# Optimal Preemptive Scheduling for General Target Functions

Leah Epstein[1] and Tamir Tassa[2]

[1] School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. ** lea@idc.ac.il.
[2] 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.

**Abstract.** We study the problem of optimal preemptive scheduling with respect to a general target function. Given $n$ jobs with associated weights and $m \leq n$ uniformly related machines, one aims at scheduling the jobs to the machines, allowing preemptions but forbidding parallelization, so that a given target function of the loads on each machine is minimized. This problem was studied in the past in the case of the makespan. Gonzalez and Sahni [7] and later Shachnai, Tamir and Woeginger [12] devised a polynomial algorithm that outputs an optimal schedule for which the number of preemptions is at most $2(m-1)$. We extend their ideas for general symmetric, convex and monotone target functions. This general approach enables us to distill the underlying principles on which the optimal makespan algorithm is based. More specifically, the general approach enables us to identify between the optimal scheduling problem and a corresponding problem of mathematical programming. This, in turn, allows us to devise a single algorithm that is suitable for a wide array of target functions, where the only difference between one target function and another is manifested through the corresponding mathematical programming problem.

## 1 Introduction

We are interested in the problem of optimal preemptive scheduling with respect to a general target function. The data in such problems consists of:

- $n$ jobs, $\mathcal{J} = \{J_i\}_{1 \leq i \leq n}$, where job $J_i$ has a weight $w_i$ and $w_1 \geq w_2 \geq \cdots \geq w_n > 0$.
- $m$ machines, $\mathcal{M} = \{M_j\}_{1 \leq j \leq m}$, $m \leq n$, where machine $M_j$ has speed $s_j$ and $1 = s_1 \geq s_2 \geq \cdots \geq s_m > 0$.

If a job of weight $w$ runs on a machine of speed $s$, its processing time will be $w/s$. A non-preemptive schedule of the jobs to the machines is a function $\sigma : \mathcal{J} \to \mathcal{M}$.

** The research of this author supported in part by Israel Science Foundation grant 250/01.

In such schedules, once a job started its process on a given machine, it is executed continuously until completion. We, however, are interested here in preemptive schedules, where a job's execution may be terminated and then resumed later, possibly on a different machine.

**Definition 1.** *A preemptive schedule is a vector $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)$ where $\sigma_j$ is the schedule on $M_j$, $1 \le j \le m$. The machine schedule $\sigma_j$ takes the form of a pair of sequences, $\sigma_j = (\tau_j, \eta_j)$, where $\tau_j$ is a sequence of strictly increasing times, $0 = \tau_{j,0} < \tau_{j,1} < \cdots < \tau_{j,k_j}$, and $\eta_j$ is a sequence of indices, i.e., $\eta_j = (\eta_{j,1}, \ldots, \eta_{j,k_j})$ where $\eta_{j,k} \in \{0, 1, \ldots, n\}$ for all $1 \le k \le k_j$. Such a schedule means that $M_j$ processes $J_{\eta_{j,k}}$ in time interval $[\tau_{j,k-1}, \tau_{j,k})$, for all $1 \le k \le k_j$, unless $\eta_{j,k} = 0$ in which case $M_j$ is idle during the corresponding time interval.*

The schedule is *legal* if the same job is never scheduled to be processed at the same time by two different machines (namely, parallelization is forbidden). The schedule is *complete* if for every given job, the sum over all machines of its processed parts amounts to its weight, i.e.,

$$\sum_{(j,k):\ \eta_{j,k}=i} (\tau_{j,k} - \tau_{j,k-1}) \cdot s_j = w_i \quad \text{for all} \quad 1 \le i \le n \ . \tag{1}$$

Hereinafter we consider only complete and legal schedules.

For a given schedule, $\boldsymbol{\sigma}$, we let $\boldsymbol{\lambda}(\boldsymbol{\sigma}) = (\lambda_1, \ldots, \lambda_m)$ denote the corresponding vector of *loads*, where $\lambda_j := \tau_{j,k_j}$ is the *time* in which $M_j$ finishes running under the schedule $\boldsymbol{\sigma}$. One usually seeks schedules that minimize the value of some target function of the loads,

$$f(\boldsymbol{\sigma}) = f(\boldsymbol{\lambda}(\boldsymbol{\sigma})) = f(\lambda_1, \ldots, \lambda_m) \ , \tag{2}$$

where $f$ is typically a convex, symmetric and monotonically non-decreasing function with respect to its arguments.

