Approximation Schemes for the Min-Max Starting Time Problem ⋆

Leah Epstein[1] and Tamir Tassa[2]

[1] School of Computer Science, The Interdisciplinary Center, P.O.B 167, 46150 Herzliya, Israel. lea@idc.ac.il.
[2] Department of Applied Mathematics, Tel-Aviv University, Ramat Aviv, Tel Aviv, Israel. tassa@post.tau.ac.il.

Abstract. We consider the off-line scheduling problem of minimizing the maximal starting time. The input to this problem is a sequence of n jobs and m identical machines. The goal is to assign the jobs to the machines so that the first time in which all jobs have already started their processing is minimized, under the restriction that the processing of the jobs on any given machine must respect their original order. Our main result is a polynomial time approximation scheme for this problem in the case where m is considered as part of the input. As the input to this problem is a sequence of jobs, rather than a set of jobs where the order is insignificant, we present techniques that are designed to handle ordering constraints. Those techniques are combined with common techniques of assignment problems in order to yield a polynomial time approximation scheme.

1 Introduction

Consider the following scenario: a computer operator needs to run an ordered sequence of n jobs having known processing times. The operator may assign the jobs to one of m identical and parallel machines, where each job must be executed continuously and completely on one machine. After the jobs have been assigned to the m machines, they must run on each machine according to their original order. The operator needs to verify that each job has started running before he may go home. The goal of the operator is to minimize the time when he could go home. Hence, he aims at finding an assignment that minimizes the maximal starting time, namely, the maximum over all machines of the time in which the last job assigned to that machine starts running.

The above scenario may be generalized to any setting where n clients are registered in a service center having m servers; e.g., patients in a clinic where there are m doctors or drivers that bring their car to a garage where there are m service stations. Each client has a priority that could be determined, for example, by the time the client has called in to make an appointment. The clients need to be assigned to servers according to their estimated service time so that the time in which the last client starts being served is minimized. That could be the

⋆ Research supported in part by the Israel Science Foundation (grant no. 250/01).

relevant cost function if the waiting clients need to be attended (for example, if the receptionist in the clinic needs to stay and watch the patients that are still waiting, he would aim at assigning the clients to doctors so that he could leave as early as possible). An assignment constraint is that a client with a lower priority cannot be served before a client of a higher priority by the same server.

We note that a similar off-line problem with ordering constraint was presented in [10]. There, the n jobs were to be assigned to $m = 2$ machines and be processed on each machine according to their original order so that the sum of all completion times is minimized.

Many scheduling problems have been studied both in off-line [3, 4, 6, 12, 7] and on-line environments. In on-line environments, the jobs usually arrive in a sequence. In off-line environments, however, the input is usually a *set* of jobs, where the order of jobs is insignificant. The min-max starting time problem was first introduced as an on-line problem in [2]. Here, we study the off-line version of the problem where the input is still viewed as a *sequence* of jobs and order does matter. Note that if we disregard order in this problem, namely, if we view the sequence of jobs as merely a set of jobs, then it becomes equivalent to the standard makespan problem. Indeed, the non-ordered min-max starting time problem may be reduced to the makespan problem if we remove the m largest jobs and then solve a makespan minimization problem for the remaining $n - m$ jobs. On the other hand, the makespan problem may be reduced to the non-ordered min-max starting time problem by adding m additional jobs of very large size to the given set of jobs. In view of the above, the non-ordered version of our problem is strongly NP-hard, and, consequently, so is the ordered version which we study herein.

Due to the strong NP-hardness of the problem, Polynomial Time Approximation Schemes (PTAS) are sought. Such schemes aim at finding a solution whose target function value is larger than the optimal value by a factor of no more than $(1 + \varepsilon)$, where $\varepsilon > 0$ is an arbitrarily small parameter. The run-time of such schemes depends polynomially on n and m, but it depends exponentially on $1/\varepsilon$. In case m is viewed as a constant, one usually aims at finding a Fully Polynomial Time Approximation Scheme (FPTAS), the run time of which depends polynomially on n and $1/\varepsilon$, but is exponential in the constant m. For example, PTAS for the classical makespan problem were designed by Hochbaum and Shmoys [6, 5], while an FPTAS to that problem was presented by Graham in [4] and later by Sahni in [12].

