Fairness-Free Periodic Scheduling with Vacations*

Jiří Sgall† Hadas Shachnai‡ Tami Tamir§

Abstract

We consider a problem of *repeatedly* scheduling n jobs on m parallel machines. Each job is associated with a profit that is gained each time the job is completed, and the goal is to maximize the average profit per time unit. Once the processing of a job is completed, it goes on *vacation* and returns to the system, ready to be processed again, only after its vacation is over. This problem has many applications, in production planning, machine maintenance, video-on-demand and database query processing, among others.

We show that the problem is NP-hard already for jobs with unit processing times and unit profits and develop approximation algorithms, as well as optimal algorithms for certain subclasses of instances. We first show that a 1.67-approximation algorithm can be obtained for instances with unit processing times, using known results for windows scheduling. We then present a preemptive greedy algorithm, which yields a ratio of $e/(e-1)$ to the optimal for general instances, with arbitrary processing times and arbitrary profits. For the special case where all jobs have the same (unit) processing times and the same (unit) profits, we present a 1.39-approximation algorithm. For this case we also show that, when the load generated by an instance is sufficiently large (in terms of n and m), any algorithm which uses no intended idle times yields an optimal schedule.

1 Introduction

We consider a scheduling problem in which jobs need to be scheduled repeatedly. The input is a set of m identical parallel machines and a set of n jobs, $J = \{1, \ldots, n\}$, that need to be scheduled on the machines. All the jobs are ready at time $t = 0$. Each job j, is associated with a processing time $p_j \geq 1$, and a window $a_j \geq p_j$; once the processing of j is completed, it goes on *vacation* and returns after $a_j - p_j$ time units, ready to be processed again. Thus, a feasible schedule is one where each machine processes at most one job at any time, and there is a gap of at least $a_j - p_j$ time units between two consecutive executions of job j. The jobs are sequential, i.e., no job can be scheduled on more than one machine at the same time. There is also a profit, b_j, associated with each execution of job j; the goal is to find a feasible schedule

*An extended abstract of this paper appeared in proceedings of the 13th Annual European Symposium on Algorithms (ESA'05).

†Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic, and Dept. of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Praha. E-mail: sgall@math.cas.cz

‡Computer Science Department, The Technion, Haifa 32000, Israel. E-mail: hadas@cs.technion.ac.il

§School of Computer science, The Interdisciplinary Center, Herzliya, Israel. E-mail: tami@idc.ac.il

that maximizes the average profit per time unit. We call this problem *scheduling with vacations (SWV)*. Our problem arises in many practical scenarios where tasks need to be accomplished periodically, but due to setup times or maintenance requirements there must be some gap between two consecutive executions of a task. The following are two of the applications that motivate our study.

Surveillance Camera Scheduling: Consider a system of robots carrying surveillance cameras, which patrol an area periodically. Each robot has a predefined path that it needs to patrol, while recording all the events along this path. Upon completion of each patrol, the robot returns to the controller, where the recorded data is downloaded/processed, and the robot is prepared for its next tour. Each patrol j is associated with a revenue b_j, that is gained once the corresponding robot completes the tour. The controller can handle at most m robots simultaneously, and it takes p_j time units to complete the processing of robot j. The time it takes robot j to traverse its path is $t_j \geq 1$. The goal of the controller is to process the robots in a way that maximizes the profit gained from all robots, throughout the operation of the system. This yields an instance of SWV, where job j has a processing time p_j, and a window $a_j = p_j + t_j$.

Commercial Broadcast: In a broadcasting system which transmits data (e.g., commercials on a running banner) there is some profit associated with the transmission of each commercial. This profit is gained only if some predefined period of time elapsed since the previous transmission. The goal is to broadcast the commercials in a way that maximizes the overall profit of the system. Thus, we get an instance of SWV, where the window of each job corresponds to the time interval between consecutive transmissions of each commercial.

1.1 Our Results and Related Work

The problem of scheduling with vacations belongs to the class of *periodic scheduling* problems, where each job may be scheduled infinite number of times. Periodic scheduling is a well-studied problem. Problems of this type arise in many areas, including production planning, machine maintenance, telecommunication systems, video-on-demand, image and speech processing, robot control/navigation systems, and database query processing. Liu and Layland introduced in [13] the periodic scheduling problem: they considered the goal of scheduling each of the jobs such that the average gap between two executions of job j is a_j, using the minimum number of machines. Another early example is the *chairman assignment* problem [14], in which the number of executions of job j in any prefix of the schedule of length t must be at least $\lfloor t/a_j \rfloor$ and at most $\lceil t/a_j \rceil$. For both problems the *earliest deadline first* algorithm was shown to be optimal. A more restricted form of periodic scheduling, known as *pinwheel scheduling*, requires that job j is scheduled at least once in any sub-sequence of a_j jobs $\{J_{\ell+1}, \ldots, J_{\ell+a_j}\}$ (see, e.g., [10]).