For a given target function $f$, we let $f_{opt}$ denote its optimal value, i.e., $f_{opt} = \min_{\boldsymbol{\sigma}} f(\boldsymbol{\sigma})$. The usual choice is the makespan, $f = \max$. This case was studied in [11, 10, 7, 12, 5]. Liu and Yang [11] introduced bounds on the cost of optimal schedules. Horvath, Lam and Sethi proved that the optimal cost is indeed the maximum of those bounds by constructing an algorithm that uses a large number of preemptions. Gonzalez and Sahni [7] devised a polynomial algorithm that outputs an optimal schedule for which the number of preemptions is at most $2(m-1)$. This number of preemptions was shown to be optimal in the sense that there exist inputs for which every optimal schedule involves that many preemptions. This algorithm was later generalized and simplified for jobs of limited splitting constraints by Shachnai, Tamir and Woeginger [12]. In this paper we extend the ideas of [12] for general symmetric, convex and monotone target functions. This general approach offers several benefits over the study of the particular makespan problem. By looking at the problem from a more general perspective, we are able to distill the underlying principles on which the algorithm of [12] is based. This approach enables us to identify between the optimal scheduling problem and a problem of mathematical programming. This, in turn, allows us to devise a single algorithm that is suitable for a wide array of target functions, where the only difference between one target function and another is manifested through the corresponding mathematical programming problem. Lastly, this approach facilitates the presentation and analysis of the algorithm.

The paper begins with a study of properties of optimal schedules, Section 2. We show that when the target function is convex, symmetric and monotone, there always exist optimal schedules of a relatively simple structure. Specifically, there always exist optimal schedules where the loads on faster machines are greater than or equal to the loads on slower machines and there are no idle times, Proposition 1. As a consequence of this characterization of (some) optimal schedules, we define a mathematical program (i.e., a problem of minimizing a multivariate target function in a bounded polyhedron) whose solution is the set of machine loads of an optimal schedule, Theorem 1. Section 3 is then dedicated to the presentation and analysis of Algorithm 1. This algorithm receives as an input a set of machine loads from the polyhedron that corresponds to the equivalent mathematical program, and it outputs a complete and legal preemptive schedule with those machine loads. Hence, if one runs this algorithm with the set of machine loads that solved the mathematical program, one gets an optimal preemptive schedule to the original problem, Theorem 3. In Appendix A we illustrate the algorithm with an example.

The problem of finding an optimal preemptive schedule is therefore separated into two independent stages. In the first stage we write down the corresponding mathematical program and solve it. In that mathematical program we aim at minimizing the function (2) in a bounded polyhedron in $\mathcal{R}^m$ that reflects a set of linear constraints that manifest our demand for completeness and legality of the schedule. After solving this mathematical program, we face an algorithmic problem: finding a preemptive schedule whose loads equal the solution of the mathematical program. This is achieved by Algorithm 1. This second stage is general in the sense that it is independent of the choice of the target function.

After presenting and studying the general algorithm, we derive explicit results for specific target functions, Section 4. In Section 4.1 we devise a polynomial time algorithm for the solution of the mathematical program when the target function is the $\ell_p$-norm, $f(\lambda_1, \ldots, \lambda_m) = \left(\sum_{j=1}^m \lambda_j^p\right)^{1/p}$, $1 \le p < \infty$. This target function was studied in the past in the non-preemptive setting [1, 6]. We show that by taking the limit $p \uparrow \infty$ the algorithm solves also the makespan minimization problem and that the minimal makespan of optimal preemptive schedules agrees with the value that was derived in [7, 12]. In Section 4.2 we continue to explore the threshold cost function, $f(\lambda_1, \ldots, \lambda_m) = \sum\limits_{j=1}^{m} \max(\lambda_j, c)$ where $c > 0$ is some constant threshold, and devise a polynomial time algorithm for its solution. This target function was also studied in the past for non-preemptive scheduling [2–4]. We note that an algorithm due to Hochbaum and Shanthikumar [8] may be applied in order to solve the mathematical program in polynomial time whenever the target function is separable, i.e., $f(\lambda_1, \dots, \lambda_m) = \sum_{j=1}^{m} g(\lambda_j)$. It should be noted that even though the $\ell_p$-norm target function, with $p < \infty$, and the threshold target function, are separable, the algorithms that we offer for these cases are simpler and more efficient than the general algorithm in [8]. More details are given in the full version.

As a concluding remark we recall that the non-preemptive versions of the above problems are typically strongly NP-hard. Approximation schemes for the makespan problem were given by Hochbaum and Shmoys [9]. The papers [1, 6] offer approximations schemes for a wide class of target functions, including the $\ell_p$-norms.

Most of the proofs are omitted from the body of the text; they may be found in the full version of this paper.

