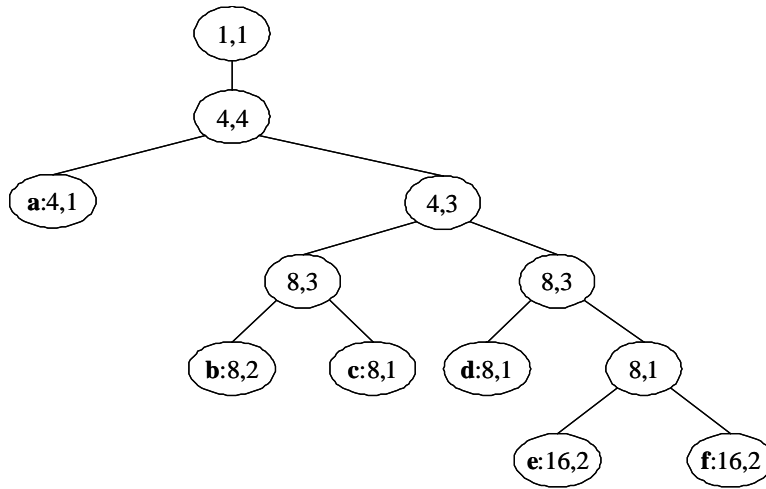


Figure 2: Tree representation of a 1-machine schedule

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 $(w, \ell_1), (w, \ell_2), \dots, (w, \ell_k)$, 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 tree in Figure 2 can be mapped into a schedule of the instance I as follows: the $(1, 1)$ -root is replaced by a $(4, 4)$ -node that splits into a $(4, 1)$ -leaf allocated to job a and a $(4, 3)$ -node that is the root of its right child. The corresponding (intermediate) schedule is $[a, *, *, *]$. In the next level, the $(4, 3)$ -node splits into two $(8, 3)$ -nodes. The corresponding schedule is $[a, *, *, *, a, *, *, *]$. Note that $'*$ ' denotes an idle slot, however, this split means that the two groups of idle slots will be allocated to different jobs. In the next level, one $(8, 3)$ -node splits and allocated to the jobs b and c , and the other splits into a $(8, 1)$ -leaf allocated to the job d , and into an idle $(8, 2)$ -node. That is, the corresponding schedule is now $[a, b, b, c, a, d, *, *]$. Finally, the idle $(8, 2)$ -node splits into two $(16, 2)$ -nodes, allocated to jobs e and f , to get the complete schedule $[a, b, b, c, a, d, e, e, a, b, b, c, a, d, f, f]$.

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 width (given by $\ell/v - \ell/w$). Ties are broken in favor of leaves (v, x) with minimal $x \geq \ell$.

Note that any job can be scheduled on a new tree with no lost width. Specifically, when a new tree is added, its root is replaced by a (w, w) -node, which splits into a (w, ℓ) leaf - allocated to the job, and an open $(w, w - \ell)$ -leaf. A new tree must be added whenever the scheduled job can not fit to any open leaf. However, to avoid lost width, it might be efficient to open a new tree also when it is not compulsory. Therefore, the algorithm should search for the optimal number of trees (see detailed in Section 5.3).

5.3 Simulation Results

The implementation consists of two parts: the algorithm and the creation of an instance. We discuss these parts separately.

Algorithm: The implementation follows directly from the algorithm specification. The implementation employs binary 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 ².

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) one of three cases. In the first case, the node will split into p (prime number chosen randomly among the first k primes) children nodes. For example, if the original node is $(4, 2)$, and the random splitting factor p is 3, then the original node splits into 3 $(12, 2)$ nodes. In the second case, the node is marked as frozen, prohibiting future splits. In the third case, the node is split into two children while conserving the window size of the children. For example, if the original node is $(12, 8)$ then a new random length between 1 and 8 is selected to determine the window size of one of the two new nodes. If the random number selected is 3, then the original node splits into the nodes $(12, 3)$ and $(12, 5)$. 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. Non-perturbed instances have subsets of jobs with the same window size due to splits from common parent nodes.

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

²A sample of the instances that have been randomly generated for this experiment, and the schedule created for each of these instances are available at http://faculty.up.edu/vandegri/WS/windows_scheduling_experiments.html

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 ($\sum_{(w,\ell)\in I} (\ell/w - \ell/w') < 1.0$).

Experimental Results: We ran the greedy algorithm and three variations of it on 20 non-perturbed instances and 20 perturbed instances for H between 5 and 100 (stepping by 5), and $k = 3$ (splitting nodes splits into one of the first 3 primes). 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.