Our problem is different from previously studied variants of periodic scheduling in two major aspects. First, our goal is to maximize the total profit of the server. This may result in lack of fairness. Indeed, in a schedule which maximizes the profit, some jobs may be waiting forever, while other (more profitable) jobs are repeatedly processed as soon as they return from vacation. In other works (e.g., [13, 7, 8, 10]), the performance of a periodic schedule is measured by some fairness criterion, or (e.g. [2, 9]) the two objectives are combined; that is, each job comprises a mandatory and an optional part, with which a non-decreasing reward function is associated. In other variants of the periodic scheduling problem (see, e.g., [1]), the processing of each job incurs a service cost, depending on the time that elapsed since the last service of this job, and the goal is to minimize the cost of the schedule; however, the cost increases with job delays − implying that an optimal solution requires some fairness.

The other major difference from well-studied variants of periodic scheduling is that in our problem, due to the vacation requirement, jobs cannot be processed too often. In the *windows scheduling* problem [4, 5], the parameter associated with each job gives the *maximal* gap between any two executions of job j, however, it is feasible to schedule a job more often. In the *periodic machine scheduling problem (PMSP)* [3], this parameter specifies the *exact* gap between any two executions of j. To the best of our knowledge, there is no earlier work in which only the *minimal* gap between any two executions is given.

We first show (in Section 2) that SWV is NP-hard already for jobs with unit lengths and unit profits. Since the total number of different configurations of the system is finite, an optimal solution can be found (inefficiently) by applying the *buffer scheme* designed for the windows scheduling problem [6]. We present several approximation algorithms for various subclasses of instances. In Section 3 we show that a simple 1.67-approximation algorithm can be obtained for instances with unit processing times, using known results for windows scheduling. We then present (in Section 4) a preemptive greedy algorithm, which yields a ratio of $e/(e-1)$ to the optimal for general instances, with arbitrary processing times and arbitrary profits. For the special case where all jobs have the same (unit) processing times and the same (unit) profits, we present (in Section 5) a 1.39-approximation algorithm. For this case we also show that, when the load generated by an instance is sufficiently large (in terms of n and m), any algorithm which uses no intended idle times yields an optimal schedule. Our approximation algorithms for unit length jobs apply a transformation of the instance to an *aligned instance*, where all window sizes satisfy certain properties. Our approximation technique, based on finding an optimal schedule for the resulting aligned instance, builds on a technique of [5] for the windows scheduling problem. To obtain better approximation ratio, we extend this technique: Our algorithm for unit processing times and unit profits applies the *best align* technique, which finds an aligned instance that yields the best approximation ratio for the original instance (see in Section 5.1).

2 Preliminaries

2.1 Definitions and Notation

Denote by a_j the *scheduling window* (for short, *window*) of job j. The *load* incurred by job j is $\ell(j) = p_j/a_j$. The best service job j can receive is to be processed whenever it returns from vacation (i.e., no waiting time). Thus, the load of job j represents the average processing requirement of j per time unit. The total load of the input is $L(J) = \sum_{j \in J} \ell(j)$. In the special case of unit processing times (i.e., for all j, $p_j = 1$), we get that $\ell(j) = 1/a_j$, and $L(J) = \sum_{j \in J} 1/a_j$. Each job is associated with an additional parameter, b_j, the profit gained from each execution of job j.

We say that a job j is *heavier* than j' if $b_j/p_j > b_{j'}/p_{j'}$, or $b_j/p_j = b_{j'}/p_{j'}$ and $j > j'$. A heavier job achieves larger profit per unit time of its execution; the second part of the definition only breaks ties consistently for the whole instance. Let $J_{\geq j}$ denote the set of jobs at least as heavy as j.

To define the profit of a schedule formally, let $compl_T^S(j)$ denote the number of times job j is completed in the first T time units in schedule S. The profit of schedule S is defined as

$$profit(S) = \liminf_{T \to \infty} \frac{1}{T} \sum_{j=1}^{n} compl_T^S(j) b_j.$$

A schedule is periodic with period p if for any time T, the jobs scheduled at time T are the same as the jobs scheduled at time $T + p$.

Observation 2.1 *Any instance J has a periodic optimal schedule.*

Proof: Define a configuration at time T to consist of the information of how much of each job is currently executed, or how much time is left until the job can start its next execution. An optimal schedule can always be chosen so that for two points of time T and T' with the same configurations, the remainder of the schedule is the same. Since the number of configurations is finite, such a schedule is periodic. ∎

In the rest of the paper we consider only periodic schedules. For a schedule S with period p, in the definition of profit, $\liminf$ can be replaced by $\lim$ which always exists and it is actually equal to the value of the remaining expression for $T = p$. Similarly, it is meaningful to speak about the load (relative frequency) of a job j in S; we denote it by $\ell^S(j)$. Finally, $L^S(J') = \sum_{j \in J'} \ell^S(j)$ denotes the load of a set of jobs $J' \subseteq J$ in a schedule S.