## 2 Properties of Optimal Schedules

In this section we derive some qualitative properties of optimal schedules for general symmetric and monotone target functions.

**Proposition 1.** *There exist optimal schedules in which the loads $\lambda_j$ are monotonically non-increasing and there are no holes (i.e., no idle times on a machine after which it resumes processing).*

Hereinafter we concentrate only on optimal schedules that comply with Propositions 1. We define the weight on $M_j$ as $\mu_j = s_j \lambda_j$. Namely, the weight on machine $M_j$ under a schedule $\boldsymbol{\sigma}$ represents the total weight of job parts that are scheduled by $\boldsymbol{\sigma}$ to be processed on $M_j$. We also define the following:

$$W_k = \begin{cases} \sum_{j=1}^{k} w_j \ 1 \le k \le m-1 \\ \\ \sum_{j=1}^{n} w_j \ k = m \end{cases} . \tag{3}$$

With these definitions, we state the following key proposition.

**Proposition 2.** *In optimal schedules that comply with Proposition 1*

$$\sum_{j=1}^{k} \mu_j \ge W_k \quad , \quad 1 \le k \le m-1 \quad \textit{while} \quad \sum_{j=1}^{m} \mu_j = W_m \ . \tag{4}$$

*Proof.* As the inequality in (4) is just the completeness requirement, we focus on proving the inequality in (4) for an arbitrary value of $1 \le k \le m-1$. Let $R_\ell$ be defined as the union of all time intervals in which *exactly* $\ell$ of the $k$ largest jobs, $\mathcal{J}_k := \{J_i\}_{1 \le i \le k}$, are running. Namely, if $t \in R_\ell$, there exists a subset of $\ell$ jobs $\{J_{i_1}, \dots, J_{i_\ell}\} \subset \mathcal{J}_k$ that are scheduled to run on $\ell$ machines at time $t$, while the remaining $k-\ell$ jobs in $\mathcal{J}_k$ are not being processed at that time. In view of Proposition 1, the entire schedule is embedded in the time interval $[0, \lambda_1)$.

We break up this interval into a disjoint union $[0, \lambda_1) = \bigcup_{\ell=0}^{k} R_\ell$. Proposition 1 implies that

$$\bigcup_{\ell=j}^{k} R_\ell \subset [0, \lambda_j) \quad , \quad 1 \le j \le k \ . \tag{5}$$

Let $r_\ell$ denote the amount of work that was done on the $k$ largest jobs during $R_\ell$. Then, as the schedule is complete, $\sum_{\ell=1}^{k} r_\ell = W_k$. On the other hand, since the schedule is legal, $r_\ell$ may not exceed the duration of $R_\ell$ times the sum of speeds of the $\ell$ fastest machines, i.e., $r_\ell \le S_\ell \cdot |R_\ell|$, where hereinafter $S_\ell = \sum_{j=1}^{\ell} s_j$. Hence, by (5), $W_k \le \sum_{\ell=1}^{k} \left(\sum_{j=1}^{\ell} s_j\right) \cdot |R_\ell| = \sum_{j=1}^{k} s_j \cdot \left(\sum_{\ell=j}^{k} |R_\ell|\right) \le \sum_{j=1}^{k} s_j \lambda_j = \sum_{j=1}^{k} \mu_j$. $\square$

Finally, we state our main result.

**Theorem 1.**

$$f_{opt} = \min_{\Omega} f\left(\frac{\mu_1}{s_1}, \ldots, \frac{\mu_m}{s_m}\right) \tag{6}$$

*where $\Omega \subset (\mathcal{R}^+)^m$ is the bounded polyhedron of all nonnegative weights $\mu_j$ that satisfy the legality and completeness constraints (4).*

*Proof.* Let $f_{min}$ denote the minimum of the optimization problem (6) under the constraints in (4) (This optimization problem is a mathematical program to which we refer henceforth as MP). $f_{min}$ is well defined since $f$ is convex and $\Omega$, the domain in which the minimum is sought, is closed and convex. Proposition 2 imply that $f_{opt} \ge f_{min}$. Since Algorithm 1 in the next section produces a complete and legal preemptive schedule with weights $\{\mu_j\}_{1\le j\le m}$ for *any* $(\mu_1, \ldots, \mu_m) \in \Omega$, we infer that $f_{opt} = f_{min}$. $\square$

## 3 An Optimal Scheduling Algorithm for a General Target Function

In this section we present and analyze an algorithm that, given a legal and complete allocation of weights to machines, $\{\mu_j\}_{1\le j\le m} \in \Omega$, finds a preemptive schedule of the jobs to the machines that agrees with those weights. Hence, if we call that algorithm with a solution of the mathematical program MP, we get in return an optimal preemptive schedule.

