Preemptive Scheduling of Equal-Length Jobs to Maximize Weighted Throughput

Philippe Baptiste * Marek Chrobak† Christoph Dürr‡ Wojciech Jawor†
Nodari Vakhania§

October 24, 2003

Abstract

We study the problem of computing a preemptive schedule of equal-length jobs with given release times, deadlines and weights. Our goal is to maximize the *weighted throughput*, which is the total weight of completed jobs. In Graham's notation this problem is described as $(1|r_j; p_j{=}p; \mathrm{pmtn}|\sum w_j U_j)$. We provide an $O(n^4)$-time algorithm for this problem, improving the previous bound of $O(n^{10})$ by Baptiste [Bap99b].

1 Introduction

We study the following scheduling problem. We are given a set of n jobs of the same integer length $p \geq 1$. For each job j we are also given three integer values: its release time r_j, deadline d_j and weight $w_j \geq 0$. Our goal is to compute a preemptive schedule that maximizes the *weighted throughput*, which is the total weight of completed jobs. Alternatively, this is sometimes formulated as minimizing the weighted number of late jobs. In Graham's notation, this scheduling problem is described as $(1|r_j; p_j{=}p; \mathrm{pmtn}|\sum w_j U_j)$, where U_j is a 0-1 variable indicating whether j is completed or not in the schedule.

Most of the literature on job scheduling focuses on minimizing makespan, lateness, tardiness, or other objective functions that depend on the completion time of all jobs. Our work is motivated by applications in real-time overloaded systems, where the total workload often exceeds the capacity of the processor, and where the job deadlines are critical, in the sense that the jobs that are not completed by the deadline bring no benefit and may as well be removed from the schedule altogether. In such systems, a reasonable goal is to maximize the throughput, that is, the number of executed tasks. In more general situations, some jobs may be more important than other. This can be modeled by assigning weights to the jobs and maximizing the weighted throughput (see, for example, [KS95]).

*CNRS LIX, Ecole Polytechnique, 91128 Palaiseau, France. baptiste@lix.polytechnique.fr.
†Department of Computer Science, University of California, Riverside, CA 92521. Supported by NSF grants CCR-9988360 and CCR-0208856. {marek,wojtek}@cs.ucr.edu.
‡Laboratoire de Recherche en Informatique, Université Paris-Sud, 91405 Orsay, France. durr@lri.fr. Supported by the EU 5th framework programs QAIP IST-1999-11234 and RAND-APX IST-1999-14036, and by CNRS/STIC 01N80/0502 and 01N80/0607 grants.
§Facultad de Ciencias, Universidad Autonoma del Estado de Morelos, 62251 Cuernavaca, Morelos, Mexico. nodari@servm.fc.uaem.mx. Supported by CONACyT-NSF cooperative research grant E120.19.14.

The above problem ($1|r_j; p_j{=}p; \text{pmtn}|\sum w_jU_j$) was studied by Baptiste [Bap99b], who showed that it can be solved in polynomial time. His algorithm runs in time $O(n^{10})$. In this paper we improve his result by providing an $O(n^4)$-time algorithm for this problem.

$1\mid p_i{=}p; r_j; \text{pmtn}\mid \sum U_j$ $O(n \log n)$ [Law94]	$1\mid r_j; \text{pmtn}\mid \sum U_j$ $O(n^5)$ [Law90] $O(n^4)$ [Bap99a] $2\mid r_j; \text{pmtn}\mid \sum U_j$ NP-hard [DLW92]		$1\mid p_i{=}p; r_j\mid \sum U_j$ $O(n^3 \log n)$ [Car81]
$1\mid p_i{=}p; r_j; \text{pmtn}\mid \sum w_jU_j$ $O(n^4)$ [this paper] was $O(n^{10})$ [Bap99b] if $r_i < r_j \Rightarrow w_i \geq w_j$ $O(n \log n)$ [Law94]	$1\mid r_j; \text{pmtn}\mid \sum w_jU_j$ NP-hard [GJ79] pseudo-polynomial [Law90]	$P\mid p_i{=}p; \text{pmtn}\mid \sum w_jU_j$ NP-hard [BK99] $Pm\mid p_i{=}p; \text{pmtn}\mid \sum w_jU_j$ $O(n^{2^m m!})$ [Bap00] $O(nm(\max d_j)^m)$ [Bap00]	$1\mid p_i{=}p; r_j\mid \sum w_jU_j$ $O(n^7)$ [Bap99b] $Pm\mid p_i = p; r_j\mid \sum w_jU_j$ $O(n^{6m+1})$ [BBKT02]

Figure 1: Complexity of some related throughput maximization problems.

