An efficient implicit enumeration for scheduling equal-length jobs with release times on a single processor to maximize throughput

Nodari Vakhania

Abstract We suggest an $O(n^2 \log n)$ combinatorial algorithm for scheduling $n$ equal-length jobs with release times and due dates on a single machine as to minimize the number of late jobs improving the earlier proposed dynamic programming algorithm with the time complexity $O(n^3)$.

1 Introduction

Jobs from $J = \{1, 2, \ldots, n\}$ have to be assigned 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 $d_j$ which is the desired time to complete $j$. Each job needs an integer processing time $p$ on the machine. A schedule assigns to every job $j$ a time interval on the machine with the length $p$ 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. A job is late (on time, respectively) if it 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).

Our problem is commonly abbreviated as $1/p_j = p, r_j/\sum U_j$ ($U_j$ is a 0-1 function taking value 1 if job $j$ is late). Its version with arbitrary job processing times $1/r_j/\sum U_j$ is known to be strongly NP-hard. The preemptive version $1/pmtn, p_j = p, r_j/\sum U_j$ can be solved in time $O(n \log n)$ with an (off-line) dynamic programming algorithm suggested by Lawler [4] for 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 $r_1 \leq r_2 \leq \ldots \leq r_n$, $p_1 \leq p_2 \leq \ldots \leq p_n$ and $w_1 \geq w_2 \geq \ldots \geq w_n$ (an alternative on-line algorithm for $1/pmtn, p_j = p, r_j/\sum U_j$ with the same time complexity was suggested in [7]). Other variations of the single-machine version are known to be polynomial. For arbitrary jobs but without release times, the problem is also easy to solve (see Moore [5] and Lawler [4]).
Our problem was earlier dealt with by Carlier [1]. His algorithm is sub-optimal (see [2] for the details). The first optimal polynomial-time (dynamic programming) algorithm was proposed by Chrobak et al. [2] with the time complexity $O(n^5)$. We improve this result to $O(n^2 \log n)$. Our direct combinatorial algorithm uses the Earliest Due-date heuristic (ED-heuristic) for schedule generation. The algorithm creates $O(n)$ scheduling times to each of which a partial ED-schedule corresponds (one constructed by ED-heuristic). The algorithm incorporates a kind of backtracking, hence the partial ED-schedules are represented as nodes of a solution tree $T$. A special kind of analysis (based on the so-called behavior alternatives) of each next generated schedule is carried out to determine which is to be the next enumerated schedule and whether an optimal schedule was already generated.

2 Some basics

We shall exclusively deal with ED-schedules in which all jobs are completed on time: if a job is late then it can be appended arbitrarily late to such a schedule (without affecting the objective function). From now on, we shall call a schedule feasible if all jobs are completed on time in it. We shall respectively refer to a job set as feasible if there is a feasible schedule including all jobs from the set. We will say that job $j$ conflicts with job $k$ if there is no feasible job set that contains both jobs: if $j$ starts at $r_j$ and $k$ immediately follows $j$ then $k$ is late, and vice-versa. We denote by $|S|$ the number of jobs (completed on time) in a schedule $S$, and by $||S||$ the completion time of the latest scheduled job (the makespan) in $S$. We will use $S$ for both, a schedule and the corresponding set of jobs.

Iteratively, among all released jobs by the current scheduling time $t$, the ED-heuristic schedules a job with the smallest deadline (ties being broken arbitrarily). Initially, $t = \min \{r_l | l \in J\}$; each successive value of $t$ is either the completion time of the latest scheduled so far job (i.e., the previous $t$ plus $p$) or if no job is released by that time, $t$ is the minimal job release time taken among yet unscheduled jobs. We denote by $t^+$ the scheduling time, the next to $t$ and will write $t > t'$ if a scheduling time $t$ occurs after scheduling time $t'$. As it is easily seen, if a job set is feasible then the ED-heuristic will produce a feasible schedule for this set. Since ED-heuristic leaves no machine idle time which can be avoided, it will give the minimal makespan for a job set to which it is applied.

ED-heuristic is not optimal: consider a problem instance with $p = 5$ and two jobs 1 and 2 with $r_1 = 0, d_1 = 11, r_2 = 1$ and $d_2 = 6$. The heuristic will schedule first job 1 and then job 2 at time 5 making late that job, whereas job 2 can be scheduled at time 1 (creating an idle time) followed by job 1 both jobs being completed on time.

An ED-schedule can be seen as a sequence of blocks; a block is a sequence of jobs scheduled in turn without any idle time (a gap) in between. So any gap in an ED-schedule is left outside of any block and it arises only if there is no released job which can be scheduled within that gap. We will denote by $B^t$ the current (uncompleted) block by time $t$ and by $\tau(B^t)$ the starting time of $B^t$. $B^t$ is iteratively completed by new incoming jobs until the next incoming job starts new block. Before this happens, $B^t$ is an extension of $B^{t'}$, for $t' < t$ and $\tau(B^t) = \tau(B^{t'})$.

