Optimal Preemptive Scheduling for General Target Functions

Leah Epstein[*] Tamir Tassa [†]

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 [6] 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.

Keywords: Scheduling, Preemptive scheduling, Optimization, Mathematical programming.

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 $\sigma : \mathcal{J} \to \mathcal{M}$. In such schedules, once a job started its process on a given machine, it is executed continuously until completion.

[*]Department of Mathematics, University of Haifa, 31905 Haifa, Israel. `lea@math.haifa.ac.il`. Research supported in part by the Israel Science Foundation, (grant no. 250/01)

[†]Department of Mathematics and Computer Science, The Open University, Raanana, Israel. `tamirta@openu.ac.il`.

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 1.1 *A preemptive schedule is a vector $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)$ where σ_j is the schedule on M_j, $1 \le j \le m$. The machine schedule σ_j takes the form of a pair of sequences, $\sigma_j = (\tau_j, \eta_j)$, where τ_j is a sequence of strictly increasing times, $0 = \tau_{j,0} < \tau_{j,1} < \cdots < \tau_{j,k_j}$, and η_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.,

$$f_{opt} = \min_{\boldsymbol{\sigma}} f(\boldsymbol{\sigma}) \ . \tag{3}$$

The usual choice is the makespan, $f = \max$. This case was studied in [11, 10, 6, 12]. 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 [6] 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, Proposition 2.2, and there are no idle times, Proposition 2.3. 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 2.5. Section 3 is then dedicated to the presentation and analysis of Algorithm 3.1. 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 3.10. 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 $\mathbb{R}^m$ 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 3.1. 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 revisit the makespan problem and we show that the minimal value of our mathematical program when $f = \max$ agrees with the makespan of optimal preemptive schedules as derived in [6, 12]. In Section 4.2 we apply our analysis to the ℓ_p-norm,

$$f(\lambda_1, \dots, \lambda_m) = \left(\sum_{j=1}^{m} \lambda_j^p \right)^{1/p} \quad , \quad 1 \le p \le \infty \ , \tag{4}$$

that was studied in the past in the non-preemptive setting [1, 5]. More specifically, we characterize the solution of the corresponding mathematical program when f is as in (4), Section 4.2.1, and offer a polynomial time algorithm to solve it, Section 4.2.2. In Section 4.3 we continue to explore the threshold cost function,

$$f(\lambda_1, \dots, \lambda_m) = \sum_{j=1}^{m} \max(\lambda_j, c) \ , \tag{5}$$

where $c > 0$ is some constant threshold. This target function was also studied in the past for non-preemptive scheduling [2, 3, 4]. Finally, in Section 4.4, we show that an algorithm due to Hochbaum and Shanthikumar [7] may be applied in order to solve the mathematical program in a 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 ℓ_p-norm target function, (4), with $p < \infty$, and the threshold target function, (5), are separable, the algorithms that we offer for these cases are simpler and more efficient than the general algorithm in [7].

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 [8, 9]. The papers [1, 5] offer approximations schemes for a wide class of target functions, including the ℓ_p-norms.

2 Properties of optimal schedules

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

Proposition 2.1 *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two sets of m machines each, $\mathcal{M}_i = \{M_{i,j}\}_{1\le j\le m}$, $i = 1, 2$, with speeds $1 = s_{i,1} \geq \cdots \geq s_{i,m} > 0$, $i = 1, 2$. Assume that the machines in $\mathcal{M}_1$ are no faster than the corresponding machines in $\mathcal{M}_2$, namely, $s_{1,j} \le s_{2,j}$ for all $1 \le j \le m$. Then for any input set of jobs $\mathcal{J}$, and for every monotone target function f, (2), $f_{opt,2} \le f_{opt,1}$, where $f_{opt,i}$ is the optimal f-value for schedules of $\mathcal{J}$ on $\mathcal{M}_i$, $i = 1, 2$.*