Figure 1 shows some complexity results for related scheduling problems where the objective function is to maximize throughput. A more extensive overview can be found at Brucker and Knust's website [BK]. (That website, however, only categorizes problems as NP-complete, polynomial, pseudo-polynomial or open, without describing their exact time complexity.)

2 Preliminaries

Terminology and notation. We assume that the jobs on input are numbered $1, 2, \ldots, n$. All jobs have the same integer length $p \geq 1$. Each job j is specified by a triple (r_j, d_j, w_j) of integers, where r_j is the release time, d_j is the deadline, and $w_j \geq 0$ is the weight of j. Without loss of generality, we assume that $d_j \geq r_j + p$ for all j and that $\min_j r_j = 0$.

Throughout the paper, by a *time unit* t we mean a time interval $[t, t+1)$, where t is an integer. A *preemptive schedule* (or, simply, a *schedule*) S is a function that assigns to each job j a set $S(j)$ of time units when j is executed. Here, the term "preemption" refers to the fact that the time units in S may not be consecutive. We require that S satisfies the following two conditions:

(sch1) $S(j) \subseteq [r_j, d_j)$ for each j (jobs are executed between their release times and deadlines.)

(sch2) $S(i) \cap S(j) = \emptyset$ for $i \neq j$ (at most one job is executed at a time.)

If $t \in S(j)$ then we say that (a unit of) j is scheduled or executed at time unit t. If $|S(j)| = p$, then we say that S *completes* j. The completion time of j is $C_j = 1 + \max S(j)$. Without loss of generality, we will be assuming that each job j is either completed ($|S(j)| = p$) or not executed at all ($S(j) = \emptyset$).

The *throughput* of S is the total weight of jobs that are completed in S, that is $w(S) = \sum_{|S(j)|=p} w_j$. Our goal is to find a schedule of all jobs with maximum throughput.

For a set of jobs $\mathcal{J}$, by $w(\mathcal{J}) = \sum_{j \in \mathcal{J}} w_j$ we denote the total weight of $\mathcal{J}$. Given a set of jobs $\mathcal{J}$, if there is a schedule S that completes all jobs in $\mathcal{J}$, then we say that $\mathcal{J}$ is *feasible*. The restriction of S to $\mathcal{J}$ is called a *full schedule* of $\mathcal{J}$.

Earliest-deadline schedules. For two jobs j, k, we say that j is *more urgent* than k if $d_j < d_k$. It is well-known that if $\mathcal{J}$ is feasible, then $\mathcal{J}$ can be fully scheduled using the following *earliest-deadline rule*: at every time step t, execute the most urgent job among the jobs that have been released by time t but not yet completed. Ties can can be broken arbitrarily, but consistently, for example, always in favor of lower numbered jobs. If S is any schedule (of all jobs), then we say that S is earliest-deadline if its restriction to the set of executed jobs is earliest-deadline.

Since any feasible set of jobs $\mathcal{J}$ can be fully scheduled in time $O(n \log n)$ using the earliest-deadline rule, the problem of computing a schedule of maximum throughput is essentially equivalent to computing a maximum-weight feasible set.