Example: Figure 1 presents two possible schedules on a single machine of an instance consisting of three unit-length jobs $\{i, j, k\}$ having windows $\{2, 3, 4\}$ (that is, vacations $\{1, 2, 3\}$) and profits $\{1, 5, 1\}$ respectively. Schedule (a) has period 4 and profit (=average profit per time slot) $(1+5+1+1)/4 = 2$. Schedule (b) has period 12 and profit $(7 \cdot 1 + 4 \cdot 5)/12 = 2.25$. Note that in the 10-th slot out of the 12 the machine is idle. This idle is inevitable because all the jobs are on vacation during this slot. For the schedule (a), we have $\ell^S(i) = 2/4$, $\ell^S(j) = \ell^S(k) = 1/4$, and taking $J' = \{i, j, k\}$, $L^S(J') = 1$; for the schedule (b) we have $\ell^S(i) = 4/12$, $\ell^S(j) = 4/12$, $\ell^S(k) = 3/12$, and $L^S(J') = 11/12$.

The next lemma is used to compare our schedules to an optimal schedule S^*.

Lemma 2.2 *Let S^* be an arbitrary schedule for an instance J and let $R \geq 1$.*

(i) Suppose that for a schedule S of J, and for any job j, $L^S(J_{\geq j}) \geq L^{S^*}(J_{\geq j})/R$. Then $profit(S) \geq profit(S^*)/R$.

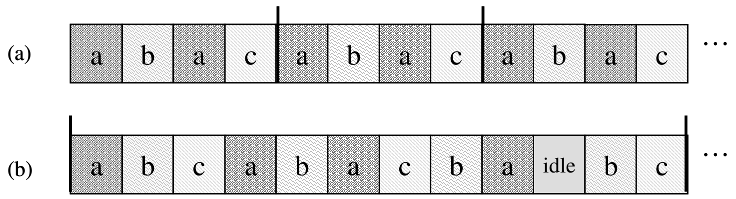

Figure 1: Two possible periodic schedules of the same instance.

(ii) Suppose that for a distribution of schedules $\mathcal{S}$ for J, and for any job j, $\mathbf{Exp}_{S\in\mathcal{S}}[L^S(J_{\geq j})] \geq L^{S^*}(J_{\geq j})/R$. Then one of the schedules S in the support of $\mathcal{S}$ satisfies profit$(S) \geq$ profit$(S^*)/R$.

Proof: The lemma follows from the fact that the profit of any schedule S can be computed as $\int_{\beta=0}^{\infty} L^S(J_\beta)\, d\beta$, where $J_\beta = \{j \in J \mid b_j \geq \beta\}$. ∎

2.2 Hardness Results

The hardness of *SWV* with a single machine, unit profits, and unit processing times is shown by a reduction from the *periodic machine scheduling problem (PMSP)*, which is known to be NP-hard (see in [3]). In PMSP, there is a set of n machines; the j-th machine requires maintenance every a_j time units, where $\sum_{j=1}^{n} 1/a_j \leq 1$. The maintenance of any machine takes one time unit, and at any time only one machine can be maintained. It is assumed that the first maintenance of machine j can be done in any of the first a_j time slots. The goal is to find a maintenance schedule that satisfies *exactly* all the requirements, that is, for all j, machine j is maintained exactly once in any window of a_j time slots.

Theorem 2.3 *SWV is NP-hard, even with a single machine, unit profits and unit processing times.*

Proof: Given an instance of PMSP with n machines in which the j-th machine requires maintenance every a_j time units, construct an instance of *SWV* with unit profits, unit processing times, and scheduling windows a_j for jobs $j = 1, \ldots, n$. Since $\sum_{j=1}^{n} 1/a_j \leq 1$, there is a feasible schedule for PMSP if and only if the average profit per time-unit from job j is exactly $1/a_j$, that is, there is a schedule of the *SWV* instance with average profit $\sum_{j=1}^{n} 1/a_j$. ∎

Remark 2.1 When $L(J) \geq 1$, the problem is still NP-hard for unit profits and unit processing times. The proof is similar to the hardness proof for the windows scheduling problem given in [5]. An instance J with $L(J) < 1$ and unit profits can be extended to the instance J' with $L(J') = 1$ in a way that does not affect the schedule of the original jobs. Let $A = LCM(a_1, a_2, ..., a_n)$, that is, each of the original windows divides A. Add to J dummy jobs, each having window A, such that the total load is 1. Then a schedule of J' with average profit 1 induces a schedule of the original instance with average profit $\sum_{j=1}^{n} 1/a_j$, and vice versa. For arbitrary profits, a single dummy job with $b_j = \varepsilon$ and $a_j = 1$ is sufficient.

