Approximation Schemes for the Min-Max Starting Time Problem*

Leah Epstein [†] Tamir Tassa [‡]

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 at which all jobs have already started running 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 (PTAS) 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 order constraints imposed by the sequence. Those techniques are combined with common techniques of assignment problems in order to yield a PTAS for this problem. We also show that when m is a constant, the problem admits a fully polynomial time approximation scheme. Finally, we show that the makespan problem in the linear hierarchical model may be reduced to the min-max starting time problem, thus concluding that the former problem also admits a PTAS.

Keywords: PTAS, approximation algorithms, scheduling, starting time, identical machines.

---

*A preliminary version of this paper appeared in Proc. of 28th Mathematical Foundations of Computer Science (2003)

†School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. `lea@idc.ac.il`. Research supported in part by the Israel Science Foundation, (grant no. 250/01)

‡Department of Mathematics and Computer Science, The Open University, Ramat Aviv, Tel Aviv and Department of Computer Science, Ben Gurion University, Beer Sheva, Israel. `tamirta@openu.ac.il`.

1 Introduction

Consider a clinic where n patients need to be seen by one of m identical doctors, $m < n$. The patients are ordered in a sequence according to some criterion, say, the time at which they called in to make an appointment. The receptionist needs to assign the patients to the doctors based on the expected "processing time" of those patients (i.e., the time that each of those patients is expected to be in the doctor's office). As the receptionist needs to stay in the clinic until the last patient is received by his assigned doctor, he aims at assigning the patients to the doctors so that he could leave the clinic as early as possible.

Clearly, the best thing to do is to identify the m patients of largest expected processing time, assign each one of them as the last patient of one of the m doctors, and then solve the makespan problem with respect to the remaining $n-m$ patients and m doctors. However, such a schedule may assign a patient after another patient that has a lower priority according to their position in the sequence. Therefore, while it is optimal in the sense that it minimizes the target function, it is totally unfair. The other extreme strategy is the fair one, in which the receptionist completely ignores the target function and obeys only fairness restrictions. The solution here is very simple - the receptionist will assign the patients according to their order in the sequence. This strategy, however, may keep the receptionist in the clinic for too long.

We consider here an intermediate strategy, one that obeys both fairness and efficiency considerations. In that strategy, the receptionist seeks an assignment that would minimize the time at which the last patient enters his assigned doctor, so that the patients that are assigned for the same doctor are scheduled in accord with their order in the sequence. The idea is that partial fairness, namely, fairness per doctor, as opposed to overall fairness, may decrease the value of the target function on one hand, and be sufficient on the other hand: patients examine the waiting list for their assigned doctor and would protest if they appeared on that list after a patient of a lower priority; on the other hand, patients are unaware or unsensitive to the waiting lists of other doctors or the times at which patients enter to be seen by other doctors. Hence, such an intermediate strategy makes sense. As opposed to the only-efficient or the only-fair strategies, it gives rise to an interesting model that does not coincide with any of the commonly studied models.

We proceed by presenting a formal definition of the problem. In the **Min-Max Starting Time Problem** one is given a *sequence* of n jobs with processing times p_i, $1 \leq i \leq n$, and $m < n$ identical machines, M_k, $1 \leq k \leq m$. 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 $A^{-1}(k) = \{i : A(i) = k\}$. The jobs are processed on each machine in the order that agrees with their position in the input sequence. 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 at 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 at which the last final job starts running is minimized. Namely, we look for an assignment for which

$$T(A) := \max_{1 \le k \le m} F_k \tag{1}$$

is minimized.

Many scheduling problems have been studied both in off-line [4, 5, 6, 12, 8] and on-line environments (see [14] for a survey). 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 [3]. 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.