### 3.1 The Algorithm

Let $\{\mu_j\}_{1\le j\le m} \in \Omega$ be a set of nonnegative weights that satisfy the conditions in (4). Let

$$\lambda_j = \frac{\mu_j}{s_j} \quad , \quad \Lambda_j = \sum_{k=1}^{j} \lambda_k \quad , \quad 1 \le j \le m \quad \text{and} \quad \Lambda_0 = 0 \ . \tag{7}$$

Next, we define the following state functions on $[0, \Lambda_m)$: a potential function

$$\Psi(x) = \sum_{j=1}^{m} s_j \cdot \chi_{[\Lambda_{j-1}, \Lambda_j)}(x) \quad \text{where} \quad \chi_I(x) = \begin{cases} 1 & x \in I \\ 0 & x \notin I \end{cases}, \tag{8}$$

a timing function

$$\Theta(x) = \begin{cases} x - \Lambda_{j-1} & \text{if } x \in [\Lambda_{j-1}, \Lambda_j) \text{ for some } 1 \leq j \leq m \\ 0 & \text{otherwise} \end{cases}, \tag{9}$$

and an indicator function

$$\Gamma(x) = \begin{cases} j & \text{if } x \in [\Lambda_{j-1}, \Lambda_j) \text{ for some } 1 \leq j \leq m \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

(see Figures 1-3).

Those three functions will always be examined as a triplet, along some interval $[a, b)$: the value of the indicator function $\Gamma(x)$ will indicate the machines under consideration; the range of values of $\Theta(x)$ along that interval will indicate the time segments on which we focus; and the corresponding integrals of the potential function $\Psi$ represents the work that can be done by the corresponding machines in the corresponding time segments. For example, let us consider the state functions in Figures 1-3. They represent a problem with 4 machines, of speeds $1, 0.8, 0.6$ and $0.3$, where the allocated loads are $10, 8, 6$ and $3$ (e.g., the length of the schedule on $M_3$ is 6 time units). Consider the interval $[9, 12)$. Then, looking at $\Gamma^1$ and $\Theta^1$, this interval corresponds to time segment $[9, 10)$ on $M_1$ and time segment $[0, 2)$ on $M_2$. The corresponding integral $\int_9^{10} \Psi^1(x)dx$ gives the amount of work that can be accomplished by $M_1$ in time segment $[9, 10)$, and $\int_{10}^{12} \Psi^1(x)dx$ gives the amount of work that can be accomplished by $M_2$ in time segment $[0, 2)$.

Algorithm 1 produces a preemptive schedule $\boldsymbol{\sigma}$ of $\mathcal{J}$ on $\mathcal{M}$ such that the weight on machine $M_j$ equals $\mu_j$. The jobs are scheduled one by one, from the largest to the smallest. We schedule a job by associating with it an interval $[a, b) \subset [0, \Lambda_m)$ such that $\int_a^b \Psi(x)dx$ equals the weight of that job, and $\Theta(x)$ is injective along that interval (in order to avoid parallelization). The main idea is to always schedule the next job to the slowest possible machine in the latest possible time; namely, we aim at finding an interval $[a, b) \subset [0, \Lambda_m)$ with the maximal $a$ that can still accommodate that job. Once we found such an interval, we record the resulting schedule of the job by looking at the values of $\Gamma$ and $\Theta$ along $[a, b)$. Then, we have to mark the interval as being occupied. Instead of doing that, we simply remove that interval from the graphs of the three state functions and left shift the "tail" of those graphs, $[b, \infty)$, by $b - a$, in order to close the gap that was created by this removal.

In carrying out the above procedure, we define for each point $a$ the point $b = End(a)$ that stands for the maximal point $b$ such that $\Theta(x)$ is injective on $[a, b]$. As $\Theta(x)$ has a very simple piecewise linear form and it retains that form after each "cut-and-shift" operation that corresponds to scheduling a job, it is very simple to compute the function $b = End(a)$.