Proof. Let $\boldsymbol{\sigma}_1$ be an optimal schedule of $\mathcal{J}$ on $\mathcal{M}_1$. Let $\boldsymbol{\sigma}_2$ be the corresponding schedule of $\mathcal{J}$ on $\mathcal{M}_2$ in the following sense: if $\boldsymbol{\sigma}_1$ scheduled $J_{\eta_{j,k}}$ to run on $M_{1,j}$ during time interval $[\tau_{j,k-1}, \tau_{j,k})$, then $\boldsymbol{\sigma}_2$ schedules the same job to run on $M_{2,j}$ during the same time interval. In case $s_{1,j} < s_{2,j}$, $M_{2,j}$ will be idle during a fraction of $1-\frac{s_{1,j}}{s_{2,j}}$ in each such interval. Concentrating on the last time interval on $M_{2,j}$, $1 \le j \le m$, we see that the time in which it finishes working in $\boldsymbol{\sigma}_2$, $\lambda_{2,j}$, is no later than $\lambda_{1,j}$, the time in which it finishes working in $\boldsymbol{\sigma}_1$. Since f is monotonically non-decreasing, we conclude that $f_{opt,2} \le f_{opt,1}$. $\Box$

Comment. We assume throughout the paper that the number of machines, m, is no larger than the number of jobs, n. Proposition 2.1 implies that if $m > n$, an optimal solution simply ignores the $m - n$ slowest machines.

Proposition 2.2 *There exist optimal schedules in which the loads λ_j are monotonically non-increasing.*

Proof. We start by showing that we may always place the smallest load on the slowest machine, $\lambda_m = \min_{1\le j\le m} \lambda_j$, without increasing the value of the target function. Assume a schedule in which the smallest load is λ_j for some $1 \le j < m$. Then, by Proposition 2.1, we may only improve the value of the target function if we rearrange the post-λ_j schedule to use the $m - 1$ fastest machines, $\{M_1, \ldots, M_{m-1}\}$, rather than using the $m - 1$ machines in $\mathcal{M} \setminus \{M_j\}$ (here we rely upon the symmetry of f). Arguing along the same lines, we may show that the next smallest load could be placed on M_{m-1} and so forth. $\Box$

Proposition 2.3 *There exist optimal schedules with no holes (i.e., no idle times on a machine after which it resumes processing). Namely, if the last time interval in every machine is always non-idle, in the sense that $\eta_{j,k_j} \in \{1,\dots,n\}$ for all $1 \le j \le m$, there exist optimal schedules in which $\eta_{j,k} \neq 0$ for all $1 \le j \le m$ and $1 \le k \le k_j$.*

Proof. Let $\boldsymbol{\sigma}$ be an optimal schedule. For every k, $0 \le k \le m-1$, we let $T_k = T_k(\boldsymbol{\sigma})$ denote the first time in which the number of unfinished jobs equals k (namely, it is the $(n-k)$th time in which a job completed its process). Setting $T_m = 0$, we have $T_m \le T_{m-1} \le \cdots \le T_0$.