2.3 An Optimal Super-Polynomial Algorithm

The general buffer scheme [6] is a tool designed for solving optimally the windows scheduling problem. We explain below how it can be modified to solve optimally the problem of scheduling with vacations. We consider unit-length jobs and arbitrary profits. The algorithm can be extended to handle jobs with arbitrary lengths similar to the extension for windows scheduling shown in [6]. The buffer scheme is based on representing the system as a nondeterministic finite state machine, in which any directed cycle corresponds to a valid periodic schedule and vice-versa. Let $a^* = \max_j\{a_j\}$. The state of a schedule is represented using a set of buffers, $B_0, B_1, \ldots, B_{a^*-1}$. Each job is stored in some buffer. A job is stored in B_i when i time slots remain till the end of its vacation. Initially, all jobs are ready to be processed so they are all stored in B_0. In each iteration, the scheme schedules at most m jobs from B_0 and updates the content of the buffers:

- For all $i > 0$, all jobs stored in B_i are moved to B_{i-1} (note that none of them is scheduled).
- Each scheduled job j is moved from B_0 to B_{a_j-1} (this ensures that at least $a_j - 1$ time slots will elapse before j is scheduled again).

Note that the total number of buffer configurations is at most $\Pi_j a_j$. In the state machine, each configuration is represented by a vertex, and there is an edge connecting configuration f to configuration g if it is possible to move from the state corresponding to f to the state corresponding to g in a single time slot. Each edge $e = (f, g)$ has a weight b_e whose value is the profit gained by moving from f to g; this is the total profit of the jobs selected for execution. A periodic schedule with cycle t corresponds to a cycle of length t in the state machine. The profit of this schedule is given by the average edge weight along this cycle. Thus, an optimal schedule corresponds to a cycle with maximal mean weight. Such a cycle can be found by using the classic *Max Cycle Mean* algorithm of Karp [11]. The running time is polynomial in the graph size, which is polynomial in $\Pi_j a_j$. Clearly, this algorithm is applicable only for small instances, however, since the problem is NP-hard, it is unlikely to admit a polynomial time algorithm for general instances.

3 Approximating by Aligned Instances

In this section we study a natural approach for handling instances with unit processing times, inspired by previous work on *window scheduling* (see, e.g., [5]). We call an instance *aligned* if there exists an integer q such that for each j, $a_j = q \cdot 2^{\alpha_j}$ for some integer $\alpha_j \geq 0$, or $a_j = 1$. (Note that this generalizes the case where all the windows are powers of 2.) The following result is due to [5].

Theorem 3.1 *An optimal solution for an aligned instance can be computed in polynomial time.*

For completeness, we describe such an optimal solution. Note that this solution works also if we start not with an empty schedule but with some schedule with period q, where some slots are already used by other jobs.

Schedule the jobs in order of increasing windows, always keeping the period of the schedule equal to the maximal window processed so far. The machines are utilized in an arbitrary order. When a new job is processed, if needed, increase the period by doubling the period and repeating the current schedule, until the period is equal to the currently processed window. Then, if possible, schedule the current job in an empty slot on some machine. Otherwise, if the current job is heavier than some of the scheduled jobs, run this job in a slot of the lightest scheduled job. (This light job can still be scheduled in some of the slots, due to a possible previous doubling of the period.)

An alternative view of this schedule S will be useful. Let j be the heaviest job such that $L(J_{\geq j}) \geq m$. Then the schedule above has the property that all jobs j' heavier than j are scheduled at their maximal rate, i.e., $\ell^S(j') = \ell(j')$, and no jobs lighter than j occur in the schedule. The rate of the borderline job j is selected such that all the rates sum to m.

It follows that a simple 2-approximation algorithm can be achieved by rounding the window of each job to the next higher power of 2 and using the optimal algorithm for aligned instances.

Below we give an algorithm that refines this idea and achieves better approximation ratio. Later, in Section 5, we further elaborate this idea to obtain our best algorithm in the special case where also the profits are the same for each job. All these algorithms are deterministic: Randomization is used only in the analysis.

3.1 A 1.67-approximation Algorithm

> **Algorithm** SimpleAlign
> Produce the instance J' where any window $a_j > 1$ is rounded up to the next power of 2.
> Produce instance J'' where any window $a_j > 1$ is rounded up to the next number of the form $3 \cdot 2^\alpha$, for some integer $\alpha \geq 0$.
> Find an optimal schedule S' for J'.
> Find an optimal schedule S'' for J''.
> Choose the better of S' and S''.

Theorem 3.2 SimpleAlign *is a* 1.67*-approximation algorithm for unit processing times and arbitrary profits.*

Proof: Using Theorem 3.1, SimpleAlign runs in polynomial time.

Consider a distribution which chooses S' with probability 2/5 and S'' with probability 3/5. We prove that the condition of Lemma 2.2 is satisfied with $R = 5/3$. We use the following claim, where by $\ell^{J'}(j)$ we denote the load of a job j rounded as in the instance J', and similarly for $\ell^{J''}$ and J''.