We note that our problem is strongly NP-hard, as demonstrated by the following reduction from the makespan problem. Given an input set of n jobs to the makespan problem (problem P1 henceforth), we first turn the set into a sequence by imposing an arbitrary order; then, we add to that sequence m additional jobs of a very large size with indices $n+1$ through $n+m$. Any optimal solution of the min-max starting time for the augmented sequence of $n+m$ jobs (problem P2 henceforth) will obviously place each of the m large jobs as the last job to run on one of the m machines. Hence, the min-max starting time in such a solution coincides with the makespan of the corresponding schedule, restricted to the original n jobs. This implies that the min-max starting time in optimal solutions to P2 is bounded from below by the optimal makespan for P1. On the other hand, there exists an optimal solution for P2 in which the min-max starting time equals the optimal makespan for P1; indeed, given an optimal solution of P1, we may rearrange the jobs on each machine in accord with their arbitrary order in P2 without affecting the makespan and then add on top of them the m large jobs, one in each machine. Consequently, an optimal solution of P2, when restricted to the original n jobs, gives an optimal solution of P1.

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 multiplicative 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, 7], while an FPTAS for that problem was presented by Graham in [5] and later by Sahni in [12].

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. 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 [7, 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 novel techniques presented here address those issues.

Related work. The on-line version of our problem was studied in [3], where an algorithm of competitive approximation ratio 12 was presented. It was also shown there that a greedy algorithm that performs list scheduling on the sequence [4] has a competitive ratio of $\Theta(\log m)$.

A similar off-line problem with ordering constraints was presented in [11]. 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.

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. Lenstra and Rinnooy Kan proved that it is impossible to achieve for this problem an approximation factor smaller than $4/3$, unless $P = NP$ [10]. 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 [4].

The paper is organized as follows. In Section 2 we describe a PTAS for the min-max starting time problem. In Section 3 we show how this implies that also the makespan problem in the linear hierarchical model [2] admits a PTAS (to the best of our knowledge, a PTAS for that problem has not been presented before). In Section 4 we utilize an FPTAS for the makespan problem in the linear hierarchical model in order to construct an FPTAS for the min-max starting time problem when the number of machines, m, is constant.

We conclude the introduction with two observations. The first relates to the order of the jobs: instead of assuming that the jobs are ordered in a sequence, we may assume a more general setting where each job is associated with a priority and fairness restricts us from scheduling to the same machine one job after another job with a strictly smaller priority (if there are no two jobs with the same priority then we get the sequence model that we presented above). Such a model is suitable to cases where some jobs could have the same priority (e.g., the patients in the clinic example are ordered not by the time at which they called in to make an appointment, but according to their medical plan or seniority). It is easy to see that this more general setting may always be reduced to the case of a sequence. Indeed, we may order the jobs in a sequence according to the following rule: let J and J' be two jobs with priorities q, q' and processing times p, p', respectively. Then J will precede J' in the sequence if $q > q'$, or if $q = q'$ and $p < p'$; if $q = q'$ and $p = p'$, the relative position of J and J' in the sequence is insignificant. In other words, in case of an equal priority, we should always schedule the smaller jobs first. Clearly, if we rearrange the jobs scheduled to run on a given machine according to this convention, we do not increase the min-max starting time of the schedule.

The second observation is that we may assume that all processing times are different, i.e., $p_i \neq p_j$ for $i \neq j$. Such an assumption allows us to identify a job with its processing time and it facilitates the discussion. In order to rely upon this assumption, we show how to translate a problem instance having coinciding processing times into another one where all processing times are different and has a close target function value. To that end, define

$$\Delta = \min\{\Delta_1, \Delta_2\} \quad \text{where} \quad \Delta_1 = \min_{1 \le i \le n} p_i \ \text{ and } \ \Delta_2 = \min_{p_i \neq p_j} |p_i - p_j| \ .$$

If $\mathcal{C} = \{p_{i_\ell}\}_{1 \le \ell \le c}$ is a cluster of jobs of equal processing time, $p_{i_1} = \ldots = p_{i_c}$, we replace that cluster of jobs with

$$\check{\mathcal{C}} = \{\check{p}_{i_\ell}\}_{1 \le \ell \le c} \quad \text{where} \quad \check{p}_{i_\ell} = p_{i_\ell} + (\ell - 1) \cdot \frac{\varepsilon \Delta}{n^2}$$

and $0 < \varepsilon \le 1$. 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 $\{\check{p}_1, \ldots, \check{p}_n\}$ where all processing times are different. 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 the original sequence of jobs, $\{p_1, \ldots, p_n\}$, 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 $\check{p}_i$).