The top of figure 3 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 at the bottom of Figure 3, 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 Summary and Open Problems

In this paper we considered the windows scheduling problem with variable length jobs. This problem has numerous applications in production planning and media on-demand systems. We presented an 8-approximation algorithm for the problem, and additional algorithms with better approximation ratios for some special cases. We also considered the cost of being thrifty - that is, allocating to each job its minimal processing demand, and showed that surprisingly, this cost is unbounded. Finally, we presented a greedy algorithm that performs very close to the optimal according to our simulations. We conclude with the following list of open problems for further research.

- 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 thrifty schedules and windows scheduling of power-2 instances. The optimal algorithm presented in this paper for power-2 instances is only for thrifty schedules. Is it NP-hard to find a non-thrifty optimal schedule for these instances? As shown in the thriftiness paradox, the optimal schedule is not necessarily thrifty.

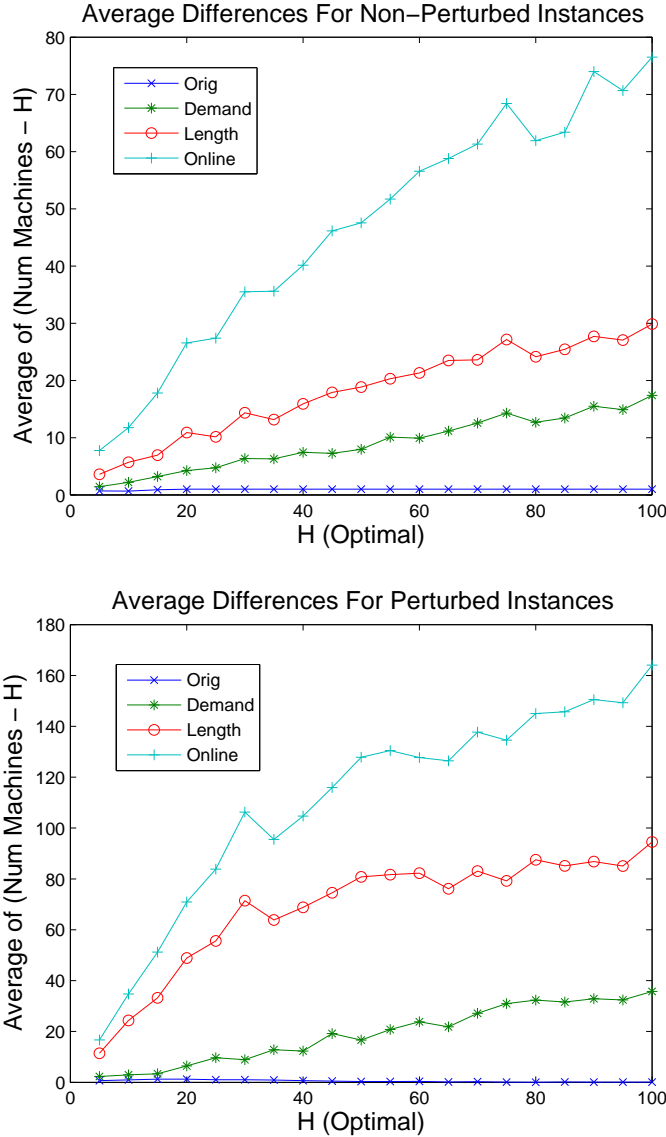


Figure 3: For non-perturbed (top) and perturbed (bottom) instances: Shows the average difference (20 runs per H) in number of machines used and the optimal number of machines (H).

- The 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 what is the power of migrations.
- Thrift schedules and windows scheduling with arbitrary job lengths 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 windows scheduling problem $j_i^{ub} = 0$ and $j_i^{lb} = w_i - 1$. This generalization is motivated by maintenance problems in which jobs should be served frequently but cannot get the service too often. Can our algorithms be extended for this problem? What is the resulting performance?