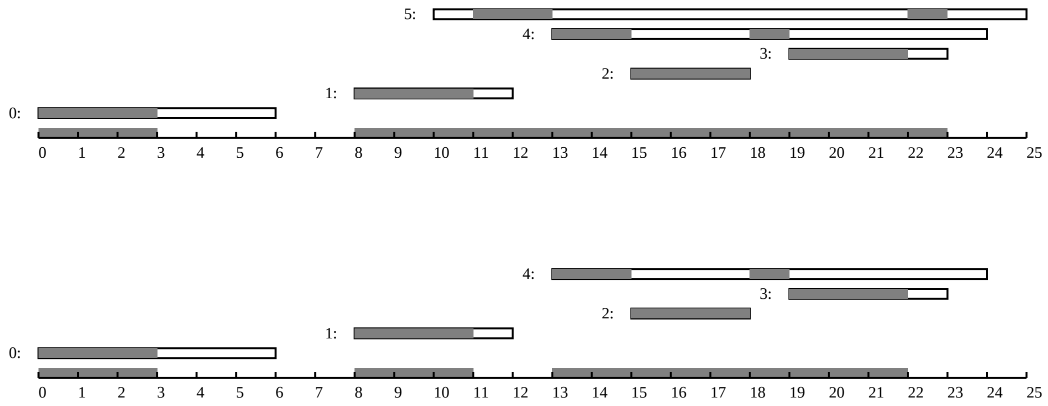

Figure 2: Examples of earliest-deadline schedules with $p = 3$. The rectangles represent intervals $[r_j, d_j)$, and the shaded areas show time units where jobs are executed. The first schedule consists of two distinct blocks. After removing the least urgent job 5, the second block splits into several smaller blocks.

Each earliest-deadline schedule S has the following structure. The time axis is divided into busy intervals (when jobs are being executed) called *blocks* and idle intervals called *gaps*. Each block is an interval $[r_i, C_j)$ between a release time r_i and a completion time C_j, and it satisfies the following two properties: (b1) all jobs executed in this block are not released before r_i, and (b2) C_j is the first completion time after r_i such that all jobs in S released before C_j are completed at or before C_j. Note that $C_j - r_i = ap$, for a equal to the number of jobs executed in this block. Figure 2 shows two examples of earliest-deadline schedules.

In some degenerate situations, where the differences between release times are multiples of p, a gap can be empty, and the end C_j of one block then equals the beginning r_m of the next block.

The above structure is recursive, in the following sense. Let k be the least urgent job scheduled in a given block $[r_i, C_j)$. Then the last completed job is k. Also, when we remove job k from the schedule, without any further modifications, we obtain again an earliest-deadline-schedule for the set of remaining jobs (See Figure 2). The interval $[r_i, C_j)$ may now contain several blocks of this new schedule.

2.1 An $O(n^4)$-Time Algorithm

We assume that the jobs are ordered $1, 2, \ldots, n$ according to non-decreasing deadlines, that is $d_1 \leq d_2 \leq \ldots \leq d_n$. Without loss of generality we may assume that job n is a "dummy" job with $w_n = 0$ and $r_n = d_{n-1}$ (otherwise, we can add one such additional job). We use letters $i, j, k, l \in [1, n]$ for job identifiers, and $a, b \in [0, n]$ for numbers of jobs.

Given an interval $[x, y)$, define a set $\mathcal{J}$ of jobs to be (k, x, y)-*feasible* if

(f1) $\mathcal{J} \subseteq \{1, 2, \ldots, k\}$,

(f2) $r_j \in [x, y)$ for all $j \in \mathcal{J}$, and

(f3) $\mathcal{J}$ has a full schedule in $[x, y)$ (that is, all jobs are completed by time y.)

An earliest-deadline schedule of a (k, x, y)-feasible set of jobs will be called a (k, x, y)-*schedule*. If ties are broken consistently, then there is a 1-1 correspondence between feasible sets of jobs and their earliest-deadline schedules. Thus, for the sake of simplicity, we will use the same notation $\mathcal{J}$ for a feasible set of jobs and for its earliest-deadline schedule.

Note that if e is the job with the earliest release time, then an optimal (n, r_e, r_n)-schedule is also an optimal schedule to the whole instance. The idea of the algorithm is to compute optimal (k, r_i, r_j)-schedules $\mathcal{F}^k_{i,j}$ in bottom-up order, using dynamic programming. As there does not seem to be an efficient way to express $\mathcal{F}^k_{i,j}$ in terms of such sets for smaller instances, we use two auxiliary optimal schedules denoted $\mathcal{G}^k_{i,a}$ and $\mathcal{H}^k_{i,j}$ on which we impose some additional restrictions.

We first define the values $F^k_{i,j}$, $G^k_{i,j}$, and $H^k_{i,a}$ that are meant to represent the weights of the corresponding schedules mentioned above. The interpretation of these values is as follows:

$F^k_{i,j}$ = the optimal weight of a (k, r_i, r_j)-schedule.

$G^k_{i,a}$ = the optimal weight of a $(k, r_i, r_i + ap)$-schedule that consists of a single block starting at time r_i and ending at $r_i + ap$.