We call a job selected by ED-heuristic at scheduling time $t$ an incoming job of time $t$; as we will see, it may happen that the current incoming job of time $t$ is omitted,
hence the current scheduling time does not change and the next incoming job also occurs at time \( t \). We will use \( i(t) \) for the latest so far arisen incoming job by time \( t \).

We shall deal with partial ED-schedules and will denote the current partial ED-schedule generated to time \( t \) by \( S^t \). \( S = S^t \) can be extended by \( i = i(t) \) at time \( t \). The resultant ED-schedule, in which \( i \) is scheduled in the interval \([t, t + p]\), is denoted by \( S(+i) \). \( S^t \) is always set to \( S(+i) \) whenever \( i \) is scheduled on time in \( S(+i) \). Suppose \( i \) is late in \( S(+i) \) (our algorithm is organized in such a way that the only late job in \( S(+i) \) might be \( i \)). The job set \( S \cup \{i\} \) can be feasible or not. If \( S \cup \{i\} \) is not feasible, some job from \( S \cup \{i\} \) is to be omitted to get a feasible schedule. We will say that a job is disregarded (omitted, respectively) if it is discarded and is never again considered for the inclusion (discarded but might later again be considered for the inclusion, respectively).

Suppose \( i(t) \) does not conflict with the latest scheduled job in \( S = S^t \) and still cannot be scheduled on time. We wish to know whether the job set \( S \cup \{i\} \) is feasible. Let us call job \( e \in B^t \) with \( d_e > d_i \) an emerging job at time \( t \). We may have none, one or more emerging jobs at time \( t \) which will be referred to as alternative emerging jobs. Given an emerging job \( e \), a derived ED-schedule denoted by \( S_e \) (a complementary to \( S \) schedule or \( C \)-schedule for short) has the following property. In \( S_e \), the job \( e \) and any other job not from \( S \) is forced to be scheduled after job \( i \) (so we schedule 1 less job before \( i \)). We will say that \( e \) is activated for job \( i \). We will refer to job \( m \in B^t \) as marched by time \( t \) if it is in the state of activation in \( S \) (we may have none, one or more marched jobs in \( S \)). Among the alternative emerging jobs, one with the maximal ordinal number is called the live (it will commonly be denoted by \( l \)).

To actually create \( S_e \), we first increase artificially the release time of \( e \) to \( r_j \) and that of any \( j \not\in S \) to \( \max\{r_i, r_j\} \). Then we apply the ED-heuristic to the modified instance: none of the above jobs can then become an incoming job ahead of job \( i \). Once we activate \( e \) we again apply ED-heuristic for the yet unscheduled (non-disregarded) jobs till the next incoming job is late (we will successfully generate our destiny optimal schedule if there occurs no late job). We shall deal with the case when this late job is again \( i \) in Section 3. Otherwise, \( e \) will become an incoming job at some time \( t' > t \) and will be included if it can be completed on time (we shall deal with the other case also in Section 3). If this happens, \( e \) might be activated again for some following incoming job \( ii(t'', t'' > t', and this scenario might be repeated.

The set of jobs in \( S \) scheduled between the live emerging job \( l \) and \( i \), including \( i \), is called the kernel of \( S \), denoted by \( K(S) \). All kernel jobs are more urgent than any corresponding emerging job. The next statement from [6] (Lemma 1) follows from ED-heuristic and from the facts that \( r_i > r_e \) and that all potential jobs which could have been becoming incoming jobs ahead of job \( i \) are forced to be scheduled after job \( i \). We use \( o(e, S) \) for the ordinal number of \( e \) in \( S \):

**Fact 1** There arises (at least one) gap between the \((o(e, S) - 1)\)st scheduled job and the earliest scheduled job of \( K(S) \) in \( S_e \). Consequently, kernel jobs restart earlier. Moreover, if \( e \) is the live emerging job, then there will arise a gap immediately before the earliest scheduled job of \( K \) in \( S_e \) (hence the earliest kernel job will be scheduled at its release time).

It will also be useful another easily seen fact from the above reference [6]:

**Fact 2** Suppose \( k \) and \( e \) are alternative emerging jobs in \( S \) with \( o(k, S) < o(e, S) \). Then the starting time of the earliest scheduled job of \( K(S) \) in \( S_k \) is more than or equal to
that of the earliest scheduled job of $K(S)$ in $S_e$. In general, the left-shift of the kernel jobs and successive jobs in $S_e$ is no less than that in $S_k$.

Note that the gaps from Fact 1 can be avoided, unlike normal (natural) gaps arisen while applying ED-heuristic to the original problem instance. We shall refer to the former gaps as the ones *imposed* by (the activation of) $e$. Note also that the imposed gaps are not relevant to the block definition (no imposed gap starts a new block, and any idle time within a block corresponds to an imposed gap). The imposed gap(s) arisen after the activation of $e$ for some $i$ will disappear if the following operations are carried out: (1) the release time of the corresponding jobs with the modified released times including $e$, prior to that activation is restored; (2) ED-heuristic is applied to that newly modified instance. We shall refer to these operations as the *revision of e* for $i$. If a job is revised for the earliest incoming job for which it was activated then it will be returned to its “natural place” and will no longer be marched. We shall use $en(t)$ for the emerging job that is in the state of activation for $i$ (i.e., it was activated for $i$ and has not been revised; in general, no revised job can again be activated).