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,

$$F_A = \{f_k : 1 \le k \le m\} . \tag{2}$$

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:

$$\mathcal{S} = \left\{p_i : p_i \le \varepsilon p^{\mathrm{lnf}}\right\} \quad , \quad \mathcal{L} = \left\{p_i : p_i > \varepsilon p^{\mathrm{lnf}}\right\} . \tag{3}$$

We proceed to describe a discretization of the large jobs, Section 2.1, and then a preprocessing of the small jobs, Section 2.2. Those two operations differ from the more standard discretization and preprocessing steps that usually take place in non-ordered contexts, as they need to take into account the ordering constraints. After those two steps, we are left with a problem instance that may be solved optimally, Section 2.3.

2.1 Discretizing the large jobs

Given a value for 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 $[\varepsilon, 1]$,

$$\xi_0 = \varepsilon \ ; \quad \xi_i = (1+\varepsilon)\xi_{i-1} \ , \quad 1 \le i \le q \ ; \quad q := \left\lceil \frac{-\lg \varepsilon}{\lg(1+\varepsilon)} \right\rceil , \tag{4}$$

and let

$$\Omega = \{\xi_0 \cdot p^{\mathrm{lnf}}, \ldots, \xi_{q-1} \cdot p^{\mathrm{lnf}}\} . \tag{5}$$

Then, for all $p \in \mathcal{L}$,

$$p' = \begin{cases} \max\{q \le p : q \in \Omega\} & \text{if } p < p^{\text{lnf}} \\ p & \text{otherwise} \end{cases} . \tag{6}$$

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 (3), and*

$$\mathcal{L}' = \{p' : p \in \mathcal{L}\} . \tag{7}$$

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

$$T(A') \le T(A) \le (1+\varepsilon) \cdot T(A') . \tag{8}$$

2.2 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 $\varepsilon p^{\text{lnf}}$ or 0. To that end, let σ_r denote the sum of the first r small jobs,

$$\sigma_r = \sum_{k=1}^{r} p_{i_k} \qquad 0 \le r \le b . \tag{9}$$

The modified small jobs are defined as follows:

$$\hat{\mathcal{S}} = \{\hat{p}_{i_1}, \ldots, \hat{p}_{i_b}\} \quad \text{where} \quad \hat{p}_{i_r} = \begin{cases} \varepsilon p^{\text{lnf}} & \text{if } \left\lceil \sigma_r / \varepsilon p^{\text{lnf}} \right\rceil > \left\lceil \sigma_{r-1} / \varepsilon p^{\text{lnf}} \right\rceil \\ \\ 0 & \text{otherwise} \end{cases} . \tag{10}$$

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

$$T(\hat{A}) \le T(A) + 2\varepsilon p^{lnf} , \tag{11}$$

may be constructed in linear time.

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 $A : \mathcal{S} \cup \mathcal{L} \rightarrow \{M_1, \ldots, M_m\}$ corresponds to the job-machine interpretation.

Proof.

The order of the machines. Consider the subset of indices of final jobs, F_A, see (2). 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,

$$\mathcal{A}_k = \{p_i : p_i \in \mathcal{S} \text{ and } A(i) \leq k\} \quad , \quad \tau_k = \sum_{p_i \in \mathcal{A}_k} p_i \qquad 0 \leq k \leq m \, . \tag{12}$$

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

$$r(k) = \min\left\{ r : \left\lceil \frac{\sigma_r}{\varepsilon p^{\text{lnf}}} \right\rceil = \left\lceil \frac{\tau_k}{\varepsilon p^{\text{lnf}}} \right\rceil \right\} \qquad 0 \leq k < m \quad \text{and} \quad r(m) = b \, , \tag{13}$$

where σ_r is given in (9) and b is the number of small jobs. Obviously,

$$0 = r(0) \leq r(1) \leq \ldots \leq r(m) = b \, . \tag{14}$$

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

$$\hat{A}(p) = A(p) \qquad \forall p \in \mathcal{L} \, , \tag{15}$$

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

$$\hat{A}(\hat{p}_{i_r}) = k \quad \text{for all } 1 \leq r \leq b \text{ such that } r(k-1) + 1 \leq r \leq r(k) \tag{16}$$

(We refer the reader to Example 1 that illustrates the relation between $\hat{A}$ and A.) Note that (14) implies that (16) 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 (15) and (16) in an increasing order according to their index.

