INTERNATIONAL JOURNAL OF COMPUTERS
Issue 3, Volume 4, 2010

Fast algorithms for preemptive scheduling of jobs with release times on a single processor to minimize the number of late jobs

Nodari Vakhania*

Abstract— We have n jobs with release times and due-dates to be scheduled preemptively on a single-machine that can handle at most one job at a time. Our objective is to minimize the number of late jobs, ones completed after their due-dates. This problem is known to be solvable in time $O(n^3 \log n)$. Here we present two polynomial-time algorithms with a superior running time. The first algorithm solves optimally in time $O(n^2)$ the special case of the problem when job processing times and due-dates are tied so that for each pair of jobs i, j with $d_i > d_j$, $p_i \geq p_j$. This particular setting has real-life applications. The second algorithm runs in time $O(n \log n)$ and works for the general version of the problem. As we show, there are strong cases when the algorithm finds an optimal solution.

Keywords–scheduling single processor, preemption, release date, due date, late job, algorithm

1 Introduction

Scheduling problems arise in various practical circumstances. Examples of such problems are job shop problems in industry, scheduling of information and computational processes, traffic scheduling and servicing of cargo trains, ships, airplanes. There are scheduling problems of diverse types and different complexities. Saying generally, one deals with two initial notions: *job* (or *task*) and *machine* (or *processor*). A job is a part of the whole work to be done, a machine is the means for the performance of a job. Each job j has its *processing time* p_j, i.e. it needs this prescribed time on a machine, and a machine cannot handle more than one job at a time. The *due date* d_j is the desirable completion time for job j, and the *release time* r_j is the time moment when job j becomes available (it cannot be processed before).

Consider the following basic scheduling problem. Jobs from $J = \{1, 2, \ldots, n\}$ have to be assigned to or scheduled on a single machine when each $j \in J$ becomes available at an integer release time r_j and has an integer due date d_j. Job j needs an integer p_j on the machine. A *schedule* assigns each job j time intervals with the total length of p_j starting no earlier than at time r_j so that there is no intersection between intervals of different jobs, i.e., the machine may handle at most one job at a time. In this way we allow job *preemptions* splitting jobs into parts. A job is *late* (*on time*, respectively) if it (its latest scheduled part) is completed after (at or before, respectively) its due date. Our objective is to minimize the number of late jobs. Due to this objective function, we may assume that every job may potentially be completed by its due date, i.e., $r_j + p \leq d_j$, for each j (then we say that job release times and due dates are *agreeable*).

The general problem described above is commonly abbreviated as $1/pmtn, r_j/\sum U_j$ (U_j is a 0-1 function taking value 1 iff job j is late). It is known to be solvable in polynomial time. In particular, Lawler [5] and Baptiste [1] have suggested dynamic programming algorithms with the time complexity of $O(n^5)$ and $O(n^4)$, respectively. Recently Vakhania [11] has improved the time complexity to $O(n^3 \log n)$. The non-preemptive version of the above problem $1/r_j/\sum U_j$ is known to be strongly NP-hard. The version with equal-length jobs $1/pmtn, p_j = p, r_j/\sum U_j$ can be solved on-line in time $O(n \log n)$ Vakhania [9] and [10]. See also Lawler [6] for an off-line dynamic programming algorithm with the same time complexity. This algorithm solves a more general problem with job weights, a special case of $1/pmtn, r_j/\sum w_j U_j$ in which the jobs can be ordered so that $r_1 \leq r_2 \leq ... \leq r_n, p_1 \leq p_2 \leq \ldots \leq p_n$ and $w_1 \geq w_2 \geq \ldots \geq w_n$. Both preemptive and non-preemptive versions can be solved relatively easily in time $O(n \log n)$ if either the release times or due-dates of all jobs are equal (Moore [8]).

For multiprocessor case, if jobs have release times and arbitrary processing times, already scheduling on 2 identical processors with preemptions $P2/pmtn, r_j/\sum U_j$ is NP-hard Du et al. [4]. The weighted preemptive version even without release times $P/pmtn, p_j = p/\sum w_j U_j$ is also NP-hard, and its non-preemptive version $P/p_j = p/\sum w_j U_j$ is surprisingly polynomial Brucker & Kravchenko [3]. Without weights for the problem $P/p_j = p, r_j/\sum U_j$ and its preemptive version, there exist dynamic programming algorithms in whose time complexity expression m appears in the power of a polynomial of n (see, for example, Lawler [7], Baptiste et. al [2]).

In this paper we deal first with a special case of the single-machine version $1/pmtn, r_j/\sum U_j$ when job processing times and due-dates are tied in the following way. For each pair of jobs i, j with $d_i > d_j$, $p_i \geq p_j$; for $d_i = d_j$, p_i and p_j have no further restriction. Our model is motivated by some practical applications when the man-

*State University of Morelos, Mexico. Inst. of Computational Math., Tbilisi, Georgia. E-mail: nodari@uaem.mx. Partially supported by CONACyT grant 48433

ufacturer prefers to finish shorter jobs ahead longer jobs in order to provide the customer with the maximal amount of the completed jobs ASAP. Then the manufacturer sets the due-dates so that shorter jobs have smaller due-dates. The algorithm we suggest has a superb time complexity of $O(n^2)$ compared to the earlier mentioned algorithms for the general single-machine version, which makes it more appropriate for the above type of applications.