We will distinguish emerging jobs scheduled after any imposed gap from those scheduled before some imposed gap in $B^i$. We call former jobs *normal* and latter jobs *passive* emerging jobs in $S^i$ (in the sequel, an “emerging job” is used for either of these types). Once an emerging job is marched all emerging jobs scheduled originally before the activated emerging job become passive, and every passive emerging job yields at least one marched job. If there is no marched emerging job $m$ then no passive one may exist. A marched emerging job may also become a passive (and then again a normal) emerging job.

A passive emerging job $π$ is “isolated” from the following part of $S^i$ by the imposed gap(s) arisen after the activation of the corresponding marched job(s). Because of these gaps, a passive emerging job $π$, unlike a normal one, cannot be beneficially activated as this cannot give the desired left-shift: if there is an (imposed) gap between jobs $π$ and $j$, clearly $j$ cannot be restarted earlier if $π$ is rescheduled after $j$. Thus a passive emerging job is of no use before it is converted into a normal one. For this, all imposed gaps after $π$ are to be eliminated, hence the corresponding marching emerged job(s) need to be revised.

**Proposition 1** Suppose the late incoming job $i = i(t)$ is not marched, and there exists no emerging job (neither normal nor passive) in $S^i$. Then the set $S^i(+i)$ is not feasible. Moreover, $i$ can be disregarded.

Proof. From the conditions it follows that the only possibility to schedule $i$ on time is to reschedule some job $j ∈ B^i$ with $d_j ≤ d_i$ after $i$ or to omit such a $j$. $j$ will be late if rescheduled after $i$, so the set $S^i ∪ \{i\}$ is not feasible and either $i$ or $j$ is to be omitted. Clearly, the implications of each of these options are to be seen on scheduling times $t' > t$ while considering the possibility of the inclusion of $i(t')$. In particular, $t'$ is to be the minimal possible given the number of the earlier included jobs. By Fact 2 and due to $p_i = p_j$, this will be provided by the omission of $i$.

By activating an emerging job more urgent jobs are given a chance to be (re)started earlier. A marched job will remain unscheduled as long as yet unscheduled jobs are more urgent and no idle time is created by ED-heuristic. A marched job $m$ is either already included into $S^i$ or is $i(t)$ or neither it belong to $S^i$ nor it is $i(t)$. In the latter case $m$ will be referred to as hidden by time $t$. We may have none, one or more hidden jobs at time $t$. The same job may become hidden and again included into the current schedule.
two or more times. In this case this job should have been activated the same number
of times for different incoming jobs (each activation yielding its own imposed gaps).

3 Behavior alternatives and their analysis

Our further decisions for the case when \( i = i(t) \) is late are based on the case-analysis
of our behavior alternatives, the first of which is introduced now. We will say that an
instance of alternative (a) (IA(a) for short) occurs at time \( t \) if \( i \) is the late incoming
job not only in \( S(+i) \) but also in \( S_l(+i) \) (as always, \( S = S^t \) and \( l \) is the live emerging
job in \( S \)).

**Proposition 2** The incoming job \( i(t) \) can be disregarded if there occurs IA(a) at time
\( t \).

Proof. First we note that \( K \setminus \{i\} = \emptyset \) is not possible. Indeed, this would mean that \( l \)
immediately precedes \( i \) in \( S(+i) \). Then there is a gap immediately before job \( i \) in \( S_t \),
i.e., \( i \) starts at its release time and hence cannot be late (release times and deadlines
are agreeable ). So, suppose \( K \setminus \{i\} \neq \emptyset \). As we have observed in Fact 1, there is a
gap before the earliest scheduled job of \( K \) in \( S_t \), i.e., the first scheduled job of \( K \) starts
at its release time in \( S_t \). Hence, the only possibility to schedule \( i \) earlier (than it was
scheduled in \( S_l(+i) \)) is to reschedule a job from \( K \setminus \{i\} \) after \( i \). But by the definition
of job \( l \), all jobs from \( K \) have a due date no grater than \( d_i \). Hence, any of these jobs
will be late if rescheduled after \( i \) and \( i \) can be disregarded, similarly as in Proposition
1.

Due to Proposition 2, assume for the rest of this section that no IA(a) occurs.
Suppose again \( i = i(t) \) is late and there exists no emerging job in \( S^t \); we call a marched
job \( m \) squeezed if either \( m = i \) or \( m \in B^t \) (so \( m \) is not hidden and it is not an emerging
job in \( S^t \)). Intuitively, squeezed jobs are these marched jobs whose due-date turns
out to be insufficiently small; they did some work but they got “old” (the younger
generation with a larger due-date is needed).