First, we show that $\boldsymbol{\sigma}$ may be modified into a schedule $\boldsymbol{\sigma}^{(1)}$ that has no holes during $[T_m, T_{m-1})$ and $f(\boldsymbol{\sigma}^{(1)}) \le f(\boldsymbol{\sigma})$. Assume that such holes exist in $\boldsymbol{\sigma}$. Such a hole takes the form of a time interval $[\tau_{j,k-1}, \tau_{j,k}) \subset [T_m, T_{m-1})$ during which M_j is idle. Let $\tau^* \in (\tau_{j,k-1}, \tau_{j,k})$ be the first time in which a preemption occurred in any machine during this time interval (if there are no preemptions at all during that time interval we take $\tau^* = \tau_{j,k}$). Then the set of (no more than) $m-1$ jobs that are running on the other machines during $[\tau_{j,k-1}, \tau^*)$ remains unchanged. Since during $[T_m, T_{m-1})$ there are at least m jobs that are still not complete, we select one of them that is not running during $[\tau_{j,k-1}, \tau^*)$ and schedule it to run during that time interval (or at least during part of it, if that job may complete its processing on M_j in less than $\tau^* - \tau_{j,k-1}$ time units). After doing so, we recalculate T_k, $1 \le k \le m$, since they may decrease in wake of such a reschedule. If the modified $[T_m, T_{m-1})$ still has holes, we repeat the same procedure until we reach a schedule $\boldsymbol{\sigma}^{(1)}$ in which $[T_m, T_{m-1})$ is free of holes. Obviously, as the loads in $\boldsymbol{\sigma}^{(1)}$ are no larger than the corresponding loads in $\boldsymbol{\sigma}$, we conclude that $f(\boldsymbol{\sigma}^{(1)}) \le f(\boldsymbol{\sigma})$, i.e., $\boldsymbol{\sigma}^{(1)}$ is also optimal.

After this is accomplished, we may apply the same process to all time intervals $[T_k, T_{k-1})$, for $k = m-1, \dots, 1$. During $[T_k, T_{k-1})$ there are exactly k unfinished jobs. In view of proposition 2.2, we may assume that they are all scheduled to run on the k faster machines, $\{M_j\}_{j=1}^{k}$, while the slower $m-k$ machines already exhausted their load. If there are holes in any of the schedules in the k faster machines, we may always fill them up, as we did earlier. This way, we get a schedule $\boldsymbol{\sigma}^{(m+1-k)}$, where $\boldsymbol{\sigma}^{(m+1-k)}$ has no holes until time T_{k-1} and it is optimal. Eventually, we arrive at $\boldsymbol{\sigma}^{(m)}$ that is optimal and has no holes. $\Box$

Hereinafter we concentrate only on optimal schedules that comply with Propositions 2.2 and 2.3. We define the weight on M_j as

$$\mu_j = s_j \lambda_j \ . \tag{6}$$

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 (note that prior to Proposition 2.3 there could have been holes in the schedule and then the weight might not have been related to the load through (6)). We also define the following:

$$W_k = \begin{cases} \sum_{j=1}^{k} w_j & 1 \le k \le m-1 \\ \\ \sum_{j=1}^{n} w_j & k = m \end{cases} \ . \tag{7}$$

With these definitions, we state the following key proposition.

Proposition 2.4 *In optimal schedules that comply with Propositions 2.2 and 2.3*

$$\sum_{j=1}^{k} \mu_j \geq W_k \quad , \quad 1 \leq k \leq m-1 \tag{8}$$

while

$$\sum_{j=1}^{m} \mu_j = W_m \ . \tag{9}$$

Proof. As (9) is just the completeness requirement, we focus on (8) and prove it for an arbitrary value of $1 \leq k \leq m-1$. In view of Proposition 2.2, the entire schedule is embedded in the time interval $[0, \lambda_1)$. We break up this interval into a disjoint union $[0, \lambda_1) = \bigcup_{\ell=0}^{k} R_\ell$ where R_ℓ is the union of all time intervals in which exactly ℓ of the k largest jobs are running (recall that parallelization is forbidden so R_ℓ is defined properly). Proposition 2.2 implies that

$$\begin{array}{rcl} R_k & \subset & [0, \lambda_k) \\ R_k \cup R_{k-1} & \subset & [0, \lambda_{k-1}) \\ & \vdots & \\ R_k \cup \cdots \cup R_1 & \subset & [0, \lambda_1) \end{array} \ . \tag{10}$$

Let r_ℓ denote the amount of work that was done on the k largest jobs during R_ℓ. Then, as the schedule is complete,

$$\sum_{\ell=1}^{k} r_\ell = W_k \ . \tag{11}$$

On the other hand, since the schedule is legal, r_ℓ may not exceed the duration of R_ℓ times the sum of speeds of the ℓ fastest machines,

$$r_\ell \leq S_\ell \cdot |R_\ell| \quad , \quad S_\ell = \sum_{j=1}^{\ell} s_j \ . \tag{12}$$