Regarding the on-line version of our problem, an algorithm of competitive approximation ratio 12 is presented in [2]. It is also shown there that a greedy algorithm that performs list scheduling on the sequence [3] has a competitive ratio of $\Theta(\log m)$.

In this paper we design two approximation schemes: a PTAS for the case where m is part of the input and an FPTAS for the case where m is constant (the FPTAS is omitted due to space limitations). In doing so, we employ several techniques that appeared in previous studies: rounding the job sizes [6], distinguishing between small and large jobs and preprocessing the small jobs [5, 1], and enumeration. However, the handling of sequences of jobs, rather than sets thereof, is significantly more delicate: the small jobs require a different handling when order matters and one must keep track of the index of the final job that is scheduled to run on each machine in order to guarantee the legality of the assignment. The techniques presented here address those issues.

Note that there exists some work on off-line scheduling of sequences. One such example is the precedence constraints problem. The jobs in that problem are given as the vertices of a directed acyclic graph. An edge (a, b) means that job a must be completed before job b is started. The goal is to minimize the makespan. It was proved in [9] that it is hard to achieve for this problem an approximation factor smaller than 4/3, unless $P = NP$. Schuurman and Woeginger [13] mention this problem as the first among ten main open problems in approximation of scheduling problems. Specifically they ask whether the problem can be approximated up to a factor smaller than $2-1/m$, an approximation factor that is achieved by the on-line List Scheduling algorithm [3].

We proceed by presenting a formal definition of the problem. In the **Minmax Starting Time Problem** one is given a sequence of n jobs with processing times p_i, $1 \le i \le n$, and $m < n$ identical machines, M_k, $1 \le k \le m$. An assignment of the jobs to the machines is a function $A : \{1, \ldots, n\} \to \{1, \ldots, m\}$. The subset of jobs that are scheduled to run on machine M_k is $\{p_i : i \in A^{-1}(k)\}$ where $A^{-1}(k) = \{i : A(i) = k\}$. The jobs are processed on each machine in the order that corresponds to their index. Hence, the index of the last job to run on machine M_k is given by $f_k = \max\{A^{-1}(k)\}$. Such jobs are referred to as the *final jobs* for assignment A. The time in which the final job on machine M_k will start running is given by $F_k = \sum_{i\in A^{-1}(k)\setminus\{f_k\}} p_i$. The goal is to find an assignment A such that the time in which the last final job starts running is minimized. Namely, we look for an assignment for which $T(A) := \max_{1\le k\le m} F_k$ is minimized.

The discussion of this problem becomes easier if we assume that the given problem instance is collision-free in the sense that all processing times are different, i.e., $p_i \ne p_j$ for $i \ne j$. Such an assumption also allows us to identify a job with its processing time. In order to rely upon that assumption, we show how to translate a problem instance having collisions into another one that is collision-free and has a close target function value. To that end, define $\Delta = \min_{1\le i,j\le n}\{p_i, |p_i - p_j|\}$. If $\mathcal{C} = \{p_{i_\ell}\}_{1\le \ell\le c}$ is a cluster of colliding jobs, $p_{i_1} = \ldots = p_{i_c}$, we replace that cluster of jobs with $\check{\mathcal{C}} = \{\check{p}_{i_\ell}\}_{1\le\ell\le c}$ where $\check{p}_{i_\ell} = p_{i_\ell} + (\ell-1)\cdot\frac{\varepsilon\Delta}{n^2}$, where $0 < \varepsilon \le 1$. By the definition of Δ, it is clear that after applying this procedure to all clusters among $\{p_1, \ldots, p_n\}$, we get a new sequence of perturbed jobs $\{\check{p}_1, \ldots, \check{p}_n\}$ that is collision-free. Moreover, as $0 \le \check{p}_i - p_i < \frac{\varepsilon\Delta}{n}$, we conclude that the value of the target function may increase in wake of such a perturbation by no more than $\varepsilon\Delta$. Since it is clear that all assignments satisfy $T(A) \ge \Delta$ (recall that $n > m$), we may state the following:

Proposition 1. *Let A be an assignment of $\{p_1, \ldots, p_n\}$, the original sequence of jobs, and let $\check{A}$ be the corresponding assignment of the perturbed sequence of jobs, $\{\check{p}_1, \ldots, \check{p}_n\}$. Then* $T(A) \le T(\check{A}) \le (1+\varepsilon)\cdot T(A) \quad , \quad 0 < \varepsilon \le 1$.