The prefix sets and sums of $\hat{A}$. Similarly to (12), we define the prefix subsets and sums for the modified small jobs $\hat{\mathcal{S}}$ and assignment $\hat{A}$:

$$\hat{\mathcal{A}}_k = \{\hat{p}_i : \hat{p}_i \in \hat{\mathcal{S}} \text{ and } \hat{A}(i) \leq k\} \quad , \quad \hat{\tau}_k = \sum_{\hat{p}_i \in \hat{\mathcal{A}}_k} \hat{p}_i \qquad 0 \leq k \leq m \, . \tag{17}$$

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 (16) implies that $\hat{t}(k) = r(k)$ while, by (13), $r(k) \leq t(k)$, we conclude that $\hat{t}(k) \leq t(k)$. (e.g., in Example 1, $t(k) = 8, 9, 9$ while $\hat{t}(k) = 2, 5, 9$, where $k = 1, 2, 3$.)

The Min-Max starting 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, $1 \leq k \leq m$. The final job on that machine, corresponding to the first assignment, is p_{f_k}, and its starting time is

$$\theta_k = \ell_k + \tau_k - \tau_{k-1} - p_{f_k} \ , \tag{18}$$

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 starting time is

$$\hat{\theta}_k = \ell_k + \hat{\tau}_k - \hat{\tau}_{k-1} - \hat{p}_{\hat{f}_k} \ . \tag{19}$$

In order to prove (11) we show that

$$\hat{\theta}_k \leq \theta_k + 2\varepsilon p^{\mathrm{lnf}} \ . \tag{20}$$

First, we observe that in view of the definition of the modified small jobs, (10), and $r(k)$, (13), $\hat{\tau}_k = \varepsilon p^{\mathrm{lnf}} \cdot \left\lceil \tau_k / \varepsilon p^{\mathrm{lnf}} \right\rceil$. Consequently,

$$\hat{\tau}_k - \varepsilon p^{\mathrm{lnf}} < \tau_k \leq \hat{\tau}_k \ . \tag{21}$$

Therefore, by (18), (19) and (21), it remains to show only that

$$p_{f_k} - \hat{p}_{\hat{f}_k} \leq \varepsilon p^{\mathrm{lnf}} \tag{22}$$

in order to establish (20). If $p_{f_k} \in \mathcal{S}$ then (22) is immediate since then

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

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,

$$\hat{p}_{\hat{f}_k} = p_{f_k} \ . \tag{23}$$

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, $i_{\hat{t}(k)} \leq i_{t(k)}$.

∘ $i_{t(k)} < f_k$ 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 (23) and, consequently, (22). □

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 for every $0 < \varepsilon \leq 1$ a legal assignment, A, of the original jobs, $\mathcal{S} \cup \mathcal{L}$, such that*

$$T(A) \leq T(\hat{A}) + 2\varepsilon p^{\mathit{lnf}} , \tag{24}$$

may be constructed in linear time.

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 $\hat{A}$,

$$\hat{\mathcal{A}}_k = \{\hat{p}_i : \hat{p}_i \in \hat{\mathcal{S}} \text{ and } \hat{A}(i) \leq k\} \quad , \quad \hat{\tau}_k = \sum_{\hat{p}_i \in \hat{\mathcal{A}}_k} \hat{p}_i \qquad 0 \leq k \leq m , \tag{25}$$

and define

$$r(k) = \min \left\{ r : \left\lceil \frac{\sigma_r}{\varepsilon p^{\mathrm{lnf}}} \right\rceil = \frac{\hat{\tau}_k}{\varepsilon p^{\mathrm{lnf}}} \right\} \qquad 0 \leq k < m \quad \text{and} \quad r(m) = b , \tag{26}$$

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

$$A(p_{i_r}) = k \quad \text{for all } 1 \leq r \leq b \text{ such that } r(k-1) + 1 \leq r \leq r(k) \tag{27}$$

(this is exemplified in Example 1 below.) The jobs in each machine – large and small – are then sorted according to their index.