We also present a fast $O(n \log n)$ time algorithm for the general version of the problem. We study the general properties of the schedules generated by that algorithm and formulate three different cases when the generated solution is optimal.

In the next section we describe our $O(n^2)$ algorithm for the case with tied parameters. First we give its description, then we prove its soundness and time complexity, and finally we show why it does not work in the general case. The following sections are devoted to the general version of our problem. In section 3 we describe some useful structural properties of the problem. In Section 4 we present our $O(n \log n)$ algorithm and formulate the first two cases when it generates an optimal schedule. In Section 4 we introduce some additional concepts and formulate our third stronger case when the algorithm still constructs an optimal solution.

2 The algorithm for the version of the problem with tied parameters

In this section we present the algorithm and its correctness proof for the version of our problem with tied parameters when for each couple of jobs i and j, $d_i > d_j$ yields $p_i \geq p_j$. First, we give the following general observation. Due to the nature of our objective function, if a job is late then it can be scheduled arbitrarily late without affecting our objective function. Suppose S is a feasible schedule with all its jobs being included on-time, and we can assert that we have included the maximal possible number of jobs in it. Then we can append all the omitted jobs in an arbitrary feasible fashion at the end of S, in linear time. Because of this, we shall take care only on on time scheduling of jobs. Thus we shall exclusively deal with the schedules in which all the jobs are scheduled on time; among all such schedules, we shall look for one containing the maximal possible number of jobs, i.e., an optimal one.

2.1 The description

First we give a brief description of our algorithm. Our first step is to renumber jobs in J in a non-decreasing order of their due-dates. After this preprocessing with the cost of $O(n \log n)$ the jobs in J are ordered so that $d_1 \leq d_2 \leq \ldots \leq d_n$. According to our assumption, this yields $p_1 \leq p_2 \leq \ldots \leq p_n$. Iteratively, we shall process the jobs in this order trying to schedule each next *incoming job* i, $i = 1, 2, \ldots, n$, at its release time. If this conflict with some already scheduled job(s) occurs then i might be split and scheduled within the available idle time intervals in case it can be completed by time d_i. Otherwise i is omitted. We give some more details below.

We will have n basic iterations, $i = 1, 2, \ldots, n$, so that we try to schedule job i at iteration i. We use a doubly linked list L to keep the track of the already occupied time intervals on the machine. Each element of this list keeps two parameters which are left and right limits of the corresponding interval. The elements will be organized so that the right limit of the interval represented by an element of the list is strictly less than the left time interval represented by the next element(s) of the list.

Now we describe procedure $SCAN(i)$ which is the main body of our algorithm used at each iteration i. $SCAN(i)$ either includes job i updating the current list L or establishes that i cannot be included, in which case i is omitted. Initially on iteration 1 $SCAN(1)$ includes job 1 within the interval $[r_1, r_1 + p_1]$ and adds our first element to L representing the time (execution) interval of job 1.

Suppose for $i \geq 2$ the time interval $[r_i, r_i + p_i]$ is idle by iteration i. Then job i is scheduled within that interval on iteration i. Either a new element is added to L or one or two elements from L might be converted into one element with the modified limits representing the time intervals of at least two jobs including job i (so the corresponding time intervals are merged). A new element will be added to L if either $r_i + p_i$ is strictly less than the left-limit of the first element of L (in this case the new element representing time interval of i will be inserted as the first one in L) or r_i is strictly more than the right limit of the last element of L (in this case the new element will be inserted as the last one in L). Otherwise, if there are two successive elements in L such that r_i is strictly more than the left limit of the first of these elements, and $r_i + p_i$ is strictly less than the left-limit of the second element, then the new element representing time interval of i will be inserted in between these two elements. Otherwise, if the equality is reached in one or both of the above strict inequalities then the corresponding (two or three) time intervals are merged. The resultant time interval will be represented by a single new element of L substituting the above one or two elements. The left and right limits of the new interval are determined in the obvious way.

If none of the above cases occur then the time interval $[r_i, r_i + p_i]$ is not idle by iteration i, i.e., it intersects with some time interval(s) already represented in L in more than one point. In this case job i might be included or may not. $SCAN(i)$ scans all occupied time intervals from time moment r_i searching for the earliest idle time moment t such that $t + p_i \leq d_i$. If there exists no such t then job i is omitted; otherwise, if i can be completely scheduled

from the moment t (no intersection with any other occupied interval occurs) then it is scheduled within the interval $[t, t + p_i]$. Otherwise, the above verification is repeated; i.e., the next earliest idle time moment t is looked for such that $t + p_i^* \leq d_i$, where p_i^* is the length of the yet unscheduled portion of job i. Similar steps are carried out till either job i is feasibly included or it cannot be feasibly included as above and hence it is omitted. In case of success (i is feasibly included) all the corresponding time intervals are merged in the straightforward way into a single time interval now also representing the execution interval of job i, the corresponding elements of L being replaced by a new single element with the above determined left and right limits.

This completes the description of the procedure $SCAN(i)$. The overall algorithm is now as follows:

ALGORITHM MAIN

Step 1.
Reorder jobs in J so that $d_1 \leq d_2 \leq \ldots \leq d_n$

Step 2.
FOR $i = 1$ to n DO $SCAN(i)$.