Now we give a brief description of our first ALGORITHM I, which works under
the assumption that at none of the scheduling times a squeezed/hidden job arises. If
during the execution of ED-heuristic on none of the scheduling times a late job occurs
then the schedule produced by ED-heuristic is clearly optimal. Otherwise, we omit a
late incoming job \( i(t) \) if one of the Propositions 1, 2 at time \( t \) apply, and we continue
with the next scheduling time. Otherwise, if there exists a normal emerging job at
time \( t \) then we activate the live emerging job \( l \) (i.e., we set \( S^t := S_l(+i) \)). Otherwise,
as Proposition 1 did not apply, there must exist a squeezed or a hidden job in \( S^t \).
ALGORITHM I has no predetermined actions for this case with which we deal with
below. The reader should have no difficulty in completing our observations to a proof
of this lemma:

**Lemma 1** ALGORITHM I generates an optimal schedule if no squeezed/hidden job
during its execution arises.

If during the execution of ALGORITHM I a squeezed/hidden job at time \( t \) occurs
then there might be necessary to reconsider some activation(s) carried out before time
\( t \) rather than just discard job \( i(t) \). For example, we may have yet inactivated (pas-
sive) emerging job with a small enough due-date that may successfully substitute the
activation(s) of the squeezed job(s). We study such circumstances in what follows.
We will say that an instance of alternative (b1) (IA(b1)) at time t occurs if the late incoming job \( i = i(t) \) is squeezed (this case was not dealt with in Proposition 1). We also distinguish a variation of IA(b1). We again have a squeezed job \( m \) in \( S^l \) but it is not now \( i(t) \), and again there is no normal emerging job in \( S^l \) (\( i(t) \) being late). As for IA(b1), we may have a passive emerging job in \( B^l \) or may not. If there exists no passive emerging job then let \( m_1, m_2, \ldots, m_l \), \( l \geq 1 \), be the marched jobs in \( B^l \) in the order as they were activated. Otherwise let \( \pi \) be the live (the latest scheduled) passive emerging job in \( S^l \), and let \( m_1, m_2, \ldots, m_l \), \( l \geq 1 \), be the intermediate marched jobs (ones activated already once \( \pi \) was scheduled) also ordered according to their activation times (\( \pi \), in general may also be a marched job). Observe that none of \( m_1 \)s is an emerging job in \( S^l \) and these jobs have imposed the gaps following job \( \pi \) in \( S^l \) (they are in the state of activation in \( S^l \)). Moreover, \( \pi \) cannot be (beneficially) activated unless all these intermediate marched jobs are revised. Each \( m_1 \) is squeezed unless it is hidden. If all \( m_1 \)s are squeezed (none of them is hidden) then we will say that an instance of alternative (b2) (IA(b2)) at time t occurs. Otherwise (at least one of \( m_1 \)s is hidden) we will say that an instance of alternative (h) (IA(h) for short) at time t occurs. Recall that by our assumption in none of the schedules IA(a) occur. In other words, every incoming job for which one of the squeezed/hidden emerging job was activated has been included in the corresponding ED-schedule.

While for IA(b1/b2) the activation of a passive emerging job may potentially be beneficial, this is not the case for IA(h) (note that this claim does not apply to a normal emerging job, i.e., a normal emerging job can be beneficially activated even in the presence of a hidden job). Indeed, let among all hidden \( m_1 \)s \( h \) be the latest activated one, and \( \gamma \) be the latest imposed gap by \( h \). Then clearly, the set constituted by the jobs scheduled after \( \gamma \) in \( S^l \) and job \( i(t) \) is not feasible; then one of these jobs is to be omitted in any feasible schedule. Similarly as in Proposition 1, we easily obtain that \( i(t) \) can be omitted as long as \( h \) remains in the state of activation. We return to this point with more details later in this section.

For IA(b1/b2), an alternative passive emerging job \( \pi \) may be beneficially used as a substitution for the corresponding squeezed job(s). In general, more than one (alternative) passive emerging job may exist. Each such an emerging job may potentially be converted into a normal one only if the corresponding intermediate marched jobs are revised. Alternative emerging jobs can always tried in the decreasing order of their ordinal numbers by on Fact 2. Thus at each scheduling time t with the late \( i(t) \), among all alternative emerging jobs the live emerging job is always activated first. Likewise, whenever there occurs a need in an alternative emerging job (IA(b1/b2) arises), among all yet untried emerging jobs the latest scheduled one is activated the next. In other words, the alternative emerging jobs are tried in the decreasing order of their ordinal numbers: each next activated alternative (former passive) emerging job \( \epsilon \) has a smaller ordinal number than the previously activated (and squeezed) one, say \( e \), and besides \( d_\epsilon < d_e \). Recall that before the earliest (live) passive emerging can be activated all the imposed gaps after \( \pi \) have to disappear, i.e., every intermediate \( m_1 \) is to be revised. In case \( m_1 \) was activated several times, all these activations will be reverted if \( m_1 \) is revised for the earliest incoming job for which it was activated. Note that no yet untried passive emerging job could have been scheduled in between these activations of \( m_1 \), as otherwise such a job would have been activated ahead \( m_1 \).