**Algorithm 1** *1. Initialize $\Psi$, $\Theta$ and $\Gamma$ according to (7)–(10).*
*2.* $i = 1$ *(current job number).*
*3. Define $End(a)$ for all $a \in [0, \Lambda_m)$ in the following manner:*

$$End(a) = \min(a+\Lambda_j-\Lambda_{j-1}, \Lambda_{j+1}) \quad \forall a \in [\Lambda_{j-1}, \Lambda_j) \quad , \quad 1 \le j \le m \ , \tag{11}$$

*where, for the sake of the last interval, we take $\Lambda_{m+1} = \Lambda_m$.*
*4. Find the maximal value of $a$ for which*

$$\int_a^{b=End(a)} \Psi(x)dx = w_i \ . \tag{12}$$

*5. Decompose the interval $[a, b)$ into a disjoint union of intervals,*

$$[a, b) = \bigcup_{r=1}^{\ell} [a_{r-1}, a_r) \tag{13}$$

*where $a_0 = a$, $a_\ell = b$, $\Gamma$ is constant along $[a_{r-1}, a_r)$, say $\Gamma|_{[a_{r-1},a_r)} = j_r$, and $j_1 < j_2 < \cdots < j_\ell$.*
*6. Compute*

$$w_{i,r} = \int_{a_{r-1}}^{a_r} \Psi(x)dx \quad , \quad 1 \le r \le \ell \ . \tag{14}$$

*7. Break up $J_i$ into $\ell$ parts, $\{J_{i,r}\}_{1\le r\le \ell}$, where the weight of $J_{i,r}$ is $w_{i,r}$, $1 \le r \le \ell$.*
*8. Schedule $J_{i,r}$ to run on $M_{j_r}$ in time interval $[\Theta(a_{r-1}), \Theta(a_r))$ , $1 \le r \le \ell$.*
*9. Remove the interval $[a, b)$ from $\Psi$, $\Theta$ and $\Gamma$. More specifically, apply on all three functions the following operator:*

$$U_{[a,b)}\Phi := \Phi \cdot \chi_{[0,a)} + L_{b-a}\left\{\Phi \cdot \chi_{[b,\infty)}\right\} \ , \tag{15}$$

*where $L_d$ is the d-left shift operator, i.e., $L_d\Phi(x) = \Phi(x + d)$.*
*10. Update $m$ to indicate the number of discontinuities in the modified timing function $\Theta$ and set $\Lambda_j$, $1 \le j \le m$, to be the corresponding $j$th discontinuity.*
*11.* $i = i + 1$.
*12. If $i > n$ stop. Else go to Step 3.*

The reader is referred to Appendix A where the algorithm is exemplified.

### 3.2 Analysis

In this section we prove the validity of the algorithm. Hereinafter, whenever necessary to distinguish between two subsequent rounds, we use the superscript $i$ to denote the values of the algorithm variables during the $i$th round in the algorithm, $1 \le i \le n$. Namely, $\Psi^i(x)$, $\Theta^i(x)$ and $\Gamma^i(x)$ are the three state functions during the $i$th round (before they are being updated in Step 9), $m^i$ is the number of discontinuities of $\Theta^i(x)$ while $\{\Lambda^i_j\}_{1\le j\le m^i}$ are those discontinuities (with $\Lambda_0^i = 0$), and $End^i(a)$ is the function that is defined in (11) at the $i$th round. We also define $\Omega^i = [0, \Lambda_{m^i}^i)$ to be the support of the state functions in round $i$, $\Omega_j^i = [\Lambda_{j-1}^i, \Lambda_j^i)$, $1 \le j \le m^i$, be the decomposition of $\Omega^i$ into intervals of continuity of $\Theta^i(x)$, and $\lambda_j^i = |\Omega_j^i|$.

The timing function and its continuity intervals play a significant role in the analysis of Algorithm 1. The next two lemmas provide important information about $\Theta^i(x)$.

**Lemma 1.** *(i) The timing function is linear on each continuity interval, i.e.,*

$$\Theta^i(x) = \begin{cases} x - \Lambda_{j-1}^i & \text{if } x \in \Omega_j^i \text{ for some } 1 \le j \le m^i \\ 0 & \text{otherwise} \end{cases} . \tag{16}$$

*(ii)* $\lambda_j^i$, $1 \le j \le m^i$, *form a non-increasing sequence for all* $i$.

Assume that round $i^*$ was the first round in which we selected in Step 4 a value of $a$ with $End^{i^*}(a) = \Lambda_{m^{i^*}}^{i^*}$ (namely, this is the first round in which the sliding window went all the way to the right end point of the current support, $[0, \Lambda_{m^{i^*}}^{i^*})$, of the three state functions). It is not hard to see that, as the jobs are ordered in a non-increasing order according to their weight, the same will happen in all subsequent rounds, i.e., $End^i(a) = \Lambda_{m^i}^i$ for all $i \ge i^*$. We refer to the first $i^* - 1$ rounds in the execution of the algorithm as *Phase 1*, while rounds $i^*$ through $n$ constitute *Phase 2*. With this terminology, we proceed as follows.

