# A Polynomially Solvable Case of a Single Machine Scheduling Problem When the Maximal Job Processing Time is a Constant

Nodari Vakhania* Frank Werner**

*Science Faculty, UAEM, Mexico
** Faculty of Mathematics, Otto-von-Guericke Universität Magdeburg, Germany

INCOM 2012, Bucharest/Romania, May 23-25, 2012

# Outline of the Talk

1. Introduction
2. Brief Description of the Algorithm
3. Binary Search Procedure
4. Seeking after $S(\delta)$: Procedure $SEEK(S(\delta))$

# 1. Introduction

$n$ jobs: $1, 2, \ldots, n$ are to be scheduled on a single machine

for each job $j$, there are given:

$p_j$ - processing time

$r_j$ - release times

$d_j$ - due-date

**schedule** $S$: described by the starting time $t_j(S)$ (or the completion time $c_j(S) = t_j(S) + p_j$ of all jobs $j$)

**Objective:** Find an optimal schedule $S$ that minimizes the *maximal lateness* $f(S) = L_{\max}(S) = \max\{c_j - d_j \mid j = 1, 2, \ldots, n\}$.

$f(j)$ – *lateness* of job $j$

*Garey and Johnson (1978):* Problem $1|r_j|L_{\max}$ is strongly NP-hard.

**Polynomially solvable cases**

*Jackson (1955):* $O(n \log n)$ algorithm for problem $1||L_{max}$ (and problem $1|r_j, d_j = d|L_{max}$, respectively)

*Garey et al. (1981):* $O(n \log n)$ algorithm for problem $1|p_j = p, r_j|L_{\max}$

*Vakhania (2004):* $O(n^2 \log n)$ algorithm for problem $1|p_j \in \{p, 2p\}, r_j|L_{\max}$

*Vakhania (2011):* $O(n^2 \log n \log p_{max})$ algorithm for problem $1|p_j : \mathit{divisible}, r_j|L_{\max}$

**but:** problem $1|p_j \in \{p, 2p, 3p, \ldots\}, r_j|L_{\max}$ is *NP*-hard

**now:** consideration of problem $1|p_j \in \{p, 2p, 3p, \ldots, kp\}, r_j|L_{\max}$

# 2. Brief Description of the Algorithm

Our framework yields an $O(n^2 \log n \log p_{\max})$ algorithm. The algorithm uses **binary search** and reduces the problem to a version of the **bin packing** problem.

The set of jobs is partitioned into **non-critical** and **critical** subsets. The non-critical subsets contain jobs that might be flexibly moved within the schedule.

The **critical sets (kernels)** contain the jobs which form tight sequences in the sense that the delay of the earliest scheduled job from the subset cannot exceed some calculated parameter between (including) 0 and $p_{\max}$.

When the delay of the latter job is 0, the lateness of the latest scheduled job from the set defines a valid **lower bound** on the optimal value.

# 2. Brief Description of the Algorithm

Just by applying the ED-heuristic to the original problem instance we define the (initial) **set of kernels** and then determine the above lower bounds yielded by each kernel. The maximum among them is a valid lower bound for the problem.

It also delineates the maximal delay $\Delta$ that might be imposed to other kernels without increasing the maximum lateness, whereas the minimal possible delay is 0.

Then we carry a **binary search** within the interval $[0, \Delta]$ to find the minimal possible delay $\delta$ that would result in an optimal schedule:

For each $\delta$, we try to distribute non-kernel jobs in order to utilize the intervals in between kernels in an optimal way so that no non-kernel job has the lateness more than that of a kernel job.

# 2. Brief Description of the Algorithm

**Related bin packing problem**

We have a fixed number of bins (intervals between the kernels) of different capacities and we wish to know if the given items (non-kernel jobs) can be distributed into these bins.

**ED-heuristic**

Iteratively, among all available jobs at time $t$, ED-H schedules a job with the smallest due-date breaking ties by selecting a longest job. Here $t$ is the maximum between the minimal release time of yet unscheduled job and the time when the machine completes the latest scheduled job (0 if no job is yet scheduled).

The **initial ED-schedule** $\sigma$ is the one generated by ED-H for the originally given problem instance. By modifying job release times, we may create different feasible ED-schedules by ED-H.