Claim 3.3 *For any job j, $\frac{2}{5}\ell^{J'}(j) + \frac{3}{5}\ell^{J''}(j) \geq \ell(j)/R$.*

Proof: We consider two cases:

(i) If for some integer α, $2^{\alpha+1} \leq a_j \leq 3 \cdot 2^{\alpha}$, then

$$\frac{2}{5}\ell^{J'}(j) + \frac{3}{5}\ell^{J''}(j) \geq \frac{2}{5} \cdot \frac{1}{2^{\alpha+2}} + \frac{3}{5} \cdot \frac{1}{3 \cdot 2^{\alpha}} = \frac{3}{5 \cdot 2^{\alpha+1}} \geq \frac{\ell(j)}{R}.$$

(ii) Otherwise, for some integer α, $3 \cdot 2^{\alpha} < a_j < 2^{\alpha+2}$, in which case

$$\frac{2}{5}\ell^{J'}(j) + \frac{3}{5}\ell^{J''}(j) \geq \frac{2}{5} \cdot \frac{1}{2^{\alpha+2}} + \frac{3}{5} \cdot \frac{1}{3 \cdot 2^{\alpha+1}} = \frac{1}{5 \cdot 2^{\alpha}} \geq \frac{\ell(j)}{R}.$$

■

Now, given an optimal schedule S^* and a job j, we prove that Lemma 2.2 can be applied here. We distinguish between two cases.

(a) First, assume that both $L^{S'}(J_{\geq j}) < m$ and $L^{S''}(J_{\geq j}) < m$. Then $L^{S'}(J_{\geq j}) = \sum_{j \in J_{\geq j}} \ell^{J'}(j)$, by the construction of an optimal schedule for aligned instances, and similarly for S''. From Claim 3.3, we obtain $\mathbf{Exp}[L^S(J_{\geq j})] \geq L(J_{\geq j})/R \geq L^{S^*}(J_{\geq j})/R$.

(b) Otherwise it must be the case that $L^{S'}(J_{\geq j}) = m$ or $L^{S''}(J_{\geq j}) = m$ (note that the load in a schedule cannot be larger than m). Either way, it implies that $L(J_{\geq j}) \geq m$ in the original instance. In both J' and J'', the size of each window is at most doubled, therefore $L^{S'}(J_{\geq j}) \geq m/2$ and $L^{S''}(J_{\geq j}) \geq m/2$. Then the average load is bounded by $\mathbf{Exp}[L^S(J_{\geq j})] \geq 2/5 \cdot m + 3/5 \cdot m/2 = 7m/10 \geq m/R \geq L^{S^*}(J_{\geq j})/R$.

The proof is completed by an application of Lemma 2.2. ■

4 A Greedy Algorithm

In the preemptive scheduling model which we use, the execution of a preempted job can be resumed at any integral time. The vacation-constraint refers only to the gaps between *different* executions of a job. This preemptive model includes the previous model of unit-length jobs. Recall that m is the number of machines.

We describe below a simple greedy algorithm for preemptive scheduling of arbitrary-length jobs, which improves upon the algorithm in Section 3.

Let us note that if we further simplify the preemptive model so that preemption is possible at any fractional time, the problem becomes trivial. This model allows us to simulate time-sharing. Thus we can simply accept the heaviest jobs until the load is m and then run each job at a constant rate.

Algorithm GREEDY

At any time t, schedule for one time unit the m heaviest available jobs.

It is easy to see that GREEDY always generates a periodic schedule, as the possible number of different states is finite.

Theorem 4.1 GREEDY *is an $e/(e-1)$-approximation algorithm.*

Proof: Sort the jobs in non-decreasing order of the ratio b_j/p_j. Using this order renumber the jobs such that J_1 is the lightest and J_n is the heaviest. Let S be the schedule of Greedy, then we define $L^S(J_{>j}) = L^S(J_{\geq j+1})$. We use in the proof the next two Claims.

Claim 4.2 *For any job j, $\dfrac{\ell^S(j)}{\ell(j)} \geq 1 - \dfrac{L^S(J_{>j})}{m}$.*

Proof: If the average distance between two executions of j in S is $a_j + d$, then heavy jobs must occupy d out of $a_j + d$ time slots on average, and thus $L^S(J_{>j})/m \geq d/(a_j + d)$. Now we have that $\ell^S(j)/\ell(j) = a_j/(a_j + d) = 1 - d/(a_j + d)$. This completes the proof. ∎

Claim 4.3 *For any job j, $\dfrac{L^S(J_{\geq j})}{m} \geq 1 - e^{-L(J_{\geq j})/m}$.*

Proof: We prove the claim using backward induction on j. Specifically, we show that if the claim holds for $J_{>j}$ (in place of $J_{\geq j}$), then it also holds for $J_{\geq j}$.

For the base case we take $j = n$ (i.e., the heaviest job), then $J_{>j}$ is defined as the empty set and the induction hypothesis trivially holds.