**Lemma 2.** *During Phase 1 the number of $\Theta^i$-continuity intervals always decreases by one. Namely, for all $i$, $1 \le i \le i^*$, $m^i = m - i + 1$. Consequently, Phase 1 lasts no more than $m - 1$ rounds.*

The next fundamental proposition is crucial for the justification of Algorithm 1.

**Proposition 3.** *In all rounds, the set of values of $a$ that satisfy requirement (12) in Step 4 of the algorithm is nonempty and it has a maximum.*

**Theorem 2.** *Algorithm 1 generates a complete and legal schedule.*

*Proof.* The algorithm is well defined in view of Lemma 1 and Proposition 3. This implies the completeness of the resulting schedule since each job is assigned time shares on the machines that enable its completion. The schedule is legal since $End(a)$ is defined so that the timing function $\Theta^i(x)$ is one-to-one along the interval $[a, End(a))$. □

In view of all of the above, we reach our final statement regarding Algorithm 1.

**Theorem 3.** *Algorithm 1 outputs an optimal preemptive schedule when the input $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_m)$ is a solution of the corresponding mathematical program* MP, *namely, when it minimizes (6) under (4).*

It may be shown that the number of preemptions that are enforced by the algorithm is bounded by $2(m - 1)$. This was shown to be minimal for some inputs for the makespan minimization problem [7].

## 4 Examples of Target Functions

### 4.1 The $\ell_p$-Norm

Here we present a polynomial time algorithm that constructs an optimal solution $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_m)$ to MP where $f$ is the $\ell_p$-norm. The run time of the algorithm is $O(m^2)$. After presenting the algorithm, we prove that its output, $\boldsymbol{\mu}$, is in $\Omega$ and that it is a minimal point in $\Omega$. Even though this section concentrates on $1 < p < \infty$, the results presented herein apply equally to $p = 1$ and $p = \infty$ by taking the corresponding limit. Moreover, when $p = \infty$, we recover an explicit expression for the optimal makespan that was derived in [7].

**Algorithm 2** *1. Set $t = 0$ and $k_t = 0$ (at each stage $k_t$ equals the number of values $\mu_j$ that were already determined).*

*2. For every $k_t + 1 \le k \le m$, compute*

$$q_k = (W_k - W_{k_t})/S_p[k_t + 1 : k] \quad \text{where} \quad S_p[a : b] = \sum_{j=a}^{b} s_j^{p/(p-1)} \ . \tag{17}$$

*and set $k_{t+1}$ to be the (minimal) value of $k$ for which $q_k$ is maximal.*

*3. For all $k_t + 1 \le j \le k_{t+1}$, set*

$$\mu_j = s_j^{p/(p-1)} \cdot (W_{k_{t+1}} - W_{k_t})/S_p[k_t + 1 : k_{t+1}] \ . \tag{18}$$

*4. If $k_{t+1} < m$ set $t = t + 1$ and go to Step 2.*

We note that the algorithm solves also the extremal cases $p = 1$ and $p = \infty$. When $p = \infty$, the powers $p/(p-1)$ need to be understood as 1. As for $p = 1$, let $b$ denote the number of machines of maximal speed, i.e., $s_j = 1$ for $1 \le j \le b$ and $s_j < 1$ for $b < j \le m$. When $p \downarrow 1$, the powers $p/(p-1) \uparrow \infty$. Hence, $s_j^{p/(p-1)} = 1$ for $1 \le j \le b$ and zero for $b < j \le m$. As a consequence, by (18), the machines which are not among the fastest, $M_j$, $b < j \le m$, will be assigned nothing, $\mu_j = 0$, and the entire weight will be spread among the $b$ fastest machines. The manner in which the total weight will be spread among those machines depends on the data but is insignificant because the $\ell_1$-norm does not distinguish between such assignments. Such schedules are of-course optimal.

**Lemma 3.** *Let $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_m)$ be the solution that Algorithm 2 returned and let $\{k_i\}_{0 \le i \le t+1}$ be the corresponding sequence of indices that were identified during the execution of the algorithm. Then $\boldsymbol{\mu} \in \Omega$. Namely, it satisfies (4). Moreover, the set of indices for which (4) holds with equality is exactly $\{k_i\}_{1 \le i \le t}$.*

Next, we claim that $\boldsymbol{\mu}$ is optimal for $1 < p < \infty$.