2.2 The correctness and time complexity

Theorem 1 *The algorithm MAIN produces an optimal schedule in time $O(n^2)$.*

Proof. As to the time complexity, Step 1 which is sorting, takes time $O(n \log n)$. We have n iterations on Step 2, on each of which $SCAN(i)$ is called. It remains to see the time complexity of this procedure. If the next incoming job i can be included without intersecting with any of the intervals in L then it is included in a constant time and the update of L with the new execution interval (that of job i) will take time $O(n)$ as clearly, there are no more than n intervals in L. In general, while inserting the next incoming job i, we may need to skip at most $n - 1$ time intervals from L. All the skipped time intervals are to be merged in a single time interval (unified by the portions of the newly included job i). Hence we will spend time $O(n)$ for the inclusion of each i and update of L on each iteration. Clearly, we will spend the same amount of time whenever i cannot be included. Hence the overall time complexity is $O(n \log n) + O(n)O(n) = O(n^2)$.

We now switch to the soundness part. According to our renumbering of jobs on Step 1, the shortest jobs will be included ahead longer ones on Step 2. The algorithm is clearly optimal if $SCAN(i)$ has succeeded to include every next incoming job i. Otherwise, let i be the first incoming job that could not have been included. Then due to our assumption that job release times and due-dates are agreeable, the feasible interval (r_i, d_i) of i must intersect with some interval(s) from L representing the execution intervals of one or more already assigned jobs. Let us denote this set of jobs by $I(i)$. From our construction, the due-date of every $j \in I(i)$ is no more than d_i. Hence, j cannot be moved (feasibly) out of the feasible interval of i, and it follows that either job i or one of the j-s is to be omitted in any feasible schedule. Suppose we omit any subset of jobs from $I(i)$. Then we claim that we may include no more than $|I(i)|$ (yet unscheduled) jobs instead. Indeed, if we remove some $j \in I(i)$ including instead some yet unscheduled job l, then l will again be scheduled within the feasible interval of i, whereas it will take no less space than was taken by job j, as $p_l \geq p_j$. Jobs j and l might be replaced by a corresponding job sets and a similar reasoning can be applied to these job sets. It follows that there is no benefit in moving or omitting one of the already assigned jobs. And since the corresponding set together with job i is not feasible, job i is to be omitted. We reiterate the whole reasoning for every next incoming job that cannot be feasibly included completing in this way the proof. □

2.3 Why Algorithm MAIN is not optimal for the general version

It is not difficult to see that our algorithm not necessarily will find an optimal solution if applied for the instances with at least three jobs i, j and k with the following property: $d_i < d_j$ and $d_i < d_k$ whereas $p_i > p_j$ and $p_i > p_k$. It is clear that the algorithm will include job i ahead jobs j and k, whereas it may happen that none of the latter two jobs might be further included. Indeed, consider a problem instance with just these three jobs defined as follows: $r_i = 1$, $r_j = 0$, $r_k = 2$, $p_i = 10$, $p_j = p_k = 4$, $d_i = 11$, $d_k = d_j = 12$. Thus jobs are ordered so that $i = 1$, $j = 2$ and $k = 3$. On the first iteration job i is included into the interval $[1, 11)$. On the second iteration, the first portion of job j is inserted into the interval $[0, 1)$; however, its second portion of the length 3 does not fit into the interval $[11, 12)$. So job k cannot be completed by its due-date 12 and hence is omitted. On the third iteration, job k is similarly omitted. There are two late jobs j and k in the resultant schedule, whereas there exists a feasible schedule with just one late job i in which jobs j and i are scheduled in the intervals $[0, 4)$ and $[4, 8)$, respectively.

In the next section we present a better algorithm for the general setting.

3 Some basic properties of the general version of the problem

In this section we give some useful structural properties of the general version of our problem $1/pmtn, r_j/\sum U_j$. Note that our earlier observation for the version with tied parameters also holds: any late job can arbitrarily be appended to a schedule that contains maximal possible number of on-time scheduled jobs. In this way we easily obtain an optimal schedule from the former schedule. Again, there will be no late job in any schedule we shall be dealing with.

We denote by $|S|$ the number of jobs (completed on time) in a schedule S, and by $||S||$ the completion time of the latest completed job (the makespan) in S. We will use S for both, a schedule and the corresponding set of jobs.

The (preemptive) ED-heuristic is a common algorithm for obtaining a feasible schedule rapidly in $O(n \log n)$ time: Iteratively, among all released jobs by the current scheduling time t, ED-heuristic schedules a job with the smallest due-date ties being broken by selecting a shortest job (further ties are broken arbitrarily). If during the execution of the current job j another job i with $d_i < d_j$, or with $d_i = d_j$ but with a smaller early completion time is released (i.e., job i if immediately scheduled will be completed earlier than job j), j is interrupted and i is scheduled. We call such a job selected by ED-heuristic at the current scheduling time t the *incoming job* of time t and denote it by $i(t)$.

Initially, $t = \min\{r_j | j \in J\}$. If no incoming job occurs while the currently scheduled job is running then (the current portion of) the latter job completes without any (further) interruption. In general, a scheduling time t is either the release time of the next incoming job or/and the completion time of the latest scheduled job so far. In the latter case either an earlier released job becomes the new incoming job or the next portion of an earlier interrupted job resumes or the new incoming job is released just at the moment when the latest scheduled job completes. Observe that t is the release time of the next incoming job whenever it is preceded by *gap* (an idle time interval on the machine). We denote by t^+ (t^-, respectively) the next (the previous, respectively) to t scheduling time, and by S^t the schedule constructed to time t. In general, we write $t \succ t'$ if t is a scheduling time occurred after scheduling time t' in our algorithm. As we will see later $t^+ < t$ is possible, in general, we may have $t < t'$ for $t \succ t'$: this may happen as some already scheduled job can be removed from S^t and a shorter job scheduled instead.