Now take any $j \leq n$ and assume that the claim holds for $J_{>j}$. Denote $x = L(J_{>j})/m$, $y = L^S(J_{>j})/m$, and $z = \ell(j)/m$. Using Claim 4.2 in the first inequality and the induction hypothesis in the second inequality, we have

$$\begin{aligned}
\frac{L^S(J_{\geq j})}{m} &= \frac{L^S(J_{>j})}{m} + \frac{\ell^S(j)}{m} \\
&\geq y + z(1-y) = y(1-z) + z \\
&\geq (1 - e^{-x})(1-z) + z = 1 - e^{-x}(1-z) \\
&\geq 1 - e^{-x}e^{-z} = 1 - e^{-L(J_{\geq j})/m}.
\end{aligned}$$

The third inequality follows from the fact that $e^{-z} \geq 1 - z$ for all $z \geq 0$, and the last equality holds since $L(J_{\geq j})/m = x + z$. ∎

To complete the proof of the theorem, we use Lemma 2.2 with the desired ratio $R = 1/(1 - e^{-1}) = e/(e-1)$. To justify the assumption of Lemma 2.2, let j be arbitrary and let $x = L(J_{\geq j})/m$.

If $x \geq 1$, then we have $L^S(J)/m \geq 1 - e^{-1}$, from Claim 4.3. We are done, since the optimum (i.e., the load in S^*) is at most m.

Otherwise, $1 - e^{-x} \geq (1 - e^{-1})x$, as the left-hand side is concave function for $0 \leq x \leq 1$. Thus, from Claim 4.3, we have, $L^S(J_{\geq j}) \geq (1 - e^{-1})L(J_{\geq j}) \geq (1 - e^{-1})L^{S^*}(J_{\geq j})$, and we are done as well. ■

We now show that our analysis of GREEDY is tight: In fact, $e/(e-1) \approx 1.58$ is a lower bound for Greedy already for unit-length jobs. Consider the following example. Let a be an integer and let $A = \{a+1, a+2, \ldots, \lfloor ea \rfloor\}$. We have $\sum_{i \in A} 1/i = \ln \lfloor ea \rfloor - \ln a \leq \ln e = 1$ and for a sufficiently large, the sum is arbitrarily close to 1. Let $m = LCM\{a+1, a+2, \ldots, \lfloor ea \rfloor\}$. That is, each of the numbers $a+1, a+2, \ldots, ea$ divides m. The instance consists of m machines, and m jobs having window $a_j = i$ for each $i = a+1, a+2, \ldots, \lfloor ea \rfloor$. The profits are all almost equal to 1, with a slight preference to jobs having long vacation.

The GREEDY schedule: In the first slot, the most profitable jobs are those having vacation $\lfloor ea \rfloor$, so these m jobs are scheduled first on each of the m machines. Next, GREEDY schedules on all m machines the jobs whose vacation is $\lfloor ea \rfloor - 1$, and so on. The resulting schedule has period $\lfloor ea \rfloor$ repeating on each machine $[\lfloor ea \rfloor, \lfloor ea \rfloor - 1, ..., (a+1), *, \ldots, *]$, where stars denote a consecutive idle slots. The average profit per slot on each of the machines is therefore arbitrarily close to $(ea - a)/(ea) = (e-1)/e$.

An optimal schedule: For each i, m/i machines schedule with no idle time all the jobs having the window i. Since $\sum_{i \in A} 1/i \approx 1$, there are enough jobs to 'saturate' this way all the machines with no idle times. Therefore the average profit per slot is arbitrarily close to 1 and the approximation ratio tends to $e/(e-1)$.

5 Unit Processing Times and Unit Profits

In this section we consider the special case where all jobs have the same (unit) processing times and the same (unit) profit. Thus, the average profit is equal to the load of the schedule. In particular, if the schedule has no idle time, it is optimal.

5.1 A 1.39-approximation Algorithm

As in Section 3, we use in our algorithm *aligned instances*; however, instead of using two schedules with periods 2 and 3 times a power of 2, the next algorithm uses one of k schedules with periods $q = k+1, k+2, \ldots, 2k$ times a power of 2. For a large but constant k, the approximation ratio approaches $2 \ln 2 \approx 1.39$. The jobs with large windows are rounded similarly to the rounding in Section 3.1.

The jobs with small windows are handled separately. Given a constant q and a set of jobs with windows at most q, we find an optimal schedule with period q in polynomial time. If the number of machines is constant, then the number of such schedules is constant and we can find an optimal one simply by exhaustive search. (Note that we do not need to distinguish between different jobs having the same window, as they have identical profits.)

For arbitrary m, we use Lenstra's polynomial algorithm for integer programming in fixed dimension (see [12]), similarly to its other applications in scheduling. Since q is a constant, we have a constant number of job types specified by their windows, and for each type of jobs we have a constant number of patterns specifying in which time slots it runs. The variables of the integer program correspond to the number of jobs of each type following each pattern; the number of these variables is a constant exponential in q. The constraints specify that (i) the number of scheduled jobs of each type equals to the number of such jobs in the instance, and that (ii) the number of jobs scheduled in any given time unit is at most m. Feasible integral solutions then correspond to schedules in a straightforward way. Given an instance, the integer program can be produced in linear time, and solved in time polylogarithmic in n and m (with a multiplicative constant doubly exponential in q).