$H^k_{i,j}$ = the optimal weight of a (k, r_i, r_j)-schedule that has no gap between r_i and r_{k+1}.

In $F^k_{i,j}$ and $H^k_{i,j}$ we assume that $r_i \leq r_j$. In $H^k_{i,j}$ we additionally assume that $k < n$ and $r_i \leq r_{k+1}$.

We now give recursive formulas these values. In these formulas we use the following auxiliary functions:

$$\begin{aligned} \Delta(x, y) &= \min\left\{n, \left\lceil \frac{y - x}{p} \right\rceil - 1\right\} \\ \lambda(x) &= \operatorname{argmin}_i\{r_i : r_i \geq x\} \\ \lambda^+(x) &= \operatorname{argmin}_i\{r_i : r_i > x\} \end{aligned}$$

Thus $\Delta(x, y)$ is the maximum number of jobs (but not more than n) that can be executed between x and y (ignoring release times and deadlines), such that the interval $[x, y)$ is not completely filled. For $x \leq r_n$, $\lambda(x)$ denotes the first job released at or after x. Similarly, for $x < r_n$, $\lambda^+(x)$ is the first job released strictly after x. (Ties can be broken arbitrarily).

Values $F^k_{i,j}$. If $r_j = r_i$ then $F^k_{i,j} = 0$. Otherwise, $F^k_{i,j}$ is defined inductively as follows:

$$F^k_{i,j} \quad = \quad \max \begin{cases} F^k_{\lambda^+(r_i),j} & (F1) \\ \\ \max\limits_{\substack{1 \le a \le n \\ r_i + ap \le r_j}} \left\{ G^k_{i,a} + F^k_{\lambda(r_i+ap),j} \right\} & (F2) \end{cases}$$

Note that in (F1) $\lambda^+(r_i)$ is well defined since $r_i < r_j$, and in (F2) $\lambda(r_i + ap)$ is well defined since $r_i + ap \le r_j$.

Values $G^k_{i,a}$. If $k = 0$ or $a = 0$, then $G^k_{i,a} = 0$. If $r_k \notin [r_i, r_i + (a-1)p]$ or $d_k < r_i + ap$ then $G^k_{i,a} = G^{k-1}_{i,a}$. Otherwise, $G^k_{i,a}$ is defined as follows:

$$G^k_{i,a} \quad = \quad \max \begin{cases} G^{k-1}_{i,a} & (G1) \\ \\ G^{k-1}_{i,a-1} + w_k & (G2) \\ \\ \max\limits_{r_k < r_l < r_i + ap} \left\{ H^{k-1}_{i,l} + G^{k-1}_{l,\Delta(r_l, r_i+ap)} + w_k \right\} & (G3) \end{cases}$$

Values $H^k_{i,j}$. If $r_j = r_i$ then $H^k_{i,j} = 0$. If $k = n$ or $r_{k+1} \notin [r_i, r_j]$ then $H^k_{i,j}$ is undefined. For other values $H^k_{i,j}$ is defined inductively as follows:

$$H^k_{i,j} \quad = \quad \max_{\substack{0 \le a \le n \\ r_{k+1} \le r_i + ap \le r_j}} \left\{ G^k_{i,a} + F^k_{\lambda(r_i+ap),j} \right\} \qquad (H)$$