**Theorem 4.** *Let $\boldsymbol{\mu}' = \{\mu'_j\}_{1 \le j \le m}$ be an optimal solution of* MP *where $1 < p < \infty$. Then $\boldsymbol{\mu}' = \boldsymbol{\mu}$.*

The optimality of $\boldsymbol{\mu}$ for $p = \infty$ is a direct consequence of Lemma 4 as the makespan is the limit of the $\ell_p$-norms when $p \uparrow \infty$. In this case, we may even derive an explicit expression for the optimal makespan. The solution $\boldsymbol{\mu}$ that Algorithm 2 outputs satisfies $\max_{1\le k\le m} \frac{W_k}{S_p[1:k]} = \frac{\mu_1}{s_1^{1+\frac{1}{p-1}}} \ge \frac{\mu_2}{s_2^{1+\frac{1}{p-1}}} \ge \cdots \ge \frac{\mu_m}{s_m^{1+\frac{1}{p-1}}}$. When $p = \infty$, this translates into $\max_{1\le k\le m} \frac{W_k}{S_\infty[1:k]} = \max_{1\le k\le m} \frac{W_k}{S_k} = \frac{\mu_1}{s_1} \ge \frac{\mu_2}{s_2} \ge \cdots \ge \frac{\mu_m}{s_m}$; here, as before, $S_k = \sum_{j=1}^k s_j$. Therefore, the optimal makespan is $\max\left(\frac{\mu_1}{s_1}, \ldots, \frac{\mu_m}{s_m}\right) = \max_{1\le k\le m} \frac{W_k}{S_k}$, as the result of [7].

### 4.2 Threshold Cost Functions

Here we study the target function $f(\mu_1, \ldots, \mu_m) = \sum_{j=1}^m \max\left(\frac{\mu_j}{s_j}, c\right)$. This case, also known as *extensible bin packing* [2–4], describes a scenario in which a fixed payment is due up-front for $c$ time units in each machine, whether they have been used or not, and, in addition, to any excessive time that was used beyond the fixed threshold in any of the machines.

Algorithm 3 computes an optimal solution $\boldsymbol{\mu} \in \Omega$ to $\mathsf{MP}$ when the target function $f$ is as above. Here, as before, $S_k = \sum_{j=1}^k s_j$ and $W_k$ is as in (3).

**Algorithm 3** *1. Set $\mu_k = 0$ for all $1 \le k \le m$ and $W = W_m = \sum_{j=1}^n w_j$.*

*2. If $W \le c \cdot s_1$ set $\mu_1 = W$ and stop.*

*3. Set $\mu_1 = \max\{c \cdot s_1, \max_{1\le k\le m}(W_k - c \cdot S_k + c \cdot s_1)\}, W = W - \mu_1$.*

*4. For $k = 2$ to $k = m$ do:*

*(a) If $W > c \cdot s_k$ then $\mu_k = c \cdot s_k$ and $W = W - c \cdot s_k$.*

*(b) Else $\mu_k = W$ and $W = 0$.*

**Theorem 5.** *The solution $\boldsymbol{\mu}$ that Algorithm 3 produces gives a minimum to the threshold cost function in $\Omega$.*

## 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. E. G. Coffman, Jr. and George S. Lueker. Approximation algorithms for extensible bin packing. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01)*, pages 586–588, 2001.
3. P. Dell'Olmo, H. Kellerer, M. G. Speranza, and Zs. Tuza. A 13/12 approximation algorithm for bin packing with extendable bins. *Information Processing Letters*, 65(5):229–233, 1998.
4. P. Dell'Olmo and M. G. Speranza. Approximation algorithms for partitioning small items in unequal bins to minimize the total size. *Discrete Applied Mathematics*, 94:181–191, 1999.
5. T. Ebenlendr and J. Sgall. Optimal and online preemptive scheduling on uniformly related machines. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computing Science (STACS-04)*, LNCS 2996:199-210, 2004.

6. L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39(1):151–162, 2004.
7. T. Gonzalez and S. Sahni. Preemptive scheduling of uniform processor systems. *Journal of the ACM*, 25(1):92–101, 1978.
8. D. S. Hochbaum and J. G. Shanthikumar. Convex separable optimization is not much harder than linear optimization. *Journal of the ACM*, 37(4):843–862, 1990.
9. 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.
10. E. C. Horvath, S. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM*, 24(1):32–43, 1977.
11. J. W. S. Liu and A. T. Yang. Optimal scheduling of independent tasks on heterogeneous computing systems. In *Proceedings ACM National Conference*, volume 1, pages 38–45. ACM, 1974.
12. H. Shachnai, T. Tamir, and G. J. Woeginger. Minimizing makespan and preemption costs on a system of uniform machines. In *Proc. of the 10th Annual European Symposium on Algorithms (ESA2002)*, pages 859–871, 2002.