At any scheduling time t the set of the already scheduled jobs can be divided into the ones which are completed by time t and the ones which were interrupted and are not completed by time t. We denote the latter set of jobs by U_t, and denote by u^+ the *residue* of $u \in U_t$; *that is*, the part of u yet unprocessed by time t. Note that a new uncompleted job arises only when the incoming job interrupts it. For the auxiliary checking purposes we may "complete virtually" jobs in U_t after time t creating respectively the *virtual part* in S^t that starts after time t.

We stress here that the virtual part has only an auxiliary purpose: given that S^t is feasible all its jobs being completed on time, we wish to know if all the jobs from S^t together with job i can also fit on time on the machine. We shall refer to the part of S^t before time t as its *real part* (the partial schedule generated so far). From now on, when referring to an ED-schedule S we will mean its both real and virtual parts; however, the makespan $||S||$ will concern only the real part of S. Below we describe how we schedule residues without any interruption in the virtual part.

Recall that if $i(t)$ is more urgent than the currently running job j then $i(t)$ interrupts j, and j^+ is non-empty. In general, we may schedule the residues of jobs in U_t applying a dislodged ED-heuristic. Since all jobs up to time t are scheduled by ED-heuristic, each $i(t)$ has a due-date, no greater than that of any job from U_t. This implies that up to time t, the jobs from U_t are scheduled by the non-increasing order of their due dates. Each newly included job in U_t has a due date no greater than that of all earlier included ones. Hence, ED-heuristic will schedule residues of jobs from U_t in the order, reverse to the order in which the completed parts before time t were scheduled.

Conserving this order for scheduling residues after time t, we may schedule them as late as possible, without making them late, as follows. The earliest arisen uncompleted job is scheduled so that it is completed exactly at its due date. Each newly arisen uncompleted job u has a due date, no greater than that of all earlier arisen uncompleted jobs with already "scheduled" residues. Let $\tau > t$ be the starting time of the latest scheduled residue so far. If $d_u \geq \tau$ then we add u^+ so that it completes at time τ; otherwise, we add u^+ so that it completes at time d_u. As S^t was feasible, no overlapping between the residues may occur and each residue will start no earlier than at time t. However, some residue may start before time $t + p_i$ ($i = i(t)$).

It may happen of course that among all (yet uncompleted) jobs released by time t, job $u \in U_t$ has the smallest due date. Then u is the lastly arisen uncompleted job and u^+ is the earliest started residue in S^t. We schedule job u at time t and update the virtual part for S^{t^+} by removing the already scheduled part of u^+. Note that since u^+ was the earliest scheduled residue, no new gap in between the remained residues will occur. If u is no more interrupted and hence completely scheduled, we let $U_{t^+} := U_t \setminus \{u\}$. Otherwise, u is interrupted once again, u^+ is updated respectively, t^+ being the release time of the next incoming job.

Suppose job j completes at time t and $U_t = \emptyset$. Then the next incoming job (released no earlier than at time t) starts new *block*. Each S^t is naturally divided into the blocks: at scheduling time t^+ a new block is initiated if $[t, t^+)$ is a gap (we assume to have a zero-length gap at time t between two blocks if the next incoming job is released right at time t). By scheduling time t, we denote the current block by B^t. If B^t is not newly initiated then either $i(t)$ interrupts the currently running job or a job from U_t is resumed or an earlier released job that did not become an incoming job now becomes $i(t)$.

Let $S = S^t$, and let $S(-k)$, $k \in S$ ($S(+l)$, $l \notin S$, respectively) be the ED-schedule generated for the job set $S \setminus \{k\}$ ($S \cup \{l\}$, respectively) the virtual part being generated by the dislodged ED-heuristic as above. We may

generalize these notations straightforwardly substituting k and l by job sets.

We will say that the incoming job i can be *perfectly scheduled* at time t if $S^t(+i)$ is a feasible schedule in which all jobs are completed on time. We may observe that i can be perfectly scheduled only if the latest arisen residue (the first one from the virtual part) starts at or after time $t + p_i$ in $S^t(+i)$ (as otherwise the overlapping will occur). Let us call a job set *feasible* if there exists a feasible schedule in which all the jobs from this set are scheduled on time. The following observation is easily verified:

Observation 2 *If $i(t)$ cannot be perfectly scheduled then the job set $S \cup \{i\}$ is not feasible.*

It follows that if $i = i(t)$ cannot be perfectly scheduled then either i is late in $S^t(+i)$ or/and it overlaps with the virtual part. Iteratively at each scheduling time t we start job $i(t)$ at time t if it can be perfectly scheduled. The real part of the resultant schedule $S^{t^+} = S^t(+i)$ now extends up to time t^+. We need the following definitions.

We will say that the incoming job i is *disregarded* (*omitted*, respectively) if it is discarded at time t and is never again considered for the inclusion (discarded at time t but can later again be considered for the inclusion, respectively). An omitted job at time t is placed into our current *reserve list* L_t.