The proof of estimate (24) is analogous to the proof of (11) in Proposition 3. In fact, if we take the proof of (11) 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 (24).

Proposition 3 offered an "assignment translation" from A to $\hat{A}$, while Proposition 4 offered a translation in the opposite direction. Note that the latter translation is not necessarily the inverse of the former one. This is illustrated by the following example.

Example 1. We concentrate here on the small jobs since the assignment of the large jobs remains invariant under the translations that are described in Propositions 3 and 4. For the sake of clarity, we divide all job sizes by $\varepsilon p^{\text{lnf}}$. Assume that there are 9 small jobs of sizes

$$\mathcal{S} = (p_{i_1}, \ldots, p_{i_9}) = (0.5, 0.6, 0.7, 0.8, 0.9, 0.1, 0.2, 0.3, 0.4) .$$

Then, in view of (10), the modified small jobs are

$$\hat{\mathcal{S}} = (\hat{p}_{i_1}, \ldots, \hat{p}_{i_9}) = (1, 1, 0, 1, 1, 0, 0, 1, 0) .$$

Assume $m = 3$ machines and the following assignment A:

$$\begin{array}{l} M_1 \leftarrow (p_{i_1}, p_{i_4}, p_{i_8}) = (0.5, 0.8, 0.3) \\ M_2 \leftarrow (p_{i_2}, p_{i_3}, p_{i_9}) = (0.6, 0.7, 0.4) \\ M_3 \leftarrow (p_{i_5}, p_{i_6}, p_{i_7}) = (0.9, 0.1, 0.2) \end{array} .$$

The corresponding assignment $\hat{A}$ would then be, (16),

$$\begin{array}{l} M_1 \leftarrow (\hat{p}_{i_1}, \hat{p}_{i_2}) = (1, 1) \\ M_2 \leftarrow (\hat{p}_{i_3}, \hat{p}_{i_4}, \hat{p}_{i_5}) = (0, 1, 1) \\ M_3 \leftarrow (\hat{p}_{i_6}, \hat{p}_{i_7}, \hat{p}_{i_8}, \hat{p}_{i_9}) = (0, 0, 1, 0) \end{array} .$$

Finally, translating $\hat{A}$ into an assignment $\bar{A}$ of the original jobs according to (27) we get

$$\begin{array}{l} M_1 \leftarrow (p_{i_1}, p_{i_2}) = (0.5, 0.6) \\ M_2 \leftarrow (p_{i_3}, p_{i_4}, p_{i_5}) = (0.7, 0.8, 0.9) \\ M_3 \leftarrow (p_{i_6}, p_{i_7}, p_{i_8}, p_{i_9}) = (0.1, 0.2, 0.3, 0.4) \end{array} .$$

Note that $\bar{A} \neq A$. Also, we see that $T(A) = 1.3$, $T(\hat{A}) = 1$ and $T(\bar{A}) = 1.5$.

2.3 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, (5). The second modification replaced the set of small jobs with modified small jobs of size either 0 or $\xi_0 \cdot p^{\text{lnf}} = \varepsilon \cdot p^{\text{lnf}}$. Hence, after applying those two modifications, we are left with job sizes p where either $p \geq p^{\text{lnf}}$ or $p \in \Omega \cup \{0\}$. After those preliminaries, we are ready to describe our algorithm.

THE MAIN LOOP

1. Identify the subsequence of $m+1$ largest jobs,

$$J^{m+1} = \{p_{j_1}, \ldots, p_{j_{m+1}}\} \quad , \quad p_{j_1} > p_{j_2} > \ldots > p_{j_{m+1}} \,. \tag{28}$$

2. For $r = 1, \ldots, m+1$ do:
   (a) Set $p^{\text{lnf}} = p_{j_r}$.
   (b) Identify the subsets of small and large jobs, (3).
   (c) Discretize the large jobs, $\mathcal{L} \mapsto \mathcal{L}'$, according to (6)+(7).
   (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.