References

- [1] S. Acharya, M. J. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *IEEE Personal Communication*, 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] S. Anily, C. A. Glass, and R. Hassin. The scheduling of maintenance service. *Discrete Applied Mathematics (DAM)*, 82(1-3):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 (MOR)*, 27(3):518–544, 2002.
- [5] A. Bar-Noy and R. E. Ladner. Windows scheduling problems for broadcast systems. *SIAM Journal on Computing (SICOMP)*, 32(4):1091–1113, 2003.
- [6] A. Bar-Noy, R. E. Ladner, and T. Tamir. Scheduling techniques for media-on-demand. In *Proceedings of the 14-th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 791–800, 2003.
- [7] A. Bar-Noy, R. E. Ladner, and T. Tamir. Windows scheduling as a restricted version of bin packing. *ACM Transactions on Algorithms (TALG)*, 3(3), 2007.
- [8] A. Bar-Noy, J. Naor, and B. Schieber. Pushing dependent data in clients-providers-servers systems. *Wireless Networks journal (WINET)*, 9(5):175–186, 2003.
- [9] A. Bar-Noy, A. Nisgav, and B. Patt-Shamir. Nearly optimal perfectly-periodic schedules. *Distributed Computing*, 15(4):207–220, 2002.
- [10] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: a notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.
- [11] S. K. Baruah and S-S. Lin. Pfair scheduling of generalized pinwheel task systems. *IEEE Transactions on Computers*, 47(7):812–816, 1998.
- [12] Z. Brakerski, A. Nisgav, and B. Patt-Shamir. Dispatching in perfectly-periodic schedules. *Journal of Algorithms (JALG)*, 49(2):219–239, 2003.
- [13] M. Y. Chan and F. Y. L. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9(5):425–462, 1993.
- [14] A. Campbell, and J. Hardin, Vehicle minimization for periodic deliveries, *European Journal of Operational Research* 165, 668–684. 2005.
- [15] N. Cherniavsky and R.E. Ladner. Practical low delay broadcast of compressed variable bit rate movies. *Data Compression Conference*, 362–371, 2006

- [16] 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.
- [17] L. Engebretsen and M. Sudan. Harmonic broadcasting is bandwidth-optimal assuming constant bit rate. *Networks*, 47(3):172–177, 2006.
- [18] W. S. Evans and D. G. Kirkpatrick. Optimally scheduling video-on-demand to minimize delay when server and receiver bandwidth may differ. In *Proceedings of the 15-th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1041–1049, 2004.
- [19] E. A. Feinberg, M. Bender, M. T. 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.
- [20] E. A. Feinberg and M. T. Curry. Generalized Pinwheel Problem. *Mathematical Methods of Operations Research*, 62:99–122, 2005.
- [21] 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.
- [22] 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.
- [23] L. Gao, J. Kurose, and D. Towsley. Efficient schemes for broadcasting popular videos. *Multimedia Systems*, 8(4): 284–294, 2002.
- [24] A. Grigoriev, J. Van De Klundert, and F. Spieksma, Modeling and solving the periodic maintenance problem, *European Journal of Operational Research* 172, 783–797. 2006.
- [25] G. Hadley and T. M. Whitin. *Analysis of inventory systems*. Prentice-Hall, 1963.
- [26] R. Hassin and N. Megiddo. Exact computation of optimal inventory policies over an unbounded horizon. *Mathematics of Operations Research (MOR)*, 16(3):534–546, 1991.
- [27] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A real-time scheduling problem. In *Proceedings of the 22-nd Hawaii International Conference on System Sciences*, 693–702, 1989.
- [28] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel. Pinwheel scheduling with two distinct numbers. *Theoretical Computer Science (TCS)*, 100(1):105–135, 1992.
- [29] 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.
- [30] L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–271, 1997.

- [31] J. Korst, E. Aarts, and J. Lenstra. Scheduling periodic tasks with slack. *INFORMS Journal on Computing* 9, 351-362, 1997.
- [32] C. Kenyon and N. Schabanel. The data broadcast problem with non-uniform transmission times. *Algorithmica*, 35(2):146–175, 2003.
- [33] C. Kenyon, N. Schabanel, and N. E. Young. Polynomial-time approximation scheme for data broadcast. *CoRR cs.DS/0205012*, 2002. Also, in *Proceedings of the 32-nd ACM Symposium on Theory of Computing (STOC)*, 659–666, 2000.
- [34] C. L. Liu and J. W. Laylend. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.
- [35] R. Roundy. 98%-effective integer-ratio lot-sizing for one-warehouse multi-retailer systems. *Management Science*, 31:1416–1460, 1985.
- [36] R. Tijdeman. The chairman assignment problem. *Discrete Mathematics (DM)*, 32:323–330, 1980.
- [37] W. F. Vega and G .S. Leuker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.
- [38] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *ACM Multimedia Systems Journal*, 4(3):197–208, 1996.
- [39] W. Wei and C. L. Liu. On a periodic maintenance problem. *Operations Research Letters (ORL)*, 2:90–93, 1983.