We say that S^t *cannot be enlarged* if there exists no feasible set with $|S^t| + 1$ jobs from $S^t \cup L_t \cup \{i(t)\}$. This means that $i(t)$ cannot be scheduled on time even by removing any subset of already scheduled jobs from S^t and including at least the same amount of jobs from L_t instead.

σ, a subset of jobs in S^t, is said to give an *admissible choice* at time t if the job set $S^t \setminus \sigma \cup \{i\}$ is feasible. When σ is a single-element set, a job giving an admissible choice will be referred to as an *admissible job*. We shall refer to the earliest scheduling time t on which $k \in S^t$ gives an admissible choice as the *check-in time* of k, denoted by $\tau(k)$.

We will say that job set $\Lambda \subset \{L^t \cup i(t)\}$ can be *freely returned* to S^t if the job set $S^t \cup \Lambda$ is feasible. To enlarge S^t, at least 2 jobs from S^t must be omitted and at most one less jobs from L_t must be freely returned, together with job $i(t)$.

4 Description of EED-Algorithm

In this section we introduce the following extension of ED-heuristic, EED-Algorithm for short: At each scheduling time t, include $i(t)$ from time t till the next scheduling time if $i(t)$ can be perfectly scheduled; otherwise, omit $i(t)$. Note that EED-Algorithm has the same time complexity of $O(n \log n)$ as ED-heuristic (as the checking whether the next incoming job can be perfectly scheduled takes a constant tame). The schedule constructed by EED-Algorithm is clearly optimal if each incoming job is perfectly scheduled. Due to Observation 3, EED-Algorithm remains optimal if a single incoming job, that could not have been perfectly scheduled, was omitted:

Observation 3 *The schedule obtained by EED-Algorithm is optimal if while its construction a single incoming job that could not have been perfectly scheduled (and hence omitted) has occurred.*

There are other less trivial cases when EED-Algorithm gives an optimal solution. We formulate one in this section, and the other case is dealt with in the following section.

Let us call a set of jobs with an admissible choice a *minimal admissible* set if no its proper subset gives an admissible choice.

Lemma 4 *Suppose $|\sigma| \geq 2$ is a minimal admissible set at time t. Then the total processing time of any proper subset of σ which does not contain the job of σ scheduled earliest in S^t, is less than p_i, $i = i(t)$.*

Proof. Let j be the job of σ scheduled latest in S^t. The left-shift in $S^t(-j)$, caused by the removal of j must be less than p_i, as otherwise j would give an admissible choice. We claim that $p_j < p_i$. Indeed, if $p_j \geq p_i$, because of the above remark, there must arise a new gap after the removal of j in $S^t(-j)$. But then the removal of any other job of σ can cause the left-shift, which contradicts the fact that σ is minimal and $|\sigma| \geq 2$. Hence for $|\sigma| = 2$ the proposition is true. For $|\sigma| > 2$ consider the next to j job $k \in \sigma$, scheduled latest in S^t. Quite analogously, if $p_j + p_k \geq p_i$, there would arise new gap(s) in $S^t(-j-k)$ before other jobs of σ, which contradicts the minimality of σ. If $|\sigma| = 3$, we are done; otherwise we continue similarly until we come to the job, neighboring the earliest scheduled job of σ in S^t. □

Theorem 5 *The incoming job $i = i(t)$ can be disregarded if there exists no job with an admissible choice in S^t.*

Proof. We have to omit at least two jobs from S^t, say k and l, to schedule i on time. In general, Suppose σ, $|\sigma| \geq 2$ is a minimal admissible set of jobs at time t and k is the job of σ scheduled earliest in S^t. It will suffice to show that any schedule S^i which contains job i (and does not contain jobs of σ) is dominated by $S^i(-i+\sigma')$, where $\sigma' = \sigma \setminus \{k\}$. We use following observations to show this claim. Any job released within the time intervals, occupied by jobs of σ' in $S^i(-i+\sigma')$ will be also released within the time interval(s), liberated by job i. Moreover, by Lemma 4, the total length of the former time intervals is no more than that of the latter time interval(s). The above two observations together with the fact that $|S^i(-i+\sigma')| \geq |S^i|$ imply that $S^i(-i+\sigma')$ dominates S^i, and the proof is complete. □

5 A deeper study of EED-Algorithm

In this section we derive a stronger case when EED-Algorithm remains optimal. If the incoming job $i(\tau)$ cannot be perfectly scheduled at time τ then we have to omit either job $i(\tau)$ or some already scheduled job(s) at time τ. A selection of this type depends on job benefits defined in this section. Let k be a job from S^τ with an admissible choice at time τ, $\tau = t(k)$ being the activation time of k. Let $t > \tau$ be a scheduling time such that the interval $[\tau, t]$ is from B^t. At time t, the set of all time intervals occupied by k in S^τ will be referred as the *(potential) liberated space* by k (because of preemptions, we may have more than one such an interval from the real or the virtual part of S^τ). We denote the above set of intervals by $\mathcal{I}_k$ (note that an interval from $\mathcal{I}_k$ can be either from the real or the virtual part of S^τ). If we omit k at time t, the liberated space by k might be used by other jobs of S^τ, job $i(t)$ and by the incoming jobs from time τ to time t released before or within an interval from $\mathcal{I}_k$.