THE CORE ALGORITHM

The core algorithm receives:
(1) a value of r, $1 \leq r \leq m+1$;
(2) a guess for the largest non-final job, $p^{\text{lnf}} = p_{j_r}$;
(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).

Step 1: Filling in the remaining final jobs.

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.

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 and independent of n.*

Proof. Consider the decomposition of Γ into types,

$$\Gamma = \cup_{\eta=-1}^{q-1} \Gamma_\eta \quad \text{where} \quad \Gamma_\eta = \Gamma \cap \{\xi_\eta \cdot p^{\text{lnf}}\} \;\; 0 \leq \eta < q \quad \text{and} \quad \Gamma_{-1} = \Gamma \cap \{0\} \, .$$

Let A be an assignment of the given modified jobs that minimizes $T(A)$. Let B_η, $-1 \leq \eta \leq q-1$, be the set of final jobs of type η, i.e., $B_\eta = \Gamma_\eta \cap F_A$. We claim that if there exist other jobs of type η, i.e., $\Gamma_\eta \setminus B_\eta \neq \emptyset$, we may rearrange the Γ_η-jobs so that the jobs in B_η have largest indices. Indeed, assume that the final job in machine M_1 is of type η and its index is i_1 and that machine M_2 has a non-final job of the same type η whose index is $i_2 > i_1$. Swapping between those two jobs would not change the value of $T(A)$ since they are of equal size. Moreover, such a swapping is permissible since in machine M_1 we replace the final job with another job of a higher index, while in machine M_2 we replace a non-final job with another one of a smaller index (the latter operation is legal since we may rearrange the non-final jobs in M_2).

In view of the above, it is sufficient to review only selections where the selected jobs of a given type η have the highest indices among the jobs of that type, Γ_η. The number of such selections equals

$$\#\{ \, (b_{-1}, \ldots, b_{q-1}) \in \mathcal{N}^{q+1} : \sum_{\eta=-1}^{q-1} b_\eta = m-r+1 \, , \; 0 \leq b_\eta \leq |\Gamma_\eta| \, \}$$

and, consequently, may be bounded by $(m+1)^{q+1}$. $\square$

Step 2: A makespan problem with constraints.

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 $1 \leq k \leq m$ 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 $1 \leq k \leq m$ 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,

$$V_k = \mathcal{P}(\mathbf{z}^k) \quad 0 \leq k < m \quad \text{and} \quad V_m = \{\mathbf{z}^m\} \,. \tag{29}$$

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, $w[(\mathbf{u}, \mathbf{v})] = \sum_{\eta=0}^{q} (v_\eta - u_\eta) \cdot \xi_\eta \cdot p^{\text{lnf}}$, where $\{\xi_\eta\}_{\eta=0}^{q-1}$ are given in (4) 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, $1 \leq k \leq m$.

2.4 Performance estimates.

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

Proof. We start with estimating the running time of the core algorithm. The outer loop in the core algorithm goes over all possible selections of the complement set of final jobs. The number of such selections is $O((m+1)^{q+1})$, Lemma 1. For a given selection, we need to sort the final jobs according to their index – $\Theta(m \log m)$ time steps – and then construct the graph and find the shortest path in that graph. The cost of this graph procedure is bounded by $\Theta(|V| + |E|)$. The size of each layer in the graph, $|V_k|$, $0 < k < m$, is bounded by $(n+1)^{q+2}$. Consequently, $|V| \leq 2+(m-1)\cdot(n+1)^{q+2}$, while $|E| \leq m\cdot(n+1)^{2(q+2)}$. Finally, the last step of recovering a job assignment from the shortest path has a cost of $\Theta(n)$. As all of the costs above are polynomial in both n and m, we conclude that the running time of the core algorithm is polynomial in n and m.