# 2. Brief Description of the Algorithm

**Overflow jobs and the kernels**

A job $o$ in and ED-schedule $S$ that realizes the maximal lateness, i.e., one with $f_S(o) = \max\{f(j) \mid 1 \leq j \leq n\}$ is an **overflow job**.

A **kernel** is a maximal job sequence/set in $S$ ending with an overflow job $o$ such that no job from this sequence has a due-date more than $d_o$ (if there are several successively scheduled overflow jobs then $o$ is the latest one).

**Observation**

*An ED-schedule $S$ is optimal if it contains a kernel with its earliest scheduled job starting at its release time.*

Proof. Reordering kernel jobs cannot reduce the lateness.

**Emerging jobs**

Otherwise, the earliest scheduled job of every kernel $K$ is immediately preceded and is delayed by a job $e$ with $d_e > d_o$.

Such a job is an **emerging job** for $K$, and the latest scheduled one the **delaying** emerging job.

Job $j$ scheduled after $K$ as a **passive emerging job** for $K$ if $d_j > d_o$ and $r_j < r(K)$.

**Activating an emerging job**

If we remove (reschedule later) a (non-passive) emerging job then the kernel jobs might be restarted earlier reducing in this way $L_{\max}$.

In this way, to restart the kernel jobs earlier, we **activate** an emerging job $e$ for $K$, *that is*, we force it and all passive emerging jobs to be rescheduled after $K$ by increasing their release times to a sufficiently large magnitude (the latter jobs also are said to be activated for $K$).

Then, when ED-H is again applied, neither job $e$ nor any passive emerging job will surpass any kernel job and hence the earliest job in $K$ *will start* at $r(K)$.

# 3. Binary Search Procedure

**Immediate Bounds**

Consider an (incomplete) ED-schedule $\sigma^{**}$ in which the delay job of every $K \in \mathcal{K}$ is just omitted, and let $f'(i)$ be the new (reduced) value of the lateness of each kernel job $i$ in $\sigma^{**}$. Since every $K$ is (re)started at time $r(K)$ in $\sigma^{**}$, $L(K) = \max_{i \in K}\{f'(i)\}$ is a **lower bound** on the value of the optimal schedule.

For any feasible $S$, $f(S) \geq L^* = \max_{\kappa}\{L(K_\kappa)\}$ is a **stronger lower bound**.

Furthermore, if $\delta(K) = L^* - L(K)$, then in any feasible $S$ we may allow the delay of $\delta(K) \geq 0$ without increasing the current maximal lateness, for every $K$.

# 3. Binary Search Procedure

We shall refer to the interval before each $K \in \mathcal{K}_\delta$ as the *bin* defined by $K$ and denote it by $B_K$.

$\delta$-**balanced schedule** $S(\delta)$

In an optimal schedule $S_{opt}$, either each kernel $K$ starts no later than at time $r(K) + \delta(K)$ or $K$ is to be delayed by some $\delta$, $0 \le \delta \le \Delta$, where $\Delta = f(o) - L^*$.

If the earliest job of every $K$ starts no later than at time $r(K) + \delta(K) + \delta$ then $f(i) \le L^* + \delta$, for any $i \in K$. More generally, we call a feasible schedule $S(\delta)$ with $f(S(\delta)) \le L^* + \delta$ $\delta$-**balanced** (we may note that $\sigma = S(\Delta)$).

$L^* + \delta$ is our $\delta$-**boundary**; job $j$ **surpasses** the $\delta$-boundary if $f(j) > L^* + \delta$.

**Does there exist $S(\delta)$?**

As a result of a simple preprocessing, we may guarantee that *no job from $K$ will surpass* the $\delta$-boundary when ED-H with the above restriction is again applied.

However, there may arise a non-kernel job that surpasses the $\delta$-boundary: we wish to find out if there exists $S(\delta)$.

At the first iteration of the binary search procedure, we use $\sigma = S(\Delta)$, $\delta = \Delta$.

The next value for $\delta$ is 0; if there exists no $S(0)$ then the next value of $\delta$ is $[\Delta/2]$. So $\delta$ is derived from the interval $[0, \Delta]$, whereas the change from larger to smaller value of $\delta$ is carried out if a $\delta$-balanced schedule for the current $\delta$ was successfully created; otherwise, $\delta$ is increased respectively on the next iteration.

# 3. Binary Search Procedure

**Observation**

*$S(\delta)$ with minimal possible $\delta$ is optimal.*

$1|r_j|L_{\max}$ is already solved given that we have a procedure that either constructs a $S(\delta)$ or asserts that it does not exist.