Different jobs may liberate different amount of useful space for scheduling other jobs. The benefit or the *profit* of job k at time $t \geq \tau$ (written $profit_t(k)$) depends on the processing time of k and the release time of the incoming jobs between times τ and t which can make the use of intervals from $\mathcal{I}_k$. In general, $profit_t(k) \leq p_k$. The strict inequality will hold if not all intervals from $\mathcal{I}_k$ can be beneficially used. Intuitively, this means that there will occur gaps representing remained unfilled space (within or behind the intervals from $\mathcal{I}_k$) after the removal of k.

Assume that $i = i(t)$ cannot be perfectly scheduled at time t. We can distinguish two kinds of jobs from S^t with an admissible choice. The jobs of the first kind are more urgent than job i. Recall from the previous section that (1) either i is late, or (2) i is not late but it overlaps with the virtual part of S, or (3) i is late and it also overlaps with the virtual part of S. Consider first the case when i is late, i.e., there occurs either case (1) or case (3). There must be at least one job in S^t which is at least as urgent as i and is completed by time t (in particular, the latest scheduled job in S^t has this property). As job release times and due dates are consecutive, i must have been released before time t, hence it is delayed by some already scheduled job(s). For $j \in S^t$, the *left-shift* at time t is the difference $||S^t(+i)|| - ||S^t(-j+i)||$. Let j be the latest scheduled job (job portion) in B^t with $d_j \geq d_i$. We call any job from B^t scheduled after job j a *category I* job; if there exists no such j in B^t, all jobs from B^t are category I jobs. Clearly, the left-shift corresponding to each category I job k is at most p_k; it will be at least $\min\{p_i, p_k\}$ if k is started no earlier than at time r_i. Job i cannot be immediately scheduled within the liberated space by k if k is completed by time r_i. In this case, the jobs which were scheduled after k might be successively left-shifted, and job i might also be left-shifted. By the ED-rule, all the above jobs will be less urgent than job k.

Since each category I job is more urgent than $i(t)$, it can have no residue at time t. If we omit the real part of any other job from B^t (one, scheduled before all category I jobs), neither job i nor any category I job can be left-shifted. Therefore, a job from B^t which is not a category I job can give an admissible choice only if there exists at least one such job with a non-negative residue, whereas i overlaps with the virtual part of S^t in $S^t(+i)$ (we call this intersection the *conflict interval* at time t). In this case, either of the above cases (2) or (3) should occur. Let e be the earliest scheduled job in B^t with a non-empty residue. We call job e and all jobs scheduled between e and the first category I job the *category II* jobs. It follows that any $k \in S^t$ with a non-empty residue is a category II job. All jobs scheduled after such k in real part of S^t are at least as urgent as k and hence cannot be scheduled within the liberated space by k. Only (a part of) the residue of a job scheduled before k can be scheduled within the liberated space by k. The following observation, which develops Observation 3, is evident:

Observation 6 *Only jobs of categories I or II may give an admissible choice at time t. A job which removal causes no left-shift (any category II job) may give an admissible choice only if the conflict interval at time t is non-empty.*

If a category I job k is omitted at its activation time τ, then other category I jobs and job $i(\tau)$ might be left-shifted in $S^\tau(-k+i(\tau))$; if there remains unused by the above jobs liberated space by k, a residue of a category II job might also be scheduled within this space. A category I job or a residue of a category II job might be scheduled within the liberated space by k, whereas other category I jobs might be scheduled within the intervals released by earlier left-shifted category I jobs. In other words, the category I jobs scheduled after k in S^τ will be successively left-shifted, which may cause the rise of new gaps behind the intervals from $\mathcal{I}_k$. If $|\mathcal{I}_k| = \pi$ then job k was interrupted $\pi - 1$ times. It follows from ED-heuristic that none of the jobs from the real part of S^τ can be scheduled within the first $\pi - 1$ intervals. Hence only the latest interval from $\mathcal{I}_k$ may provide the successive left-shift of the jobs of S^τ. If k is a category II job then none of the intervals from $\mathcal{I}_k$ can yield such a left-shift. The gaps induced by the intervals from $\mathcal{I}_k$ can further be filled out by the incoming job $i(\tau)$ and successive incoming jobs.

We distinguish two types of newly arisen gaps in $S^\tau(-k+i(\tau))$: the gaps within intervals from $\mathcal{I}_k$, and the gaps behind an interval from $\mathcal{I}_k$ arisen because of the successive left-shift. As we have already noted, only the latest interval from $\mathcal{I}_k$, for a category I job k, may produce a gap of the second type. We call a gap of either of the above two types a *pending gap* of k. A pending gap of a category I job might either be from the real part of S^τ (a *real pending gap* of k) or it might also be from its virtual part (a *virtual pending gap* of k). A virtual pending gap of k will arise if

the ED-rule has "moved" the residue of a category II job to the real part in $S^\tau(-k+i(\tau))$, hence the former busy interval from the virtual part of S^τ becomes idle. Obviously, if k is a category II job (in cases (2) and (3)), it may only have a virtual gap. Such a gap will be from a former busy interval in the virtual part of S^τ, occupied either by job k or by some other category II job j. For the latter case, job j is moved to the liberated space by the real part of k.