The live passive emerging job \( \pi \) can be converted into a normal emerging job at time t if the intermediate \( m_1 \)s are revised and ED-heuristic is applied (then \( \pi \) becomes the latest scheduled yet untried normal emerging job to be activated next). For IA(b1),
the squeezed \( i(t) \) is to be revised as well. The release times of all these squeezed jobs and that of job \( \pi \) are modified in one turn and ED-heuristic to the modified problem instance is applied. We shall refer to these our actions as the substitution initiated by \( \pi \). Note that while \( \pi \) is activated for the current late incoming job it is also substituting the earlier activations of the \( m_i s \) (and \( i(t) \) in case of IA(b1)).

Thus whenever a former passive emerging job becomes squeezed (IA(b1/b2) occurs), the substitution initiated by the live passive emerging job is carried out. In this way, a series of repeated instances of alternative (b1/b2) may occur (each new such an instance will arise as long as the next passive emerging job exists). Consider the \( k \)th element from the series, \( k \geq 2 \), occurring at time \( t \). The left-shift for the jobs included after the imposed gaps for the \( k \)th element might be less than that for the \( (k-1) \)th instance due to Fact 2. Hence, these jobs may start later than they were started before the \( k \)th instance (in \( S^t \)) and some may become late. We wish to include the late job \( i(t) \) on time, whereas the number of jobs scheduled before \( i(t) \) is to be one less than \( |S^t| \); it cannot be \( |S^t| \) as now a new emerging job \( j \) is to be included after \( i(t) \), and it should not be less than \( |S^t| - 1 \) (as otherwise we will be loosing an extra job). Assume the left-shift corresponding to the \( k \)th element from the series is “not enough”: either \( i(t) \) could not have been included or the number of jobs included before \( i(t) \) is less than \( |S^t| - 1 \). Then we will say that the substitution initiated by job \( j \) is weak, where \( j \) is the live passive emerging job of time \( t \) (the one initiated the substitution corresponding to the \( k \)th element).

Suppose the substitution initiated by \( j \) is weak, and we substitute \( j \) by an earlier scheduled (yet untried) alternative emerging job still hoping to recuperate the late job \( i(t) \). Then due to Fact 2 this new substitution will also be weak and hence nothing is to be gained from it. Thus all yet untried potential substitutions can be neglected whenever the latest accomplished substitution is weak:

**Proposition 3** All yet untried potential substitutions can be neglected if the substitution initiated by a former passive emerging job is weak.

**An exhaustive instance of alternative (b).** We introduce our last behavior alternative, a special case of IA(b1/b2). It delineates a stage in the algorithm when the omission of one extra job (either the current incoming job or some job from \( B^t \)) becomes unavoidable, as all the potentially useful emerging jobs have already been tried. There occurs an instance of one of the alternatives (b1/b2) or the latest accomplished activation is weak: either there existed no passive emerging job for that IA(b1/b2), or all the (former passive) alternative emerging jobs have already been tried (with the outcome of IA(b1/b2)) or the latest accomplished substitution was weak (see Proposition 3). Then we will say that there occurs an exhaustive instance of alternative (b) at time \( t \) (ELA(b) for short).

Consider the earliest instance of alternative (b1/b2) from the respective series of instances of alternative (b1/b2) ending with ELA(b). If there exists no passive emerging job then the series consists of this single instance of alternative (b1/b2); otherwise, the number of instances (b1/b2) in the series is bounded by the corresponding number of the passive emerging jobs. In general, we may have two outcomes for a series consisting of one or more occurrences of IA(b1/b2): it either completes with ELA(b) (the unsuccessful outcome), or the latest activated/substituted emerging job turns out to be an appropriate candidate, i.e., neither it becomes squeezed nor the corresponding
substitution is weak, hence no EIA(b) for the corresponding series occurs (the successful outcome). In the latter case another EIA(b) may occur only within some succeeding block.