As $\Delta < p_{\max}$, the number of iterations for the binary search procedure is bounded by $\log p_{\max}$.

**Instance of alternative (b1)**

If ED-H with the restrictions above has succeeded to construct a complete schedule so that no bin job has surpassed the $\delta$-boundary, then this schedule is $S(\delta)$.

Otherwise, let $\mathcal{K}_\delta$ be the set of kernels corresponding to $\delta$, and let $K$ was the latest scheduled kernel from $\mathcal{K}_\delta$ when there has occurred (a non-kernel job) $j$ surpassing the $\delta$-boundary.

If $j$ is a former emerging job (one activated for $K$ or/and some preceding kernel) then we will say that an **instance of alternative (b1)** (IA(b1)) with job $j$ occurs.

# 4. Seeking after $S(\delta)$: Procedure $SEEK(S(\delta))$

**Defining new kernels**

If job $j$ above is not a former emerging job, then an activated (former emerging) job must be pushing $j$. If among such jobs there is an emerging job for $j$, let $e$ be the latest scheduled one.

If $e$ was included before $K$, then the jobs from $K$ together with $j$ and all jobs that were included after $e$ (before $j$ has occurred) define a new kernel, also denoted by $K$.

If $e$ was included after $K$, then the sequence of jobs in between $e$ and $j$ (including $j$) forms a new kernel $K'$ for the current $\delta$. We update the current $\mathcal{K}_\delta$ correspondingly.

**Instance of alternative (b2)**

If no new kernel can be defined, i.e., there is no $e$, let $i$ be an activated (former emerging) job pushing $j$. Then an **instance of alternative (b2)** (IA(b2)) with job $i$ is said to occur.

It follows that if there has arisen a non-kernel job surpassing the $\delta$-boundary, then there must be occurring an IA(b1/b2).

Hence, if no IA(b1/b2) occurs then we already have a correct answer (for the general problem $1|r_j|L_{\max}$). Otherwise, we need to describe how we rearrange non-kernel jobs for an IA(b1/b2).

In the rest assume IA(b1/b2) with job $j$ occurs.

# 4. Seeking after $S(\delta)$: Procedure $SEEK(S(\delta))$

At least one passive emerging job $q$ for $K$ is to be rescheduled before $K$. This will not be possible (in $S(\delta)$), unless some job $s$ scheduled in $B_K$ or some earlier bin is rescheduled after $K$ (if this were possible, ED-H would include $q$ in $B_K$).

We call job $s$ pushing $q$ a **substitution job** for $q$ if it is an emerging job for $K$ ($s$ is from $B_K$ or some earlier bin).

$\Rightarrow$ $SUBST(K, \delta)$ - set of **substitution jobs** for $K$

**Observation**

*Suppose there occurs an IA(b1/b2) behind kernel $K$. Then there exists no $S(\delta)$ if there arises no valid gap for none of the passive emerging jobs for $K$ subject to some substitution jobs.*

# 4. Seeking after $S(\delta)$: Procedure $SEEK(S(\delta))$

Thus all we need to do is to activate substitution jobs for $K$ in a proper fashion, whenever IA(b1/b2) occurs.

**Complexity of $SEEK(S(\delta))$**

For fixed $p^*$, the number of non-congruent subsets $S \subseteq SUBST(K, \delta)$ is equal to the number $P(p^*)$ of representations of $p^*$ as a sum of positive integers (without considering the order), where $P(p^*)$ is the **partition function**:

$$P(p^*) \approx \frac{\exp(\pi\sqrt{2p^*/3})}{4p^*\sqrt{3}}$$

$p^* < p^{max} \Rightarrow$ total number of non-congruent subsets is

$$O(p_{max}P(p^*)) = O(1)$$

# 4. Seeking after $S(\delta)$: Procedure $SEEK(S(\delta))$

$\Rightarrow$ complexity of procedure $SEEK(S(\delta))$ : $O(n^2 \log n)$

**Theorem**

The binary search procedure finds an optimal schedule in time $O(n^2 \log n \log p_{\max})$ (or $O(d_{max} n \log n \log p_{\max})$.

**Remark:** Problem $1|p_j \in \{p, 2p, 3p, \ldots, kp\}, r_j|L_{\max}$ is the maximal polynomially solvable case of problem $1|r_j|L_{\max}$.