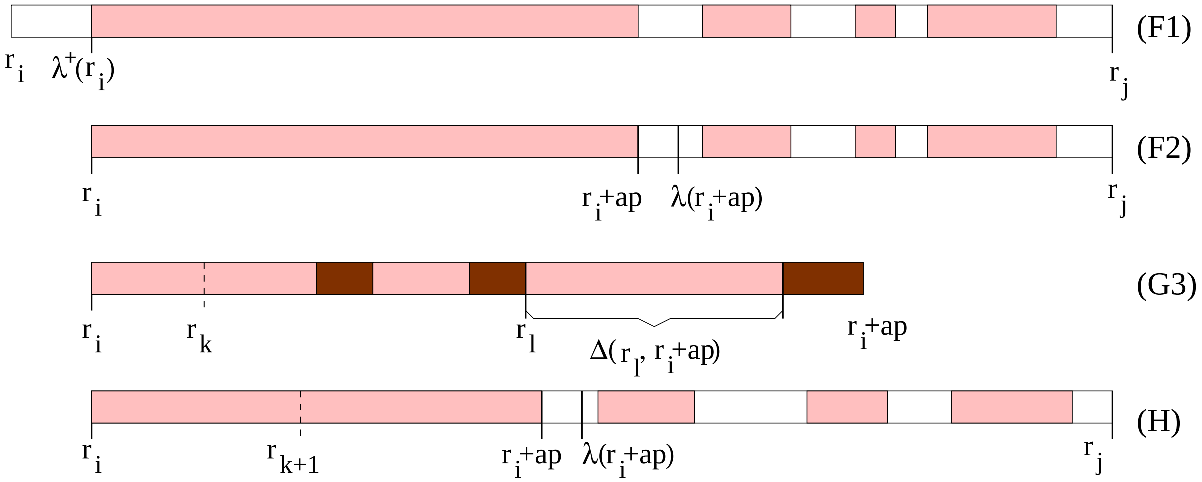

Figure 3: Graphical explanation of the recursive formulas for $F^k_{i,j}$, $G^k_{i,a}$ and $H^k_{i,j}$. Shaded regions show blocks. In (G3), the whole schedule is one block, and darker shade shows where k is executed.

Algorithm DP. The algorithm first computes the values $F^k_{i,j}, G^k_{i,a}, H^k_{i,j}$ bottom-up. The general structure of this first stage is as follows:

```
for k ← 0 to n do
    for i ← n downto 1 do
        for a ← 0 to n do
            compute G^k_{i,a}
        for j ← i to n do
            compute F^k_{i,j} and H^k_{i,j}
```

The values $F^k_{i,j}$, $G^k_{i,a}$, and $H^k_{i,j}$ are computed according to their recursive definitions, as given earlier. At each step, we record which value realized the maximum.

In the second stage, we construct an optimal schedule $\mathcal{F}^n_{e,n}$, where e is the job with earliest deadline. This is achieved by starting with $\mathcal{F}^n_{e,n}$ and recursively reconstructing optimal schedules $\mathcal{F}^k_{i,j}$, $\mathcal{G}^k_{i,a}$ and $\mathcal{H}^k_{i,j}$ that realize weights $F^k_{i,j}, G^k_{i,a}$, and $H^k_{i,j}$, respectively, according to the following procedure.

Computing $\mathcal{F}^k_{i,j}$. If $F^k_{i,j} = 0$, return $\mathcal{F}^k_{i,j} = \emptyset$. If $F^k_{i,j}$ was maximized by choice (F1), let $\mathcal{F}^k_{i,j} = \mathcal{F}^{k-1}_{\lambda^+(r_i),j}$. If $F^k_{i,j}$ was maximized by choice (F2), let $\mathcal{F}^k_{i,j} = \mathcal{G}^k_{i,a} \cup \mathcal{F}^k_{\lambda(r_i+ap),j}$, where a is the integer that realizes the maximum.

Computing $G^k_{i,a}$. If $G^k_{i,a} = 0$, return $\mathcal{G}^k_{i,a} = \emptyset$. If $G^k_{i,a}$ is realized by choice (G1), let $\mathcal{G}^k_{i,a} = \mathcal{G}^{k-1}_{i,a}$. If $G^k_{i,a}$ is realized by choice (G2), let $\mathcal{G}^k_{i,a} = \mathcal{G}^{k-1}_{i,a-1} \cup \{k\}$. If $G^k_{i,a}$ is realized by choice (G3), let $\mathcal{G}^k_{i,a} = \mathcal{H}^{k-1}_{i,l} \cup \mathcal{G}^{k-1}_{l,\Delta(r_l,r_i+ap)} \cup \{k\}$, where l is the job that realizes the maximum in (G3).

Computing $\mathcal{H}^k_{i,j}$. If $H^k_{i,j} = 0$, return $\mathcal{H}^k_{i,j} = \emptyset$. Otherwise, $\mathcal{H}^k_{i,j} = \mathcal{G}^k_{i,a} \cup \mathcal{F}^k_{\lambda(r_i+ap),j}$, where a is the integer that realizes the maximum.

Theorem 2.1 *Algorithm* DP *correctly computes a maximum-weight feasible set of jobs and it runs in time* $O(n^4)$.