Now we discuss in more details the implications of exhaustive instances of alternative (b) assuming that EIA(b) at time \( t \) arises. We use \( ED(S) \) for the ED-schedule generated for job set \( S \), and we denote by \( I(B_i) \) the set of all the former late incoming jobs in block \( B_i \). Observe that each \( i \in I(B_i) \) has been successfully processed within \( B_i \), i.e., an emerging job for \( i \) was activated (with the non-weak outcome). From the definition of EIA(b) and Propositions 1 and 3 it follows that the job set \( B_i \cup \{ i(t) \} \) is not feasible. Hence, a feasible schedule cannot contain all jobs from this set, i.e., one of these jobs is to be omitted. We claim that among all such candidate jobs \( j \in B_i \cup \{ i(t) \} \) with \( |ED(B_i \cup \{ i(t) \} \setminus \{ j \})| = |B_i'| \), one yielding the minimum \( |ED(B_i \cup \{ i(t) \} \setminus \{ j \})| \) can be omitted. Indeed, \( |B_i'| \) is the maximal number of jobs in \( B_i \cup \{ i(t) \} \) that can be feasibly scheduled. By the definition of EIA(b) and Proposition 3, no job from \( B_i \) can be beneficially activated at any scheduling time \( t' > t \). Then no job from \( B_i \cup \{ i(t) \} \) is to be inserted at or after scheduling time \( t' \) (see Proposition 1) and our claim clearly follows. Now, by omitting any \( j \) we will liberate an interval with the same length \( p \).

The later a job is scheduled the more might be the left-shift for successively scheduled jobs. By the definition of EIA(b) and Proposition 3, no job from \( B_i \) we give a possibility for the left-shift to the jobs have been scheduled after these gaps. This gives a potential for the left-shift to the jobs have been scheduled after these gaps. The following lemma is now apparent:

**Lemma 2 (A sub-block optimality Lemma)** Job set \( B_i \cup \{ i(t) \} \) is not feasible if EIA(b) at time \( t \) occurs, hence at least one job from the set is to be omitted. Furthermore, among all such candidate jobs \( j \in B_i \cup \{ i(t) \} \) with \( |ED(B_i \cup \{ i(t) \} \setminus \{ j \})| = |B_i'| \), one yielding the minimum \( |ED(B_i \cup \{ i(t) \} \setminus \{ j \})| \) can be omitted. This minimum is reached when \( j \in I(B_i) \cup \{ i(t) \} \) and \( wj(m(j)) \) is revised for \( j \) for \( j \in I(B_i) \), and by omitting \( j \) for \( j = i(t) \).

Based on this lemma a procedure to find a job in \( I(B_i) \cup \{ i(t) \} \) minimizing the above magnitude is straightforward. The procedure will have no special cost within the framework of the whole algorithm (though its direct application yields the running time \( O(\nu n \log n) \), where \( \nu \) is the number of jobs in \( I(B_i) \)). Let us first give some details of this procedure called \( \text{SCHEDULE_SUBBLOCK}(t) \). We aim to find a \( j \in I(B_i) \cup \{ i(t) \} \) with the minimum \( |ED(B_i \cup \{ i(t) \} \setminus \{ j \})| \) provided that \( |ED(B_i \cup \{ i(t) \} \setminus \{ j \})| = |B_i'| \). Either \( j \in I(B_i) \) or \( j = i(t) \). By omitting a job from \( I(B_i) \) we give a possibility for the left-shift to the succeeding jobs from \( B_i \) and \( i(t) \), while the above minimum provides the earliest starting for the jobs which will be scheduled after time \( t \) (given the number of jobs included by time \( t \).) Let \( j' \) be any job from \( I(B_i) \) succeeding \( j \) in \( S' \). We wish to keep both \( j' \) and \( wj(m(j')) \) scheduled. Moreover, we have to revise \( wj(m(j')) \) for \( j' \) in order to give a chance to the succeeding jobs to be left-shifted. (Remind that even though job \( j \) is omitted, such left-shift cannot occur as long as the corresponding imposed gap(s) by \( wj(m(j')) \) remain; these gaps will disappear if \( wj(m(j')) \) is revised for \( j' \).) Let \( \tau \) be the starting time of \( wj(m(j')) \) prior to its activation for \( j' \) (i.e., that in \( S' \), with \( j' = i(t') \)). Roughly, if \( wj(m(j')) \) is not released early enough before time \( \tau \) then it cannot be sufficiently left-shifted within the liberated space due to the omission of job \( j \). In particular, this left-shift should be more than \( p \), as otherwise we may provide a not less left-shift for the rest of the succeeding jobs by just revising \( wj(m(j')) \) for \( j' \), omitting...
got hidden. We have the gap(s) imposed by each
in the selection of a subset of jobs to be included.

τ(h)). Each new dependent job
is omitted (but not disregarded) at time
t after the imposed gap(s) by (any)
hι becomes hidden while all
Bι block
I omit em job (this definition extends, in general, to any instance of alternative
j blocks unless some
em job is activated for two or more times for different jobs
from I(Bί) (in row). In this particular case, when considering the possibility of the
omission of each of these jobs from I(Bί), em(j) is revised for that selected job.

Consequences of IA(h). Consider the earliest IA(h) occurring within our current
block Bί, h1,h2,...,hk being the corresponding hidden jobs at time t. Since each hi
becomes hidden while all h1,h2,...,hi−1 still remain hidden, dh1 > dh2 > ... > dhk.
We remind that i(t) is late and there exists no normal emerging job in Sί (that could
potentially become another hidden job). We have also seen that the set of jobs scheduled
after the imposed gap(s) by (any) hi together with job i(t) is not feasible, and that i(t)
is omitted (but not disregarded) at time t. We shall refer to each such an omitted i(t)
as a dependent job (this definition extends, in general, to any instance of alternative
(hi)). Each new dependent job i(τ) ∈ Bί is omitted at time τ. Hence the current τ does
not change and there may occur more than one dependent job at time τ. As we will see
now, dependent jobs delineate an interval within which there are more candidates
to be included than can actually be (on time) included. Hence we have some freedom
in the selection of a subset of jobs to be included.