In view of the above, we assume henceforth that all processing times are different and we maintain the original notation (i.e., p_i and not $\check{p}_i$).

In the subsequent section we describe a PTAS for the case where m is part of the input. The FPTAS the case where m is constant appears in the full version of this paper.

2 A Polynomial Time Approximation Scheme

To facilitate the presentation of our PTAS, we introduce the following notations:

1. Given an assignment A, F_A denotes the subset of indices of final jobs, $F_A = \{f_k : 1 \le k \le m\}$.
2. F_A^c denotes the complement subset of indices of non-final jobs.
3. p^{lnf} denotes the size of the largest non-final job, $p^{\mathrm{lnf}} = \max\{p_i : i \in F_A^c\}$.
4. $J^{m+1} = \{p_{j_1}, \ldots, p_{j_{m+1}}\}$ denotes the subset of the $m+1$ largest jobs.

The above definitions imply that one of the jobs in J^{m+1} has processing time p^{lnf}.

The main loop in the algorithm goes over all jobs $p \in J^{m+1}$ and considers assignments in which $p^{\mathrm{lnf}} = p$. Obviously, by doing so, we cover all possible assignments. For a given value of p^{lnf}, we may conclude that all corresponding assignments A satisfy $T(A) \ge p^{\mathrm{lnf}}$. In view of that, we decompose the set of jobs $\{p_1, \ldots, p_n\}$ to small and large jobs as follows:

$$\mathcal{S} = \left\{p_i : p_i \le \varepsilon p^{\mathrm{lnf}}\right\} \quad , \quad \mathcal{L} = \left\{p_i : p_i > \varepsilon p^{\mathrm{lnf}}\right\} \ . \tag{1}$$

Discretizing the large jobs
Given a lower bound p^{lnf}, we discretize the processing times of all large jobs that are smaller than p^{lnf}. Namely, we treat all jobs p_i for which $\varepsilon p^{\mathrm{lnf}} < p_i < p^{\mathrm{lnf}}$. To this end, we define a geometric mesh on the interval $[\varepsilon, 1]$,

$$\xi_0 = \varepsilon \ \ ; \quad \xi_i = (1+\varepsilon)\xi_{i-1} \ \ , \quad 1 \le i \le q \ \ ; \quad q := \left\lceil \frac{-\lg \varepsilon}{\lg(1+\varepsilon)} \right\rceil \ , \tag{2}$$

and then, for all $p \in \mathcal{L}$,

$$p' = \begin{cases} p^{\mathrm{lnf}} \cdot \mathcal{H}\left(p/p^{\mathrm{lnf}}\right) & \text{if } \ p < p^{\mathrm{lnf}} \\ p & \text{otherwise} \end{cases} \ , \tag{3}$$

where $\mathcal{H}$ replaces its argument by the left end point of the interval $[\xi_{i-1}, \xi_i)$ where it lies. Note that if $p < p^{\mathrm{lnf}}$ then p' belongs to a finite set of size q, $\Omega = \{\xi_0 \cdot p^{\mathrm{lnf}}, \ldots, \xi_{q-1} \cdot p^{\mathrm{lnf}}\}$. With this definition, we state the following straightforward proposition.

Proposition 2. *For a given $0 < \varepsilon \le 1$ and $p^{lnf} \in J^{m+1}$, let $\mathcal{S}$ and $\mathcal{L}$ be as in (1), and*

$$\mathcal{L}' = \{p' : p \in \mathcal{L}\} \ . \tag{4}$$

Let A be an assignment of the original jobs, $\mathcal{S} \cup \mathcal{L}$, and let A' be the corresponding assignment of the modified jobs $\mathcal{S} \cup \mathcal{L}'$. Then $T(A') \le T(A) \le (1+\varepsilon) \cdot T(A')$.