Algorithm BestAlign
Let $\varepsilon > 0$ be given.
Let $K = \lceil 1/\varepsilon \rceil$.
For all $k \in \{K, 4K, \ldots, 4^x K, \ldots, 4^K K\}$ do
  For all $q = \{k+1, k+2, \ldots, 2k\}$ do
    Let J' be the set of jobs with windows at most $k/4$.
    Let J'' be the set of jobs with windows larger than k.
    Generate from J'' the aligned instance J''' by rounding up the window of each job to the next number of the form $q2^\alpha$, for some integer $\alpha \geq 0$.
    Find the best schedule with period q for J'.
    Schedule J''' in the remaining slots, using an optimal algorithm for aligned instances.
Output the best schedule over all values of k and q.

Theorem 5.1 BestAlign *is* $(2\ln 2 + O(\varepsilon))$*-approximation algorithm for the jobs with unit processing times and unit profits. Thus, there exists an* R*-approximation algorithm for any* $R > 2\ln 2 \approx 1.39$.

Proof: We first note that BestAlign can be implemented in polynomial time. Indeed, by the above discussion, we can find in polynomial time an optimal schedule for J', since q is a constant. Also, we can schedule optimally the aligned instance J'', using Theorem 3.1.

Let S^* be an optimal schedule for a given instance. For the proof, fix a value of k among those used in the algorithm, so that the contribution of the jobs with windows between $k/4$ and k to $L(S^*)$ is at most $\varepsilon \cdot L(S^*)$. Since there are $\lceil 1/\varepsilon \rceil$ choices of k, one of them has a sufficiently small contribution. This defines the sets of jobs J' and J''. Let S' and S'' denote the schedule S^* restricted to J' and J'', respectively, i.e., all the other jobs are simply removed from the schedule.

For any q, let S_q be the schedule generated by BestAlign for this choice of q and for k fixed as above. If any of S_q has load m, it is optimal for J and the theorem follows. Thus, for the remaining proof we can assume that S_q always schedules all the jobs in J''' with their maximal load, i.e., $L^{S_q}(J''') = L(J''')$.

We prove that one of the schedules S_q has load at least $3/4 \cdot L^{S'}(J') + L(J'')/(2\ln 2 + O(\varepsilon))$. The theorem then follows since, by the choice of k, $L^{S^*}(J) \leq (1+O(\varepsilon))(L^{S'}(J') + L^{S''}(J'')$;

furthermore, $L^{S''}(J'') \le L(J'')$, and $2\ln 2 > 4/3$.

The schedule generated by BestAlign for J' has load at least $3/4$ of $L(S')$: Indeed, if we take in S' a segment of length $q - k/4$ with the maximal load and append to it $k/4$ empty slots, we get a schedule with period q and load at least $3/4$ of $L(S')$. The optimal schedule for J' with period q chosen by BestAlign has at least the same load.

Now we complete the proof of the theorem by showing that for some distribution over the schedules S_q, $\mathbf{Exp}[L^{S_q}(J''')] \ge L(J'')/(2\ln 2 + O(\varepsilon))$. Define the probability distribution so that the probability of choosing S_q is $\pi_q = X/(q-1)$, where X is chosen so that the sum of the probabilities is 1, i.e.,

$$\left(\frac{1}{k} + \ldots + \frac{1}{2k-1}\right) X = 1. \tag{1}$$

We proceed with the proof job by job, as in Theorem 3.2. Since S_q schedules all the jobs with maximal rate, it is sufficient to prove for each job j that its expected load in J''' (i.e., after rounding) is at least $\ell(j)/(2\ln 2+O(\varepsilon))$. Assume that the integers $\alpha \ge 0$ and $t \in \{k, \ldots, 2k-1\}$ satisfy $t2^\alpha < a_j \le (t+1)2^\alpha$ (for each j, there exist unique values of t and α). Then the average load of j in J''' is

$$\begin{aligned}
\sum_{q=k+1}^{t} \pi_q \cdot \frac{1}{q2^{\alpha+1}} + \sum_{q=t+1}^{2k} \pi_q \cdot \frac{1}{q2^{\alpha}} &= t\left(\sum_{q=k+1}^{t} \pi_q \cdot \frac{1}{2q} + \sum_{q=t+1}^{2k} \pi_q \cdot \frac{1}{q}\right)\frac{1}{t2^\alpha} \\
&\ge t\left(\sum_{q=k+1}^{t} \pi_q \cdot \frac{1}{2q} + \sum_{q=t+1}^{2k} \pi_q \cdot \frac{1}{q}\right)\ell(j).
\end{aligned}$$

It remains to bound the coefficient on the right-hand side. By the definition of π_q, we have

$$\frac{\pi_q}{q} = \frac{X}{(q-1)q} = \left(\frac{1}{q-1} - \frac{1}{q}\right) X. \tag{2}$$

Using (2) and telescoping the sums we have

$$t\left(\sum_{q=k+1}^{t} \pi_q \cdot \frac{1}{2q} + \sum_{q=t+1}^{2k} \pi_q \cdot \frac{1}{q}\right) = tX\left(\frac{1}{2k} - \frac{1}{2t} + \frac{1}{t} - \frac{1}{2k}\right) = \frac{X}{2}.$$

Using (1), we have

$$\frac{2}{X} = 2\left(\frac{1}{k} + \ldots + \frac{1}{2k-1}\right) \le 2\int_k^{2k} \frac{dx}{x} + \frac{2}{k} = 2(\ln(2k) - \ln k) + \frac{2}{k} \le 2\ln 2 + O(\varepsilon).$$

This completes the proof of the theorem. ∎

5.2 Instances with a Large Load

In the case where $p_j = b_j = 1$, the goal is to maximize the utilization of the machine. A simple observation is that the best one can expect to achieve is a schedule in which the machines are busy all the time. In this section we focus on two cases where such a schedule can be generated.