Proof: The time complexity is quite obvious: We have $O(n^3)$ values $F^k_{i,j}$, $G^k_{i,a}$, $H^k_{i,j}$, and they can be stored in 3-dimensional tables. The functions $\Delta(\cdot,\cdot)$, $\lambda^+(\cdot)$, and $\lambda(\cdot)$ can be precomputed. Then each entry in these tables can be computed in time $O(n)$. The reconstruction of the schedules in the second part takes only time $O(n)$.

To show correctness, we need to prove two claims:

Claim 1:

(1f) $w(\mathcal{F}^k_{i,j}) = F^k_{i,j}$ and $\mathcal{F}^k_{i,j}$ is a (k, r_i, r_j)-schedule.

(1g) $w(\mathcal{G}^k_{i,a}) = G^k_{i,a}$ and $\mathcal{G}^k_{i,a}$ is $(k, r_i, r_i + ap)$-schedule that consists of a single block of a jobs starting at r_i.

(1h) $w(\mathcal{H}^k_{i,j}) = H^k_{i,j}$ and $\mathcal{H}^k_{i,j}$ is (k, r_i, r_j)-schedule that has no gap before r_{k+1} (assuming that $k < n$ and $r_i \le r_{k+1} \le r_j$.)

Claim 2:

(2f) If $\mathcal{J}$ is a (k, r_i, r_j)-schedule then $w(\mathcal{J}) \le F^k_{i,j}$.

(2g) If $\mathcal{J}$ is a $(k, r_i, r_i + ap)$-schedule that is a single block of a jobs then $w(\mathcal{J}) \le G^k_{i,a}$.

(2h) If $\mathcal{J}$ is a (k, r_i, r_j)-schedule that has no gap before r_{k+1} then $w(\mathcal{J}) \leq H^k_{i,j}$ (assuming that $k < n$ and $r_i \leq r_{k+1} \leq r_j$.)

We prove both claims by induction. We first define a partial order on all function instances $F^k_{i,j}$, $G^k_{i,a}$ and $H^k_{i,j}$. We first order them in order of increasing k. For a fixed k, we order them in order of increasing length of their time intervals, that is, $r_j - r_i$ for $F^k_{i,j}$ and $H^k_{i,j}$, and ap for $G^k_{i,a}$. Finally, for a fixed k, i and j, we assume that $F^k_{i,j}$ is before $H^k_{i,j}$. The induction will proceed with respect to this ordering.

We now prove Claim 1. The base cases are when $k = 0$ or $a = 0$ in $G^k_{i,a}$, or $r_i = r_j$ in $F^k_{i,j}$ and $H^k_{i,j}$. In all these cases Claim 1 holds trivially. We now examine the inductive steps.

To prove (1f), if $\mathcal{F}^k_{i,j}$ was constructed from case (F1), the claim holds by induction. If $\mathcal{F}^k_{i,j}$ was constructed from case (F2), let a be the integer that realizes the maximum and $l = \lambda(r_i + ap)$. Since $r_i + ap \leq r_l$, sets $\mathcal{G}^k_{i,a}$ and $\mathcal{F}^k_{l,j}$ are disjoint, and so are the intervals $[r_i, r_i + ap)$, $[r_l, r_j)$. Thus both $\mathcal{G}^k_{i,a}$ and $\mathcal{F}^k_{l,j}$ can be fully scheduled in $[r_i, r_j)$ and $w(\mathcal{F}^k_{i,j}) = w(\mathcal{G}^k_{i,a}) + w(\mathcal{F}^k_{l,j}) = G^k_{i,a} + F^k_{l,j} = F^k_{i,j}$, by induction.