Preprocessing the small jobs
Denote the subsequence of indices of small jobs by $i_1 < i_2 < \ldots < i_b$, where $b = |\mathcal{S}|$. We describe below how to modify the small jobs into another set of jobs, the size of which is either $\varepsilon p^{\mathrm{lnf}}$ or 0. To that end, let σ_r denote the sum of the first r small jobs,

$$\sigma_r = \sum_{k=1}^{r} p_{i_k} \qquad 0 \le r \le b \ . \tag{5}$$

The modified small jobs are defined as follows:

$$\hat{\mathcal{S}} = \{\hat{p}_{i_1}, \ldots, \hat{p}_{i_b}\} \quad \text{where} \quad \hat{p}_{i_r} = \begin{cases} \varepsilon p^{\mathrm{lnf}} & \text{if} \quad \left\lceil \sigma_r / \varepsilon p^{\mathrm{lnf}} \right\rceil > \left\lceil \sigma_{r-1} / \varepsilon p^{\mathrm{lnf}} \right\rceil \\ 0 & \text{otherwise} \end{cases} \ . \tag{6}$$

Proposition 3. *Let A be an assignment of the original jobs, $\mathcal{S} \cup \mathcal{L}$, with target value $T(A)$. Then for every $0 < \varepsilon \le 1$ there exists a legal assignment, $\hat{A}$, of the modified jobs, $\hat{\mathcal{S}} \cup \mathcal{L}$, such that*

$$T(\hat{A}) \le T(A) + 2\varepsilon p^{lnf} \ . \tag{7}$$

Notation agreement. Each assignment $A : \{1, \ldots, n\} \to \{1, \ldots, m\}$ induces a unique function from the set of processing times $\{p_1, \ldots, p_n\}$ to the set of machines $\{M_1, \ldots, M_m\}$. In view of our assumption of distinct processing times, the converse holds as well. In order to avoid cumbersome notations, we identify between those two equivalent functions and use the same letter to denote them both. For example, notations such as $A(i)$ or $A^{-1}(k)$ correspond to the index-index interpretation, while $A : \mathcal{S} \cup \mathcal{L} \to \{M_1, \ldots, M_m\}$ corresponds to the job-machine interpretation.

Proof. *The order of the machines.* Consider the subset of indices of final jobs, F_A. Without loss of generality, we assume that they are monotonically increasing, i.e., $f_1 < f_2 < \ldots < f_m$.
Description of $\hat{A}$. Define the prefix subsets and sums of A,

$$\mathcal{A}_k = \{p_i : p_i \in \mathcal{S} \ \text{ and } \ A(i) \le k\} \quad , \quad \tau_k = \sum_{p_i \in \mathcal{A}_k} p_i \qquad 0 \le k \le m \ . \tag{8}$$

Namely, $\mathcal{A}_k$ denotes the prefix subset of small jobs that are assigned by A to one of the first k machines, while τ_k denotes the corresponding prefix sum. Next, we define

$$r(k) = \min \left\{ r : \left\lceil \frac{\tau_k}{\varepsilon p^{\mathrm{lnf}}} \right\rceil = \left\lceil \frac{\sigma_r}{\varepsilon p^{\mathrm{lnf}}} \right\rceil \right\} \qquad 0 \le k < m \quad \text{and} \quad r(m) = b \ , \tag{9}$$

where σ_r is given in (5) and b is the number of small jobs. Obviously,

$$0 = r(0) \le r(1) \le \ldots \le r(m) = b \ . \tag{10}$$

Next, we define the assignment $\hat{A} : \hat{\mathcal{S}} \cup \mathcal{L} \to \{M_1, \ldots, M_m\}$ as follows:

$$\hat{A}(p) = A(p) \qquad \forall p \in \mathcal{L} \ , \tag{11}$$

namely, it coincides with A for the large jobs, while for the modified small jobs

$$\hat{A}(\hat{p}_{i_r}) = k \quad \text{for all } 1 \le r \le b \ \text{ such that } \ r(k-1)+1 \le r \le r(k). \tag{12}$$