A pending gap of k might or might not be potentially used by the future incoming jobs, depending on the release times of these jobs. If within a pending gap of k the incoming job $i(t)$, $t > \tau$, can be scheduled then we will say that this gap can be *(partially or completely) recuperated* at time t. If by time t all pending gaps of k can be completely recuperated then the profit of k will reach its maximal value p_k at time t and the set of pending gaps of k will become empty. In general, the set of pending gaps of k at time t is formed by all pending gaps of k of time τ which could not have been recuperated by time t (if a pending gap of k of time τ can be partially recuperated then it is reduced respectively). We denote the set of pending gaps of k at time t by $pg_t(k)$, and will use $|pg_t(k)|$ for the summary length of gaps from $pg_t(k)$.

Filling jobs. We call a *filling job* of k any job which can be scheduled within a gap from $pg_\tau(k)$. We denote by $F_t(k)$ the set of filling jobs of k at time t. Observe that any filling job of k is an incoming job $i(t')$, $\tau \preceq t' \preceq t$. The next proposition immediately follows from the ED-rule and from the fact that job release times and deadlines are consecutive:

Proposition 7 *If the incoming job $i(t)$ cannot be perfectly scheduled at time t then it is a filling job of at least one job in B^t.*

At time τ, we define the *initial active subinterval* of an interval $I \in \mathcal{I}_k$ as a (longest) non-idle interval in $S^\tau(-k+i(\tau))$ within I. As we have already observed, if k is a category I job and I is the latest interval from $\mathcal{I}_k$, then either some other category I job or the residue of a category II job or the incoming job of time t', $\tau \preceq t' \preceq t$, can be scheduled within I; if I is not the latest interval from $\mathcal{I}_k$ or k is a category II job and I is from the real part of S^τ, then either the residue of a category II job or the incoming job of time t' can be scheduled within I; if I is from the virtual part of S^t then either the residue of a category II job, or the incoming job of time t' might be scheduled within I. The active subinterval of I at time $t > \tau$ is defined as for time τ, the contribution of the incoming jobs from time τ^+ being taken into account.

The *active liberated space* of I at time $t \geq \tau$, $als_t(I)$ is the union of all active subintervals of I at time t. We will denote by $|als_t(I)|$ the summary length of all active subintervals of I at time t. Note that $0 \leq |als_t(I)| \leq |I|$.

Let I' be the latest interval from $\mathcal{I}_k$ for a category I job k. The *initial pending liberated space* by I', $pls_\tau(I')$, is the set of all pending gaps arisen after I' before time τ in $S^\tau(-k+i(\tau))$ (the pending gap(s) arisen within I' are not counted). If k is a category II job, let I' be the interval from $\mathcal{I}_k$ from the virtual part of S^τ. If k is omitted, some other residues might be scheduled within I', and there may arise virtual gaps within the intervals occupied by the above residues in S^τ, between time τ and I'; i.e., there may occur successive right-shift of these residues (similarly as there occur successive left-shift in the real part for a category I job). Analogously as for the category I jobs, $pls_\tau(I')$ is defined as the set of all these new virtual pending gaps. For both, category I and II jobs, the pending liberated space by I' at time $t > \tau$ is defined analogously as for active subintervals, the contribution of the incoming jobs from time τ^+ being similarly taken into account.

For any interval from $\mathcal{I}_k$, different from I', $|pls_t(I)| = 0$.

It follow that for any $I \in \mathcal{I}_k$, $|pls_t(I)| \leq |pls_\tau(I)|$ and $|pls_t(I)| \leq |als_t(I)|$, $|pls_t(I)|$ being the overall length of all gaps from $pls_t(I)$. From now on, for notational simplicity, when this will cause no confusion, we may use $als_t(I)$ and $pls_t(I)$ for $|als_t(I)|$ and $|pls_t(I)|$.

The *profit by interval* I at time $t \geq \tau$ is $profit_t(I) = als_t(I) - pls_t(I)$. Note that $0 \leq profit_t(I) \leq |I|$; $profit_t(I) > 0$ holds for at least one $I \in \mathcal{I}_k$ as k gives an admissible choice. The *profit by job* k at time t, $profit_t(k) = \sum_{I \in \mathcal{I}_k} profit_t(I)$. Note again that $0 < profit_t(k) \leq p_k$. For the notation simplicity, we may use $profit(k)$ for $profit_\tau(k)$.

We can look at $profit_t(k)$ as a sum constituted of two summands, the initial part $profit(k)$ calculated at time τ, and the excrescence. To the initial part category I jobs from S^τ and job $i(\tau)$ contribute. To the excrescence, the filling jobs of k, which are incoming jobs from time τ^+ up to time t, contribute. The profit of k from time τ to time t is increased by the total length of all recuperated pending gaps from time τ up to time t. Later in Section ??? we will see that a filling job of k might be omitted at time $t > \tau$ so that the corresponding liberated space is not completely used by the other jobs. In this case the profit of k will decrease by the total length of the newly arisen pending gaps of k. We can give a recurrent definition of the profit as $profit_t(k) = profit_{t^-}(k) + (|pg_{t^-}(k)| - |pg_t(k)|)$. The difference $|pg_{t^-}(k)| - |pg_t(k)|$ is positive if a gap from $pg_{t^-}(k)$ can be recuperated; it is negative if a new pending gap of job k arises at time t (observe that both cases cannot occur simultaneously). If $i(t)$ is not a filling job of k and no job is omitted at time t, $profit_t(k) = profit_{t^-}(k)$; if $pg_t(k) = \emptyset$ then $profit_t(k) = p_k$.