Hence, by (11), (12) and (10),

$$W_k \leq \sum_{\ell=1}^{k} \left(\sum_{j=1}^{\ell} s_j \right) \cdot |R_\ell| = \sum_{j=1}^{k} s_j \cdot \left(\sum_{\ell=j}^{k} |R_\ell| \right) \leq \sum_{j=1}^{k} s_j \lambda_j = \sum_{j=1}^{k} \mu_j \ .$$

$\square$

Finally, we state our main result.

Theorem 2.5

$$f_{opt} = \min_{\Omega} f\left(\frac{\mu_1}{s_1}, \ldots, \frac{\mu_m}{s_m} \right) \tag{13}$$

where $\Omega \subset (\mathbb{R}^+)^m$ is the bounded polyhedron of all nonnegative weights μ_j that satisfy the legality and completeness constraints (8)+(9).

Proof. Let f_{min} denote the minimum of the optimization problem (13) under constraints (8)+(9) (This optimization problem is a mathematical program to which we refer henceforth as MP). f_{min} is well defined since f is convex and Ω, the domain in which the minimum is sought, is closed and convex. Proposition 2.4 imply that $f_{opt} \geq f_{min}$. Since Algorithm 3.1 in the next section produces a complete and legal preemptive schedule with weights $\{\mu_j\}_{1\leq j\leq m}$ for *any* $(\mu_1, \ldots, \mu_m) \in \Omega$, we infer that $f_{opt} = f_{min}$. $\square$

We conclude this section with two notes on the properties of the target function:

• A note on the symmetry assumption: If the target function is not symmetric, most of the properties of optimal solutions on which we relied do not hold any longer. As an example, consider a problem with two machines with speeds $s_1 = 1$ and $s_2 = 1/2$, one job of weight $w_1 = 5$ and the target function is

$$f(\lambda_1, \lambda_2) = \max(\lambda_1, \lambda_2/4) \ .$$

It is not hard to see that an optimal schedule in this case is to run a part of weight 2 of the input job on M_1 in time slot $[0, 2)$ and then run the complementary part of weight 3 on M_2 in time slot $[2, 8)$. This optimal solution has a "hole" in M_2 and the loads are not monotone, $\lambda_1 = 2 < \lambda_2 = 8$. Therefore, asymmetry might require a different approach.

• A note on the convexity and monotonicity assumptions: If the function f is strictly monotonically increasing with respect to each of its arguments and is also strictly convex, it may be shown that Propositions 2.2 and 2.3 apply to *all* optimal schedules (i.e., filling up holes and arranging the loads so that $\lambda_j > \lambda_k$ whenever $s_j > s_k$ and $\lambda_j = \lambda_k$ if $s_j = s_k$, always improve the value of the target function). As a consequence, Proposition 2.4 applies to all optimal schedules. We note that the ℓ_p-norms are strictly monotone and strictly convex for all $1 < p < \infty$. The ℓ_1-norm is strictly monotone, but it is convex only in the weak sense; the ℓ_∞-norm, on the other hand, is neither strictly convex nor strictly monotone. Indeed, it is easy to construct examples with optimal schedules for the ℓ_1-norm that fail to comply with Proposition 2.2, and examples with optimal schedules for the ℓ_∞-norm that fail to comply with both Propositions 2.2 and 2.3.