Our analysis of SIMPLEALIGN shows that if $L(J) \geq 5m/3$, then SIMPLEALIGN produces a schedule with load m: One of the schedules S' and S'' is guaranteed to have at least this load, and no larger load is possible.

The next theorem gives a condition for the optimality of *any* greedy algorithm in scheduling jobs with unit lengths and unit profits. In the resulting schedule, we get full utilization of all machines. In other words, no machine is ever idle, regardless of the jobs selected to be scheduled at any time slot.

Theorem 5.2 *Suppose that $n = km + r$ for some $r < m$ (i.e., $k = \lfloor n/m \rfloor$ and $r = n \bmod m$). If $L(J) > mH_k - 1 + (r+1)/(k+1)$ then any algorithm with no intended idle times achieves the optimal profit. In particular, for $m = 1$, the condition is $L(J) > H_{n+1} - 1$.*

Proof: We show that if a machine is idle, $L(J)$ must be small. Assume that some machine is idle for the first time at t. Thus, none of the n jobs is available, meaning that all jobs are either processed on the other machines or on vacation. Order the jobs by the last execution up to time t, including possible executions on the other $m-1$ machines at time t. The last $m-1$ jobs in this order are possibly executed at time t, we have no bound on their vacation, so each can contribute as much as 1 to $L(J)$. The previous m jobs in this order are scheduled no later than at time $t-1$, they are on vacation at time t, thus they have $a_j \geq 2$ and contribute at most $1/2$ each to $L(J)$. Similarly, the previous m jobs are scheduled no later than at time $t-2$ and contribute at most $1/3$ each to $L(J)$, and so on. The last $r+1$ jobs contribute at most $1/(k+1)$ each. Thus, the total load of the input is

$$L(J) \leq m - 1 + m\left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}\right) + \frac{r+1}{k+1} = mH_k - 1 + \frac{r+1}{k+1}.$$

Therefore, $L(J) > mH_k - 1 + (r+1)/(k+1)$ implies that there is no idle time. ■

Acknowledgment We thank Gerhard Woeginger for stimulating discussions on this paper.

References

[1] S. Anily, J. Bramel, Periodic scheduling with service constraints. *Operations Research*, Vol. 48, pp. 635-645, 2000.

[2] H. Aydin, R. Melhem, D. Mosse, P. Mejia-Alvarez, Optimal Reward-Based Scheduling of Periodic Real-Time Tasks. In 20th IEEE Real-Time Systems Symp., 1999.

[3] A. Bar-Noy, R. Bhatia, J. Naor, B. Schieber, Minimizing Service and Operation Costs of Periodic Scheduling. *Math. Oper. Res.* 27(3): 518–544, 2002.

[4] A. Bar-Noy and R. E. Ladner. Windows Scheduling Problems for Broadcast Systems. *SIAM Journal on Computing (SICOMP),* Vol. 32(4):1091–1113, 2003.

[5] A. Bar-Noy, R. E. Ladner, T. Tamir. Windows Scheduling as a Restricted Version of Bin Packing. *ACM Transactions on Algorithms* (TALG), Vol. 3(3), 2007.

[6] A. Bar-Noy, J. Christensen, R. E. Ladner, T. Tamir. A General Buffer Scheme for the Windows Scheduling Problem. In *Proc. of WEA*, 2005.

[7] S. K. Baruah, N. K. Cohen, C. G. Plaxton, D. A. Varvel, Proportionate Progress: A Notion of Fairness in Resource Allocation. *Algorithmica*, 15(6), 600–625, 1996.

[8] S. K. Baruah, S-S. Lin. Pfair Scheduling of Generalized Pinwheel Task Systems. *IEEE Trans. on Comp.,* Vol. 47, 812–816, 1998.

[9] J. Chung, J.W.S. Liu, K. Lin. Scheduling Periodic Jobs that Allow Imprecise Results. *IEEE Trans. on Comp.,* Vol. 39 (9),pp. 1156–1174, 1990.

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

[11] R. M. Karp. A Characterization of the Minimum Cycle Mean in a Digraph. *Discrete Math.*, 23:309-311, 1978.

[12] H.W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research,* Vol. 8, 538–548, 1983.

[13] C. L. Liu, J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM,* Vol. 20, No. 1, 46-61, 1973.

[14] R. Tijdeman. The Chairman assignment Problem. *Discrete Mathematics,* Vol. 32, 323-330, 1980.