# A An Example

Consider a scenario with $m = 4$ machines, the speeds of which are $(s_1, s_2, s_3, s_4) = (1, .8, .6, .3)$. Assume that the set of job weights dictates machine loads $(\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (10, 8, 6, 3)$ (when $p = 2$ and the global minimum of the $\ell_2$-norm in $\Omega$ coincides with the global minimum in $\mathcal{R}^m$, it may be shown that the machine loads indeed relate to each other like the machine speeds). Then the three state functions will be initially as described in Figures 1-3. There are $m^1 = 4$ jump discontinuities in the timing function, $\Theta^1(x)$, at $(\Lambda_1^1, \Lambda_2^1, \Lambda_3^1, \Lambda_4^1) = (10, 18, 24, 27)$.

We proceed to describe the scheduling of the first job. Assume that $w_1 = 9$. It is not hard to see that the window in which it fits, Step 4, is $[5, 15)$ (i.e., $a = 5$). The values of the indicator and timing functions, $\Gamma^1$ and $\Theta^1$, along this window imply that $J_1$ will be scheduled to run on $M_2$ in time interval $[0, 5)$ and on $M_2$ in $[5, 10)$. After scheduling $J_1$ we remove the occupied time slots by applying the cut-and-shift operator $U_{[5,15)}$. The function $\Theta^2(x)$ has $m^2 = 3$ jump discontinuities at $(\Lambda_1^2, \Lambda_2^2, \Lambda_3^2) = (8, 14, 17)$.

Next, assume that the second job is of size $w_2 = 7$. Here, the value of $a$ in Step 4 is $a = 1$ and the corresponding window is $[1, 9)$. Therefore, the values of $\Gamma^2$ and $\Theta^2$ along this interval imply that $J_2$ will be scheduled to run on $M_3$ during $[0, 1)$, on $M_1$ during $[1, 5)$ and on $M_2$ during $[5, 8)$. After applying $U_{[1,9)}$, $\Theta^3(x)$ has $m^3 = 2$ jump discontinuities at $(\Lambda_1^3, \Lambda_2^3) = (6, 9)$.

We note that if $w_3 < 0.9$, then $J_3$ will mark the beginning of *Phase 2* and the corresponding window will be completely within the last interval of continuity of $\Theta^3$; in that case $m^4 = m^3 = 2$. If, on the other hand, $w_3 \geq 0.9$, $m^4 = 1$ and then $J_4$ will be the first job in *Phase 2*.

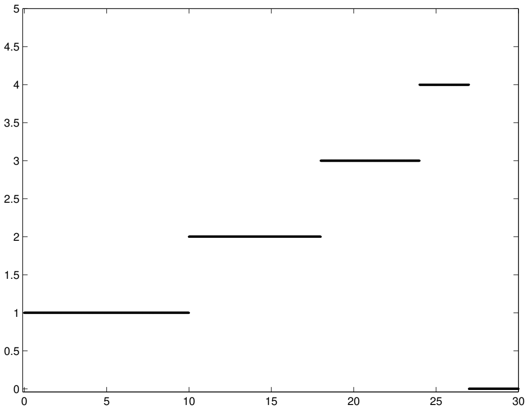


**Fig. 1.** $\Gamma^1(x)$

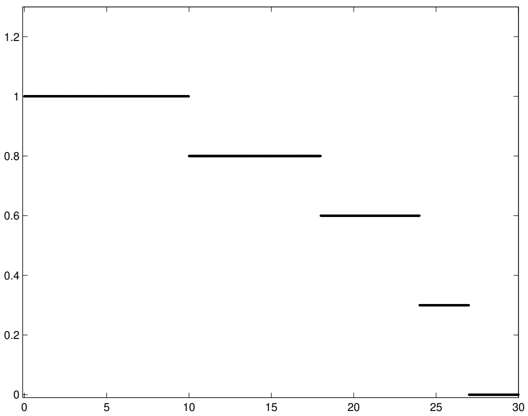


**Fig. 2.** $\Psi^1(x)$

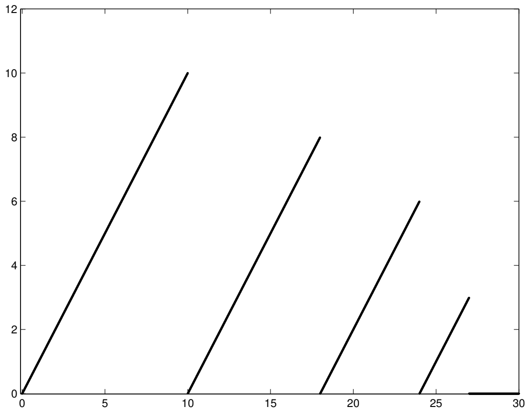


**Fig. 3.** $\Theta^1(x)$