Let i(hi) be the (former late incoming) job for which hi was activated before it got
hidden (i = 1, 2, ..., k), and let s(h) be the start time of h before it was activated and
got hidden. We have the gap(s) imposed by each hi between time s(hi) and job i(hi) in
Sί (with the total length less than p); if any job is scheduled within such a gap then i(τ)
cannot be included on time and will be omitted. This might be a correct thing to do as
we will see now. When we have activated hi for i(hi) we hoped to keep jobs i(hi), hi
gether with the next incoming jobs scheduled. However, this hope disappears when
corresponding dependent job occurs. The set of jobs scheduled between s(hi) and
t (including i(hi)), together with the dependent jobs, is not feasible: too many jobs
to be included within that interval. It might be better to omit i(hi) and include a
dependent job as the corresponding imposed gaps might be reduced or eliminated at
all (by scheduling job(s) released within or before these gaps). This, in turn, gives a
potential to (restart)later scheduled jobs earlier and hence to increase the number of
the included jobs.

Roughly, if we have enough dependent jobs then we can include one dependent job
instead of some i(hi)s in case this yields the reduction of the imposed gaps between
times s(h1) and t. If there are enough jobs released early enough so that all the imposed
gaps disappear then we may clearly assert that the resultant ED-portion is optimal,
that is, it is to be a part of the destiny optimal schedule.
We next specify how a set of dependent jobs can naturally be associated with some hidden jobs. Recall that we had $k$ hidden jobs $h_1, h_2, \ldots, h_k$ when the earliest instance of alternative (h) within the current $B^t$ at time $t$ has occurred: $i(t)$ is the first arisen late incoming job for which no normal emerging job exists, we have called it dependent job (given that there is at least one hidden job). Once $i(t)$ is omitted, there may occur repeated IA(h) at time $t$, i.e., the next incoming job may again turn out to be a dependent job. A similar picture might be repeated at time $t$ or at some succeeding scheduling time.

Let $t' > t$ be the earliest scheduling time such that job $h_k$ is no more hidden, i.e., $i(t') = h_k$ (as $d_{h_1} > d_{h_2} > \ldots > d_{h_k}$, $h_k$ will be the first of $h_n$ that will (again) become an incoming job). Further, let $D(t')$ be the set of dependent jobs arisen in between times $t$ and $t'$. No $j \in D(t')$ can be released before time $s(h_k)$. Indeed, by definition of $j$, $d_j < d_{h_k}$ and hence ED-heuristic would include $j$ instead of $h_k$ if $j$ were released by time $s(h_k)$. So $j$ can potentially be included only after time $s(h_k)$, hence cannot “substitute” any $i(h_i), i = 1, 2, \ldots, k - 1$. Hence jobs from $D(t')$ can be included only within the interval $[s(h_k), t')$, instead of job $i(h_k)$ (reducing or eliminating at all the corresponding imposed gaps). So $D(t')$ is naturally associated with $h_k$ (and with none of the other hidden jobs). Later in the formal description we will also use $D(h_k)$ for this set, and we call its jobs dependent on $h_k$. As we have already seen, $D(h_i) = \emptyset$ for $i < k$, i.e., to some former hidden jobs no dependent job may correspond.

Either (i) $h_k = i(t')$ is included at time $t'$ or (ii) it is late (hence IA(h1) at time $t'$ occurs). Consider first case (i): job $h_k$ is no more hidden and it got included. $D(h_k)$ cannot grow any more and we may try to reschedule the interval $[s(h_k), t')$. This time, we do not need to force job $i(h_k)$ to be included, we know that this job might be omitted and a corresponding dependent job included instead. We restore the release times of all the jobs except job $h_k$ that were artificially modified (increased) during the activation of $h_k$ for $i(h_k)$ (such a job might well be one from $D(h_k)$); now these jobs will no more be forced to be scheduled after $i(h_k)$ and hence might be rescheduled earlier and the imposed gaps by $h_k$ can respectively be reduced/eliminated. We call this the weak revision of job $h_k$. Note that the current release time of job $h_k$ itself is not modified as we still hope to include both, $h_k$ and some dependent job from $D(t')$ (thus the name for such a revision). At the same time such an option is not to be realized if there still remains an imposed gap by $h_k$ once it is weakly revised and $h_k$ gets squeezed at some posterior scheduling time (in the case the (normal) revision of $h_k$ is clearly better as all the imposed gaps will be eliminated). Thus we came to the following proposition:

**Proposition 4** The portion of ED-schedule obtained after the weak revision of $h_k$ is optimal if $h_k$, once revised, does not get squeezed.