This implies that the entire algorithm has a polynomial running time:

1. Identifying the $m+1$ largest jobs and sorting them: $\Theta(n + m \log m)$ steps.
2. A loop over all possible values of $p^{\text{lnf}} = p_{j_r} \in J^{m+1}$: $m+1$ steps.
   (a) Define $\mathcal{L}'$ and $\hat{\mathcal{S}}$ – $\Theta(n)$ steps.
   (b) Apply the core algorithm – polynomial in n and m.
3. Translating the assignment of modified jobs into an assignment of the original jobs – $\Theta(n)$.

□

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 $0 < \varepsilon \leq 1$, $T(A) \leq (1+35\varepsilon)T^o$.*

Proof. Let A be an optimal assignment and denote $T^o = T(A)$. After the $\mathcal{S} \cup \mathcal{L} \mapsto \hat{\mathcal{S}} \cup \mathcal{L}'$ modification of the original jobs, the value of the assignment $\hat{A}$ that corresponds to A is bounded by $T(\hat{A}) \leq (1+2\varepsilon)T^o$, as implied by Propositions 2 and 3. The shortest path solution may yield a different assignment, $\check{A}$, such that $T(\check{A}) \leq T(\hat{A})$. Translating that assignment into an assignment $\bar{A}$ in terms of the original jobs, we may conclude that $T(\bar{A}) \leq (1+\varepsilon)\cdot(1+2\varepsilon)T(\check{A})$, Propositions 2 and 4. $\bar{A}$ is the assignment that our PTAS outputs. Summarizing all of the above, we conclude that $T(\bar{A}) \leq (1+\varepsilon)\cdot(1+2\varepsilon)^2\cdot T^o \leq (1+17\varepsilon)T^o$ for all $0 < \varepsilon \leq 1$.

Finally, taking into account the possibility of collisions in the original sequence of jobs, we should multiply the latter factor of $(1+17\varepsilon)$ by $(1+\varepsilon)$ which is a bound on the price incurred by the procedure that removes collisions, Proposition 1. That yields a multiplicative factor of $(1+35\varepsilon)$ for all $0 < \varepsilon \leq 1$. □

3 A PTAS for the Makespan Problem in the Linear Hierarchical Model

In the unrelated machine model, the jobs and machines are arranged in a full weighted bipartite graph, $G = (V, E)$, where $V = \mathcal{J} \cup \mathcal{M}$, $\mathcal{J}$ is the set of input jobs and $\mathcal{M}$ is the set of machines, $E = \mathcal{J} \times \mathcal{M}$, and the weight of an edge $e = (J_i, M_k)$ is the processing time of job J_i on machine M_k. If for each of the nodes $J_i \in \mathcal{J}$, $w(J_i, M_k) \in \{p_i, \infty\}$ for all $M_k \in \mathcal{M}$, we get the restricted assignment model. In that model, each job is associated with a subset of permissible machines and it may be executed only on one of those machines, where its processing time on a permissible machine is always p_i. If $\mathcal{M}$ is partially ordered and whenever a job is executable on a given machine it is also executable on machines that are higher in the hierarchy, we get the hierarchical model. Finally, if the hierarchy is linear, in the sense that $M_1 \leq M_2 \leq \cdots \leq M_m$, we get the linear hierarchical model [2]. In such a 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 $1 \leq k \leq m$. Bar-Noy et al [2] studied the makespan problem in several variants of the hierarchical model and designed competitive algorithms for their solution. Here we show that the makespan problem in the linear hierarchical model may be reduced to the min-max starting time problem.