Note that (10) implies that (12) defines $\hat{A}$ for all jobs in $\hat{\mathcal{S}}$. Finally, for each of the machines, we rearrange the jobs that were assigned to it in (11) and (12) in an increasing order according to their index.
The prefix sets and sums of $\hat{A}$. Similarly to (8), we define the prefix subsets and sums for the modified small jobs $\hat{\mathcal{S}}$ and assignment $\hat{A}$:

$$\hat{\mathcal{A}}_k = \{\hat{p}_i : \hat{p}_i \in \hat{\mathcal{S}} \ \text{ and } \ \hat{A}(i) \le k\} \quad , \quad \hat{\tau}_k = \sum_{\hat{p}_i \in \hat{\mathcal{A}}_k} \hat{p}_i \qquad 0 \le k \le m \ . \tag{13}$$

Denote the largest index of a job in $\mathcal{A}_k$ by $i_{t(k)}$ and the largest index of a job in $\hat{\mathcal{A}}_k$ by $i_{\hat{t}(k)}$. As (12) implies that $\hat{t}(k) = r(k)$ while, by (9), $r(k) \le t(k)$, we conclude that $\hat{t}(k) \le t(k)$.
The Min-Max start time of $\hat{A}$. Given an assignment $A : \mathcal{S} \cup \mathcal{L} \to \{M_1, \ldots, M_m\}$ we defined an assignment $\hat{A} : \hat{\mathcal{S}} \cup \mathcal{L} \to \{M_1, \ldots, M_m\}$. Let M_k be an arbitrary machine, $1 \le k \le m$. The final job in that machine, corresponding to the first assignment, is p_{f_k}, and its start time is $\theta_k = \ell_k + \tau_k - \tau_{k-1} - p_{f_k}$, where ℓ_k is the sum of large jobs assigned to M_k. Similarly, letting $\hat{p}_{\hat{f}_k}$ denote the final job in that machine as dictated by the second assignment, its start time is $\hat{\theta}_k = \ell_k + \hat{\tau}_k - \hat{\tau}_{k-1} - \hat{p}_{\hat{f}_k}$. In order to prove (7) we show that

$$\hat{\theta}_k \le \theta_k + 2\varepsilon p^{\mathrm{lnf}} \ . \tag{14}$$

First, we observe that in view of the definition of the modified small jobs, (6), and $r(k)$, (9), $\hat{\tau}_k = \varepsilon p^{\mathrm{lnf}} \cdot \left\lceil \tau_k / \varepsilon p^{\mathrm{lnf}} \right\rceil$. Consequently,

$$\hat{\tau}_k - \varepsilon p^{\mathrm{lnf}} < \tau_k \le \hat{\tau}_k \ . \tag{15}$$

Therefore, it remains to show only that

$$p_{f_k} - \hat{p}_{\hat{f}_k} \le \varepsilon p^{\mathrm{lnf}} \tag{16}$$

in order to establish (14). If $p_{f_k} \in \mathcal{S}$ then (16) is immediate since then

$$p_{f_k} - \hat{p}_{\hat{f}_k} \le p_{f_k} \le \varepsilon p^{\mathrm{lnf}} \ .$$

Hence, we concentrate on the more interesting case where $p_{f_k} \in \mathcal{L}$. In this case the job p_{f_k} is assigned to M_k also by $\hat{A}$. We claim that it is in fact also the final job in that latter assignment, namely, $\hat{p}_{\hat{f}_k} = p_{f_k}$. This may be seen as follows:

∘ The indices of the modified small jobs in M_k are bounded by $i_{\hat{t}(k)}$.

∘ As shown earlier, $i_{\hat{t}(k)} \le i_{t(k)}$.

∘ $i_{t(k)} < f_k$ since the machines are ordered in an increasing order of f_k and, therefore, the largest index of a small job that is assigned to one of the first k machines, $i_{t(k)}$, is smaller than the index of the final (large) job on the kth machine, f_k.

∘ The above arguments imply that the indices of the modified small jobs in M_k cannot exceed f_k.

∘ Hence, the rearrangement of large and modified small jobs that were assigned to M_k by $\hat{A}$ keeps job number f_k as the final job on that machine.