To prove (1g), if $G^k_{i,a}$ is realized by case (G1), the claim is obvious. In case (G2), we have $k \notin \mathcal{G}^{k-1}_{i,a-1}$, $r_k \leq r_i + (a-1)p$, and $d_k \geq r_i + ap$. Thus we can schedule $\mathcal{G}^{k-1}_{i,a-1}$, and then schedule k at $r_i + (a-1)p$. By induction, $w(\mathcal{G}^k_{i,a}) = w(\mathcal{G}^{k-1}_{i,a-1}) + w_k = G^{k-1}_{i,a-1} + w_k = G^k_{i,a}$. In case (G3), let l be the job that realizes the maximum and $b = \Delta(r_l, r_i + ap)$. The sets $\mathcal{H}^{k-1}_{i,l}$ and $\mathcal{G}^{k-1}_{l,b}$ are disjoint and so are the intervals $[r_i, r_l)$, $[r_l, r_l + bp)$. By the definition of b, we have $r_l + bp < r_i + ap$, so there is a non-zero idle time in $\mathcal{H}^{k-1}_{i,l} \cup \mathcal{G}^{k-1}_{l,b}$ between $r_l + bp$ and $r_i + ap$. Since the total interval $[r_i, r_i + ap)$ has length ap, the total idle time in this interval must be at least p. Moreover, all gaps occur after r_k. This implies that we can schedule job k in these idle intervals. Also, note that $w(\mathcal{G}^k_{i,a}) = w(\mathcal{H}^{k-1}_{i,l}) + w(\mathcal{G}^{k-1}_{l,b}) + w_k = H^{k-1}_{i,l} + G^{k-1}_{l,b} + w_k = G^k_{i,a}$, so the claim holds.

To prove (1h), let a be the integer that realizes the maximum in (H) and $l = \lambda(r_i + ap)$. As before, since $r_i + ap \leq r_l$, sets $\mathcal{G}^k_{i,a}$ and $\mathcal{F}^k_{l,j}$ are disjoint, and so are the intervals $[r_i, r_i + ap)$, $[r_l, r_j)$. Thus both $\mathcal{G}^k_{i,a}$ and $\mathcal{F}^k_{l,j}$ can be fully scheduled in $[r_i, r_j)$ and $w(\mathcal{H}^k_{i,j}) = w(\mathcal{G}^k_{i,a}) + w(\mathcal{F}^k_{l,j}) = G^k_{i,a} + F^k_{l,j} = H^k_{i,j}$, by induction.

We now show Claim 2. Again, we proceed by induction with respect to the ordering of the instances described before the proof of Claim 1. The claim holds trivially for the base cases. We now consider the inductive step.

To prove (2f), we have two cases. If $\mathcal{J}$ does not start at r_i, then it cannot start earlier than at r_m, for $m = \lambda^+(r_i)$, so the claim follows by induction. If $\mathcal{J}$ starts at r_i, let a be the length of its first block. The second block (if any) cannot start earlier than at r_l, for $l = \lambda(r_i + ap)$. (Note that there might be no gap between the blocks.) We partition $\mathcal{J}$ into two sets: $\mathcal{J}_1$ containing the jobs scheduled in $[r_i, r_i + ap)$ as a single block, and $\mathcal{J}_2$ containing the jobs scheduled in $[r_l, r_j)$. By induction, $w(\mathcal{J}) = w(\mathcal{J}_1) + w(\mathcal{J}_2) \leq G^k_{i,a} + F^k_{l,j} \leq F^k_{i,j}$.

We now prove (2g). If $k \notin \mathcal{J}$ then $\mathcal{J}$ is a $(k-1, r_i, r_i + ap)$-schedule, so $w(\mathcal{J}) \leq G^{k-1}_{i,a} \leq G^k_{i,a}$, by induction and by case (G1). Now assume that $k \in \mathcal{J}$. If job k has not been interrupted, then $\mathcal{J} - \{k\}$ is a $(k-1, r_i, r_i + (a-1)p)$-schedule. Thus, by induction and (G2), $w(\mathcal{J}) \leq G^{k-1}_{i,a-1} + w_k \leq G^k_{i,a}$.

Otherwise, let l be the last job that interrupted k. Starting at r_l, $\mathcal{J}$ executes $b = \Delta(r_l, r_i + ap)$ jobs with deadlines smaller than d_k, after which it executes a portion $r_i + ap - r_l - bp > 0$ of job k. We partition $\mathcal{J} - \{k\}$ into two sets: $\mathcal{J}_1$ containing the jobs scheduled before r_l, and $\mathcal{J}_2$ containing the jobs scheduled after r_l. Note that $\mathcal{J}_1 \cup \mathcal{J}_2 = \mathcal{J} - \{k\}$, since the jobs scheduled before r_l must also be completed before r_l and the other jobs cannot be released yet. By induction, sets $\mathcal{J}_1$ is a

$(k-1, r_i, r_l)$-schedule in which the first block starts at r_i and ends after r_k, and $\mathcal{J}_2$ is a single block starting at r_l and ending at $r_l + bp$. Thus, by induction and (G3), $w(\mathcal{J}) = w(\mathcal{J}_1) + w(\mathcal{J}_2) + w_k \leq H_{i,l}^{k-1} + G_{l,b}^{k-1} + w_k \leq G_{i,a}^{k}$.