3 An optimal scheduling algorithm for a general target function

3.1 The algorithm

Let $\{\mu_j\}_{1\leq j\leq m} \in \Omega$ be a set of nonnegative weights that satisfy conditions (8) and (9). Let

$$\lambda_j = \frac{\mu_j}{s_j} \quad , \quad \Lambda_j = \sum_{k=1}^{j} \lambda_k \quad , \quad 1 \leq j \leq m \quad \text{and} \quad \Lambda_0 = 0 \ . \tag{14}$$

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

$$\Psi(x) = \sum_{j=1}^{m} s_j \cdot \chi_{[\Lambda_{j-1}, \Lambda_j)}(x) \quad \text{where} \quad \chi_I(x) = \left\{ \begin{array}{ll} 1 & x \in I \\ 0 & x \notin I \end{array} \right. \ , \tag{15}$$

a timing function

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

and an indicator function

$$\Gamma(x) = \begin{cases} j & \text{if } x \in [\Lambda_{j-1}, \Lambda_j) \text{ for some } 1 \le j \le m \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

(see Figures 1-3 in Appendix A). The function Ψ represents initially the potential work of the m machines, assuming the loads λ_j. Algorithm 3.1 produces a preemptive schedule $\boldsymbol{\sigma}$ of $\mathcal{J}$ on $\mathcal{M}$ such that the weight on machine M_j equals μ_j.

Algorithm 3.1

1. *Initialize Ψ, Θ and Γ according to (14)-(17).*
2. $i = 1$ *(current job number).*
3. *Define $End(a)$ for all $a \in [0, \Lambda_m)$ in the following manner:*

$$End(a) = \min(a + \Lambda_j - \Lambda_{j-1}, \Lambda_{j+1}) \quad \forall a \in [\Lambda_{j-1}, \Lambda_j) \quad , \quad 1 \le j \le m \ , \tag{18}$$

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

$$\int_a^{b=End(a)} \Psi(x)dx = w_i \ . \tag{19}$$

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

$$[a, b) = \bigcup_{r=1}^{\ell} [a_{r-1}, a_r) \tag{20}$$

   where $a_0 = a$, $a_\ell = b$, Γ 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*

$$w_{i,r} = \int_{a_{r-1}}^{a_r} \Psi(x)dx \quad , \quad 1 \le r \le \ell \ . \tag{21}$$

7. *Break up J_i into ℓ 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 Ψ, Θ and Γ. More specifically, apply on all three functions the following operator:*

$$U_{[a,b)}\Phi := \Phi \cdot \chi_{[0,a)} + L_{b-a}\left\{\Phi \cdot \chi_{[b,\infty)}\right\} \ , \tag{22}$$

   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 Θ and set Λ_j, $1 \le j \le m$, to be the corresponding jth discontinuity.*
11. $i = i + 1$.
12. *If $i > n$ stop. Else go to Step 3.*

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 ith 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 ith 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 $\Lambda^i_0 = 0$), and $End^i(a)$ is the function that is defined in (18) at the ith round. We also define $\Omega^i = [0, \Lambda^i_{m^i})$ to be the support of the state functions in round i, $\Omega^i_j = [\Lambda^i_{j-1}, \Lambda^i_j)$, $1 \le j \le m^i$, be the decomposition of Ω^i into intervals of continuity of $\Theta^i(x)$, and $\lambda^i_j = |\Omega^i_j|$.

Lemma 3.2 *(i) The timing function is linear on each continuity interval, i.e.,*

$$\Theta^i(x) = \begin{cases} x - \Lambda^i_{j-1} & \textit{if } x \in \Omega^i_j \textit{ for some } 1 \le j \le m^i \\ 0 & \textit{otherwise} \end{cases} . \tag{23}$$

(ii) λ^i_j, $1 \le j \le m^i$, form a non-increasing sequence for all i.
(iii) The potential function, $\Psi^i(x)$, is monotonically non-increasing.

Proof. The statement clearly holds when $i = 1$, in view of (15)–(16) and Proposition 2.2. Moreover, statement (iii) is obviously true in all rounds since the cut-and-shift operator, $U_{[a,b)}$, leaves Ψ^i monotonic non-increasing. Hence, we may concentrate on the first two statements and we proceed, by induction, to show that if they hold in the ith round they must hold in the $(i+1)$th round as well.