This proves $\hat{p}_{\hat{f}_k} = p_{f_k}$ and, consequently, (16). □

Proposition 4. *Let $\hat{A}$ be an assignment of the modified jobs, $\hat{\mathcal{S}} \cup \mathcal{L}$, with target value $T(\hat{A})$. Then there exists a legal assignment, A, of the original jobs, $\mathcal{S} \cup \mathcal{L}$, such that*

$$T(A) \le T(\hat{A}) + 2\varepsilon p^{lnf} . \qquad (17)$$

Remark. Proposition 4 complements Proposition 3 as it deals with the inverse reduction, from a solution in terms of the modified jobs to a solution in terms of the original jobs. Hence the similarity in the proofs of the two complementary propositions. Note, however, that the direction treated in Proposition 4 is the algorithmically important one (as opposed to the direction treated in Proposition 3 that is needed only for the error estimate).

Proof. Due to the similarity of this proof to the previous one, we focus on the constructive part of the proof. We first assume that the indices of the final jobs, as dictated by $\hat{A}$, are monotonically increasing, namely, $\hat{f}_1 < \hat{f}_2 < \ldots < \hat{f}_m$. Then, we consider the prefix subsets and sums of $\hat{A}$,

$$\hat{\mathcal{A}}_k = \{\hat{p}_i : \hat{p}_i \in \hat{\mathcal{S}} \text{ and } \hat{A}(i) \le k\} \quad , \quad \hat{\tau}_k = \sum_{\hat{p}_i \in \hat{\mathcal{A}}_k} \hat{p}_i \qquad 0 \le k \le m \ , \qquad (18)$$

and define

$$r(k) = \min\left\{ r : \frac{\hat{\tau}_k}{\varepsilon p^{\mathrm{lnf}}} = \left\lceil \frac{\sigma_r}{\varepsilon p^{\mathrm{lnf}}} \right\rceil \right\} \qquad 0 \le k < m \quad \text{and} \quad r(m) = b \ , \qquad (19)$$

where σ_r and b are as before. Note that $\hat{\tau}_k$ is always an integral multiple of $\varepsilon p^{\mathrm{lnf}}$ and $0 = r(0) \le r(1) \le \ldots \le r(m) = b$. Finally, we define the assignment $A : \mathcal{S} \cup \mathcal{L} \to \{M_1, \ldots, M_m\}$: A coincides with $\hat{A}$ on $\mathcal{L}$; as for the small jobs $\mathcal{S}$, A is defined by

$$A(p_{i_r}) = k \quad \text{for all } 1 \le r \le b \ \text{ such that } \ r(k-1) + 1 \le r \le r(k) \ . \qquad (20)$$

The jobs in each machine – large and small – are then sorted according to their index.

The proof of estimate (17) is analogous to the proof of (7) in Proposition 3. In fact, if we take the proof of (7) and swap there between every hat-notated symbol with its non-hat counterpart (i.e., $A \leftrightarrow \hat{A}$, $p_i \leftrightarrow \hat{p}_i$, $i_t(k) \leftrightarrow i_{\hat{t}(k)}$ etc.), we get the corresponding proof of (17). □

The algorithm
In the previous subsections we described two modifications of the given jobs that have a small effect on the value of the target function, Propositions 2–4. The first modification translated the values of the large jobs that are smaller than p^{lnf} into values from a finite set Ω. The second modification replaced the set of small jobs with modified small jobs of size either 0 or $\xi_0 \cdot p^{\mathrm{lnf}} = \varepsilon \cdot p^{\mathrm{lnf}}$. Hence, after applying those two modifications, we are left with job sizes p where either $p \geq p^{\mathrm{lnf}}$ or $p \in \Omega \cup \{0\}$. After those preliminaries, we are ready to describe our algorithm.

<u>The Main Loop</u>

1. Identify the subsequence of $m+1$ largest jobs, $J^{m+1} = \{p_{j_1}, \ldots, p_{j_{m+1}}\}$, $p_{j_1} > p_{j_2} > \ldots > p_{j_{m+1}}$.
2. For $r = 1, \ldots, m+1$ do:
   (a) Set $p^{\mathrm{lnf}} = p_{j_r}$.
   (b) Identify the subsets of small and large jobs, (1).
   (c) Discretize the large jobs, $\mathcal{L} \mapsto \mathcal{L}'$, according to (3)+(4).
   (d) Preprocess the small jobs, $\mathcal{S} \mapsto \hat{\mathcal{S}}$.
   (e) Solve the problem in an optimal manner for the modified sequence of jobs $\hat{\mathcal{S}} \cup \mathcal{L}'$, using the core algorithm that is described below.
   (f) Record the optimal assignment, A^r, and its value, $T(A^r)$.
3. Select the assignment A^r for which $T(A^r)$ is minimal.
4. Translate A^r to an assignment A in terms of the original jobs, using Propositions 2 and 4.
5. Output assignment A.