The case not covered in this proposition ($h_k$ becomes squeezed) and case (ii) above are similar: in the former case we have IA(h2), and in the latter case IA(h1) occurs (hence we revise job $h_k$ for $i(h_k)$ as earlier specified for these instances).

Once $h_k$ is scheduled at time $t'$ and processed as above, another hidden job $h_{k+1}$ with a non-empty $D(h_{k+1})$ may occur, and so on. Observe at the same time that at most one yet unprocessed hidden job $h$ with a non-empty $D(h)$ for each scheduling time $t$ may exist. We shall refer to that job as the heading (hidden) job of time $t$. 
4 Summarizing the algorithm

In the formal description of our algorithm $t$ is the current scheduling time with $S = S^t$ being the current schedule and $i = i(t)$ the corresponding incoming job. The values of $t$ and $S$ are updated according to the description. ACTIVATE($l$) activates the live emerging job $l$ for the late incoming job.

**Algorithm II**

Initial settings:
$s := ∅$; $t := \min\{r_j | j ∈ J\}$

Procedure MAIN

IF no unconsidered job is left THEN output $S$; halt

IF $i = i(t)$ is on time in $S(+) \text{ THEN } (S^t := S(+) \text{; weakly revise } i \text{ if it is the heading job})$

ELSE {
  $i$ is late in $S(+) \text{ THEN carry out the substitution initiated by } \pi$ if it is not weak
}

BEGIN {
  $i$ is late
  IF $i$ conflicts with the latest scheduled job in $S$ THEN omit $i$
  IF there exists no emerging job and $i$ is not marched in $B^t$ THEN disregard $i$
  IF there exists a normal emerging job THEN ACTIVATE($l$)
  IF there occurs IA(a) THEN disregard $i$ and revise $l$ for $i$
  IF there occurs IA(h) THEN update $D(h)$ for the heading job $h$
  IF there occurs IA(b1/b2) and there exists the live passive emerging job $\pi$
    THEN carry out the substitution initiated by $\pi$ if it is not weak
  ELSE {
    there occurs EIA(b) \text{ SCHEDULE_SUBBLOCK($t$)}
  }
  END { $i$ is late }
}

End { main }.

**Theorem 1** ALGORITHM II produces an optimal schedule in time $O(n^2 \log n)$.

Proof. The schedule generated by the algorithm is optimal if the ED-heuristic schedules each incoming job on time. The time complexity in this case is $O(n \log n)$ (that of the ED-heuristic). In general, the number of scheduling times is $O(n)$. Indeed, from any scheduling time $t$ to time $t^+$, either one job is scheduled, or one job is omitted or a substitution initiated by the next passive emerging job is carried out. Each such a job may initiate a single substitution whereas the total number of emerging jobs is $O(n)$. Hence, the total number of scheduling times is $O(n) + O(n) + O(n) = O(n)$. At any scheduling time $t$, we either add or omit a job (with no extra cost), or activate the live emerging job or carry out a substitution with a brutal cost of $O(n \log n)$ (that of ED-heuristic), or call procedure SCHEDULE_SUBBLOCK($t$). The latter implies the application of ED-heuristic to the modified problem instance from the (earlier) scheduling time, the next to time $r_{em}(j)$, for every $j ∈ I(B^t)$. This yields a single extra application of ED-heuristic for some earlier considered scheduling times, hence it has no extra cost. So the overall time complexity is $O(n^2 \log n)$.

The soundness part follows from the already established facts. We give a sketch. A late incoming job $i(t)$ can be disregarded if it is not marched and there exists neither normal nor passive emerging job in $S^t$ (no marched job in $B^t$) By Proposition 1. By Fact 2, if there exists a normal emerging job in $S^t$ then the live one is to be tried first; more generally, alternative emerging jobs are to be tried in the decreasing order of their ordinal numbers. If Proposition 1 does not apply and there exists no (yet untried) normal emerging job in $S^t$ then one of the behavior alternatives will occur (whenever $i$ is late). So assume there exists a normal emerging job at time $t$ and we activate the live emerging job $l$. If in the resultant C-schedule $i$ still cannot be included on-time,
i.e., IA(a) occurs, i is disregarded by Proposition 2. It remains to consider instances of alternatives (b1/b2), (h) and exhaustive instances of alternative (b). As to IA(h), as we have seen in the previous section, each heading job that doesn’t become squeezed is to be weakly revised (consequences of IA(h)). In the rest of the cases IA(b1/b2) or/and EIA(b) occurs. For EIA(b) procedure SCHEDULE_SUBBLOCK(t) produces a segment of the destiny optimal schedule by Lemma 2. Finally, as we have already seen for IA(b1/b2), a substitution initiated by the next (yet untried) alternative emerging job is to be carried out.

5 Acknowledgements

The author is grateful to Christoph Durr for his attention and comments.

References

7. 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)