We will use $profit\_(k)$ for the latest updated value of profit of job k, initially, $profit\_(k) = profit(k)$. We call k a *pending job* at time t if $|pg_t(k)| > 0$, equivalently, $profit_t(k) < p_k$. Job k is *saturated* at time t if $profit_t(k) = p_k$, i.e., the profit of k cannot further in-

crease.

The next proposition immediately follows from the facts that no job will be released within a block which is completed and that no job will be omitted from this block:

Observation 8 *If B^t completes at time t then the profit of none of the jobs will be altered from time t^+.*

Because of Observation 8, from now on, we assume without loss of generality that each job in L^t is an incoming job arrived after the starting time of B^t.

The following fact straightforwardly follows from the profit definition:

Fact 9 *Suppose k and k' are jobs from S^t with an admissible choice such that $profit_t(k) \geq profit_t(k')$. Then if $l \in L_t$ cannot be freely returned to $S^t(-k)$ neither it can be freely returned to $S^t(-k')$.*

Let us say that job profits are *stable* if for any pair of jobs k and l with the activation time τ, $profit_\tau(k) \leq profit_\tau(l)$ implies $profit_t(k) \leq profit_t(l)$, for each $t > \tau$.

Assume from now on that job profits are stable. We already know that whenever the incoming job $i = i(t)$ ($t > \tau$) cannot be perfectly scheduled, either job i or one of the jobs from S^t must be omitted. If i is omitted S^t cannot be enlarged as j cannot be freely returned, since we will get again an infeasible set as at time τ. In general, j cannot be freely returned unless one of the jobs which gave an admissible choice at time τ is omitted at time t. Assume s is such a job. We have $profit_\tau(s) \leq profit_\tau(j)$. As job profits are stable, $profit_t(s) \leq profit_t(j)$, and j cannot be freely returned, by Fact 9. Consider any subsequent scheduling time at which the corresponding incoming job cannot be perfectly scheduled and hence a new job is omitted. Due to the similar arguments as above, none of the earlier omitted jobs can be freely returned. This holds for all subsequent scheduling times as well.

Lemma 10 *Suppose job profits are stable, t is any scheduling time in the MED-Algorithm and O is any set of jobs from S^t. Then there exists no set of jobs $L \subset L^t$ with $|L| \geq |O|$, such that the job set $S^t \setminus \{O\} \cup \{L\}$ is feasible.*

Proof. For each job $j \in O$, for only some $l \in L_t$ the job set $S^t \setminus \{j\} \cup \{l\}$ can be feasible. In particular, j had to give an admissible choice at the scheduling time τ when l was included into the reserve list. We have $profit_\tau(j) \leq profit_\tau(l)$, and since job profits are stable, $profit_t(j) \leq profit_t(l)$. Then it is straightforward to verify, from the definition of the profit, that in the ED-schedule $S^t(-j+l)$, neither we will obtain a new gap within which a job from L^t can be scheduled, nor we will have any left-shift. We apply the same reasoning to all jobs of O successively, updating each time the current schedule according to the above accomplished interchanging. The resultant ED-schedule, as well as all intermediate ones have the claimed property, and the lemma follows. □

If we apply Lemma 10 at the latest scheduling time in MED-Algorithm, we obtain the following result:

Theorem 11 *The MED-Algorithm is optimal if job profits are stable.*

Acknowledgements

This work was Partially supported by CONACyT grant 48433.

References:

[1] P. Baptiste. "An $O(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs". *Operations Research Letters* 24, 175-180 (1999)

[2] P. Baptiste, P.Brucker, S. Knust and V. Timkovsky. "Ten notes on equal-processing-time scheduling". *4OR* 2, 111-127 (2004)

[3] P.Brucker and S.A. Kravchenko. "Preemption can make parallel machine scheduling problems hard". *OSM Reihe P, Helf* 211, Universität Usnabrück (1999)

[4] J.Du, J.Y. Leung and C.S. Wong. "Minimizing the number of late jobs with release time constraint". *Journal of Combinatorial Mathematics and Combinatorial Computing* 11, 97-107 (1992)

[5] E.L. Lawler. "A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs". *Annals of Operations Research* 26, 125-133 (1990)

[6] E.L. Lawler. "Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the tower of sets property". *Mathematical Computer Modelling* 20, 91-106 (1994)

[7] E.L. Lawler. "Efficient implementation of dynamic programming algorithms for sequencing problems". *Report BW106/79* Math. Centre, Amsterdam (1979)

[8] J.M. Moore. "An n job, one machine sequencing algorithm for minimizing the number of late jobs." *Management Science* 15, 102-109 (1968)

[9] N.Vakhania. "A fast algorithm for the preemptive scheduling of equal-length jobs on a single processor". *Proc. 2nd WSEAS Conf. on Computer Engineering and Applications*, p.158-161 (2008)

[10] N.Vakhania. “Fast algorithms for preemptive scheduling of equal-length jobs on a single and identical processors to minimize the number of late jobs”. *Int. J. of Mathematics and Computers in Simulation* 1, p.95-100 (2008)

[11] N. Vakhania. “Scheduling jobs with release times preemptively on a single machine to minimize the number of late jobs”. *Operations Research Letters* 37, 405-410, 2009.