<u>The Core Algorithm</u>

The core algorithm receives:
(1) a value of r, $1 \leq r \leq m+1$;
(2) a guess for the largest non-final job, $p^{\mathrm{lnf}} = p_{j_r}$;
(3) a sequence Φ of $r-1$ jobs $p_{j_1}, \ldots, p_{j_{r-1}}$ that are final jobs;
(4) a sequence Γ of $n-r$ jobs, the size of which belongs to $\Omega \cup \{0\}$.

It should be noted that the given choice of p^{lnf} splits the remaining $n-1$ jobs from $\hat{\mathcal{S}} \cup \mathcal{L}'$ into two subsequences of jobs: those that are larger than p^{lnf} (i.e., the $r-1$ jobs in Φ that are non-modified jobs) and those that are smaller than p^{lnf} (i.e., the $n-r$ jobs in Γ that are modified jobs, either large or small).

<u>Step 1: Filling in the remaining final jobs.</u>

The input to this algorithm dictates the identity of $r-1$ final jobs, Φ. We need to select the additional $m-r+1$ final jobs out of the $n-r$ jobs in Γ. We omit the proof of the next lemma.

Lemma 1. *The essential number of selections of the remaining $m-r+1$ final jobs out of the $n-r$ jobs in Γ is polynomial in m.*

<u>Step 2: A makespan problem with constraints.</u>

Assume that we selected in Step 1 the remaining $m-r+1$ final jobs and let $F_A = \{f_k : 1 \leq k \leq m\}$ be the resulting selection of final jobs, f_k being the index of the selected final job in M_k. Without loss of generality, we assume that $f_1 < f_2 < \ldots < f_m$. Given this selection, an optimal solution may be found by solving the makespan problem on the remaining $n - m$ jobs with the constraint that a job p_i may be assigned only to machines M_k where $i < f_k$.

Next, define Γ_η^* to be the subset of non-final jobs from Γ_η, $-1 \leq \eta \leq q-1$, and let $\Gamma_q^* = \{p^{\text{lnf}}\}$. For each $0 \leq k \leq m$ and $-1 \leq \eta \leq q$, we let z_η^k be the number of jobs from Γ_η^* whose index is less than f_k. Finally, we define for each $0 \leq k \leq m$ the vector $\mathbf{z}^k = (z_{-1}^k, \ldots, z_q^k)$ that describes the subset of non-final jobs that could be assigned to at least one of the first k machines. In particular, we see that $\mathbf{0} = \mathbf{z}^0 \leq \mathbf{z}^1 \leq \ldots \leq \mathbf{z}^m$ where the inequality sign is to be understood component-wise and $\mathbf{z}^m$ describes the entire set of non-final jobs, F_A^c. In addition, we let $\mathcal{P}(\mathbf{z}^k) = \{\mathbf{z} \in \mathcal{N}^{q+2} : \mathbf{0} \leq \mathbf{z} \leq \mathbf{z}^k\}$ be the set of all sub-vectors of $\mathbf{z}^k$.

Step 3: Shortest path in a graph.

Next, we describe all possible assignments of those jobs to the m machines using a layered graph $G = (V, E)$. The set of vertices is composed of $m+1$ layers, $V = \cup_{k=0}^m V_k$, where $V_k = \mathcal{P}(\mathbf{z}^k) \quad 0 \leq k < m \quad \text{and} \quad V_m = \{\mathbf{z}^m\}$. We see that the first and last layers are composed of one vertex only, $\{\mathbf{z}^0 = \mathbf{0}\}$ and $\{\mathbf{z}^m\}$ respectively, while the intermediate layers are monotonically non-decreasing in size corresponding to the non-decreasing prefix subsets $\mathcal{P}(\mathbf{z}^k)$.