The proof of (2h) is similar. We have two subcases. Suppose first that $r_{k+1} = r_i$. By induction, we have $w(\mathcal{J}) \leq F_{i,j}^{k} = H_{i,j}^{k}$, since in this case we can choose $a = 0$ in (H). If $r_{k+1} > r_i$, then the first block of $\mathcal{J}$ starts at r_i and ends after r_{k+1}. Let a be the length of its first block. The second block (if any) cannot start earlier than at r_l, for $l = \lambda(r_i + ap)$. We partition $\mathcal{J}$ into two sets $\mathcal{J}_1$ containing the jobs scheduled in $[r_i, r_i + ap)$ as a single block and $\mathcal{J}_2$ containing the jobs scheduled in $[r_l, r_j)$. By induction, $w(\mathcal{J}) = w(\mathcal{J}_1) + w(\mathcal{J}_2) \leq G_{i,a}^{k} + F_{l,j}^{k} \leq H_{i,j}^{k}$. □

3 Final Remarks

Several open problems remain. Although our running time $O(n^4)$ for the scheduling problem $(1|r_j; p_j{=}p; \mathrm{pmtn}|\sum w_j U_j)$ is substantially better than the previous bound of $O(n^{10})$, it would be interesting to see whether it can be improved further. Similarly, it would be interesting to improve the running time for the non-preemptive version of this problem, $(1|r_j; p_j{=}p|\sum w_j U_j)$, which is currently $O(n^7)$ [Bap99a].

In the multi-processor case, the weighted version is known to be NP-complete [BK99], but the non-weighted version remains open. More specifically, it is not known whether the problem $(P|r_j; p_j{=}p; \mathrm{pmtn}|\sum U_j)$ can be solved in polynomial time. (One difficulty that arises for 2 or more processors is that we cannot restrict ourselves to earliest-deadline schedules. For example, an instance consisting of three jobs with feasible intervals $(0,3)$, $(0,4)$, and $(0,5)$ and processing time $p = 3$ is feasible, but the earliest-deadline schedule will complete only jobs 1 and 2.) In the multi-processor case, one can also consider a preemptive version where jobs are not allowed to migrate between processors.

References

[Bap99a] Philippe Baptiste. An $O(n^4)$ algorithm for preemptive scheduling of a singler machine to minimize the number of late jobs. *Operations Research Letters*, 24:175–180, 1999.

[Bap99b] Philippe Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal on Scheduling*, 2:245–252, 1999.

[Bap00] Philippe Baptiste. Preemptive scheduling of identical machines. Technical report, Universite de Technologie de Compiegne, France., 2000.

[BBKT02] Philippe Baptiste, Peter Brucker, Sigrid Knust, and Vadim G. Timkovsky. Fourteen notes on equal-processing-time scheduling. Submitted for publication, 2002.

[BK] Peter Brucker and Sigrid Knust. Complexity results for scheduling problems. www.mathematik.uni-osnabrueck.de/research/OR/class.

[BK99] Peter Brucker and Svetlana A. Kravchenko. Preemption can make parallel machine scheduling problems hard. *Osnabrücker Schriften zur Mathematik*, 211, 1999.

[Car81] Jacques Carlier. Problèmes d'ordonnancement à durées égales. *QUESTIO*, 5(4):219–228, 1981.

[DLW92] Jianzhong Du, Joseph Y.T. Leung, and Chin S. Wong. Minimizing the number of late jobs with release time constraint. *Journal on Combinatorial Mathematics and Combinatorial Computing*, 11:97–107, 1992.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-completeness.* Freeman, 1979.

[KS95] G. Koren and D. Shasha. d^{over}: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24:318–339, 1995.

[Law90] Eugene L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Ann. Oper. Res.*, 26:125–133, 1990.

[Law94] Eugene L. Lawler. Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the 'tower of sets' property. *Mathl. Comput. Modelling*, 20(2):91–106, 1994.