Proposition 5 *The makespan problem in the linear hierarchical model is reducible to the min-max starting time problem.*

Proof. We denote herein the two problems by P1 and P2 respectively and we show that P1 is reducible to P2. Let $\mathcal{J}$ be the set of input jobs to P1. We decompose it into $\mathcal{J} = \bigcup_{k=1}^{m} \mathcal{J}_k$ where $\mathcal{J}_k$ is composed of all jobs for which the permissible suffix subset of machines is $\{M_k, \ldots, M_m\}$, $1 \leq k \leq m$. We define $\hat{\mathcal{J}} = \mathcal{J} \cup \{t_k\}_{k=1}^{m}$ where t_k are jobs of a very large processing time, and we order the jobs in $\hat{\mathcal{J}}$, thus turning it into a sequence, in the following manner,

$$(\mathcal{J}_1, t_1, \mathcal{J}_2, t_2, \ldots, \mathcal{J}_m, t_m) ,$$

where the inner order within each of the subsets $\mathcal{J}_k$ is arbitrary. We then consider problem P2 with the input sequence $\hat{\mathcal{J}}$.

An optimal solution of the latter problem will obviously place t_k to be the last job in one of the machines, for all $1 \leq k \leq m$. Without loss of generality, we may assume that t_k is the last job on machine M_k, $1 \leq k \leq m$. The way that we ordered the sequence of jobs $\hat{\mathcal{J}}$ implies that such an optimal solution induces a legal solution for P1 by removing the m largest jobs. Hence, the optimal min-max starting time for P2 is bounded from below by the optimal makespan for P1.

On the other hand, any optimal solution of P1 may be translated to a legal solution for P2 by adding t_k as the last job on machine M_k, $1 \leq k \leq m$, and rearranging the original jobs on that machine according to their arbitrary order in the sequence $\hat{\mathcal{J}}$; such a rearrangement has no effect on neither the makespan in P1, nor the min-max starting time in P2. Hence, the optimal makespan for P1 is bounded from below by the optimal min-max starting time for P2.

Therefore, the optimal makespan for P1 equals the optimal min-max starting time for P2, and P2 may be solved by the above simple reduction to P1. □

Corollary 1 *The makespan problem in the linear hierarchical model admits a polynomial time approximation scheme.*

4 A Fully Polynomial Time Approximation Scheme for a Constant m

In this section we design an FPTAS for the case where m is constant. This construction is relatively simple. Let m be a given constant. Then there are $O(n^m)$ possibilities of selecting the m final jobs. Assume that the selected final jobs are $F_A = \{f_k : 1 \leq k \leq m\}$ and that $f_0 = 0 < f_1 < f_2 < \ldots < f_m$. Such a selection induces a partition of the remaining jobs into m subsets,

$$F_A^c = \cup_{k=1}^m \mathcal{J}_k \quad \text{where} \quad \mathcal{J}_k = \{i : f_{k-1} < i < f_k\} \ .$$

The jobs of $\mathcal{J}_k$ may be assigned only to a suffix subset of machines, $\{M_k, \ldots, M_m\}$. The goal now is to assign the jobs from F_A^c so that the above constraints are respected and the makespan is minimized. But this latter problem is the makespan minimization problem in the linear hierarchical model, which, in turn, is a special case of the unrelated machines model, as discussed in Section 3. Horowitz and Sahni [8], and later Jansen and Porkolab [9], designed an FPTAS for the makespan problem in that latter model. We may use that FPTAS as the core algorithm in our setting in order to approximate the optimal solution subject to a given selection of m final jobs. Repeating this procedure for each of the $O(n^m)$ selections of final jobs and choosing the assignment that achieves the minimal cost, we get an approximate solution to our problem in time that is polynomial in n and $\frac{1}{\varepsilon}$.

Corollary 2 *The min-max starting time problem, with a fixed number of machines m, admits a fully polynomial time approximation scheme.*

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] A. Bar-Noy, A. Freund and J. Naor. 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.

[3] 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, Springer-Verlag, 2002.

[4] R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[5] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:416–429, 1969.

[6] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.

[7] D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[8] 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.

[9] 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.

[10] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity os scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[11] S.R. Mehta, R. Chandrasekaran, and H. Emmons. Order-presrving allocation of jobs to two machines. *Naval Research Logistics Quarterly*, 21:846–847, 1975.

[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.

[14] J. Sgall. On-Line Scheduling. In *A. Fiat and G. J. Woeginger, editors,* Online Algorithms: The State of the Art, volume 1442 of *LNCS*, pages 196–231. Springer-Verlag, 1998.

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