The set of edges is composed of m layers, $E = \cup_{k=1}^m E_k$ where E_k is defined by $E_k = \{\ (\mathbf{u}, \mathbf{v}) \in V_{k-1} \times V_k : \mathbf{u} \leq \mathbf{v}\ \}$. It is now apparent that all possible assignments of jobs from F_A^c to the m machines are represented by all paths in G from V_0 to V_m. Assigning weights to the edges of the graph in the natural manner, $w[(\mathbf{u}, \mathbf{v})] = \sum_{\eta=0}^{q} (v_\eta - u_\eta) \cdot p^{\text{lnf}} \cdot \xi_\eta$, where $\{\xi_\eta\}_{\eta=0}^{q-1}$ are given in (2) and $\xi_q = 1$, we may define the cost of a path $(\mathbf{u}^0, \ldots, \mathbf{u}^m) \in V_0 \times \ldots \times V_m$ as $T[(\mathbf{u}^0, \ldots, \mathbf{u}^m)] = \max\{w[(\mathbf{u}_{k-1}, \mathbf{u}_k)] : 1 \leq k \leq m\}$. In view of the above, we need to find the shortest path in the graph, from the source V_0 to the sink V_m, and then translate it to an assignment in the original jobs.

Step 4: Translating the shortest path to an assignment of jobs.

For $1 \leq k \leq m$ and $-1 \leq \eta \leq q$, let u_η^k be the number of jobs of type η that were assigned by the shortest path to machine k. Then assign the u_η^k jobs of the lowest indices in Γ_η^* to machine M_k and remove those jobs from Γ_η^*. Finally, assign the final jobs of indices f_k, $1 \leq k \leq m$.

Performance estimates

We omit the proofs of the following theorems.

Theorem 1. *For a fixed $0 < \varepsilon \leq 1$, the running time of the above described algorithm is polynomial in m and n.*

Theorem 2. *Let T^o be the value of an optimal solution to the given Min-Max Starting Time Problem. Let A be the solution that the above described PTAS yields for that problem. Then for all $0 < \varepsilon \leq 1$, $T(A) \leq (1 + 35\varepsilon)T^o$.*

3 Concluding Remarks

The min-max starting time problem is closely related to the makespan minimization problem in the hierarchical model [11]. In that model, each of the given jobs may be assigned to only a suffix subset of the machines, i.e., $\{M_k, \ldots, M_m\}$ for some $1 \leq k \leq m$. An FPTAS for the makespan minimization problem in the case where m is constant is given in [7, 8]. That FPTAS may serve as a building block for an FPTAS for the min-max starting time problem. We also observe that the same techniques that were presented above to construct a PTAS for the min-max starting time problem may be used in order to construct a PTAS for the makespan minimization problem in the hierarchical model (to the best of our knowledge, such a PTAS was never presented before).

References

1. N. Alon, Y. Azar, G. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:1:55–66, 1998.
2. L. Epstein and R. van Stee. Minimizing the maximum starting time on-line. In *Proc. of the 10th Annual European Symposium on Algorithms (ESA'2002)*, pages 449–460, 2002.
3. R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
4. R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:416–429, 1969.
5. D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.
6. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the Association for Computing Machinery*, 34(1):144–162, 1987.
7. E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling non-identical processors. *Journal of the Association for Computing Machinery*, 23:317–327, 1976.
8. K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *Proceedings of the 31st annual ACM Symposium on Theory of Computing (STOC'99)*, pages 408–417, 1999.
9. J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity os scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.
10. S.R. Mehta, R. Chandrasekaran, and H. Emmons. Order-presrving allocation of jobs to two machines. *Naval Research Logistics Quarterly*, 21:846–847, 1975.
11. J. Naor, A. Bar-Noy, and A. Freund. On-line load balancing in a hierarchical server topology. In *Proc. of the 7th European Symp. on Algorithms (ESA'99)*, pages 77–88. Springer-Verlag, 1999.
12. S. Sahni. Algorithms for scheduling independent tasks. *Journal of the Association for Computing Machinery*, 23:116–127, 1976.
13. P. Schuurman and G. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2:203–213, 1999.

Acknowledgement. The authors would like to thank Yossi Azar from Tel-Aviv University for his helpful suggestions.