There are three cases to consider, according to the position of a that is selected in Step 4 in the ith round:

1. $a \in \Omega^i_j$, $j < m^i$, (namely, a does not fall in the last interval) and $End(a) = a + \lambda^i_j$.
2. $a \in \Omega^i_j$, $j < m^i$, and $End(a) = \Lambda^i_{j+1}$.
3. $a \in \Omega^i_{m^i}$, whence $End(a) = \Lambda^i_{m^i}$.

In the first case, the modification of Θ^i by means of the cut-and-shift operator $U_{[a,b)}$, Step 9, creates a continuous linear segment in Θ^{i+1}, Ω^{i+1}_j, out of two neighboring segments in Θ^i, Ω^i_j and Ω^i_{j+1} (the continuity stems from the fact that $\Theta^i(x+\lambda^i_j) = \Theta^i(x)$ for all $x \in [\Lambda^i_{j-1}, \Lambda^i_{j-1}+\lambda^i_{j+1})$; see Figure 3 and compare to Figure 6 and then to Figure 9). The form of Θ^{i+1} is as in (23). Moreover, $\lambda^{i+1}_j = \lambda^i_{j+1}$ in this case. Therefore, since the lengths of all other intervals remain unchanged, the new sequence of lengths is still monotone, $\lambda^{i+1}_1 \ge \cdots \ge \lambda^{i+1}_{m^{i+1}}$. We note that in this case the number of intervals decreases by one, $m^{i+1} = m^i - 1$.

In the second case we also have $m^{i+1} = m^i - 1$. Here, however, if $a \in \Omega^i_j$, then the interval Ω^i_{j+1} disappears in the next stage, while the preceding interval Ω^i_j will be shortened into $\Omega^{i+1}_j = [\Lambda^{i+1}_{j-1}, \Lambda^{i+1}_j)$ where the left end point remains the same, $\Lambda^{i+1}_{j-1} = \Lambda^i_{j-1}$, while the right end point is modified into $\Lambda^{i+1}_j = a < \Lambda^i_j$. All other intervals remain the same. $\Theta^{i+1}(x)$ is still of the form (23). Moreover, the length of the newly formed interval, λ^{i+1}_j, is smaller than the length of the jth interval in round i, and therefore it is smaller than all intervals to the left.

On the other hand, it is longer than the old interval that disappeared, Ω^i_{j+1}, and, consequently, longer than all intervals that remain to the right. Hence, also in this case the monotonicity of the interval lengths is preserved.

In the last case, there are two possibilities: either a falls in the interior of $\Omega^i_{m^i}$, i.e., $a > \Lambda^i_{m^i-1}$, or $a = \Lambda^i_{m^i-1}$. In the first case, the number of intervals remains unchanged, $m^{i+1} = m^i$, and all intervals remain the same apart from the last one that is shortened to $\Omega^{i+1}_{m^{i+1}} = [\Lambda^i_{m^i-1}, a)$. In the second case, the last interval vanishes altogether and $m^{i+1} = m^i - 1$; here also, all other intervals remain the same. In both cases, Θ^{i+1} is still as described in (23) and the sequence of lengths remains non-increasing. $\square$

Lemma 3.3 *Define the sliding window potential function*

$$\hat{\Psi}(a) = \int_a^{End(a)} \Psi(x)dx \ . \tag{24}$$

Then $End(a)$ and $\hat{\Psi}(a)$ are both continuous, where the former is non-decreasing while the latter is non-increasing.

Proof. (Here, since we do not use induction, we omit the superscript i.) $End(a)$ is clearly continuous in the interior of each interval Ω_j. Let Λ_j be the transition point between Ω_j and Ω_{j+1}. Then, by (18) and Lemma 3.2-(ii),

$$End(\Lambda_j-) = \min(\Lambda_j + \lambda_j, \Lambda_{j+1}) = \min(\Lambda_{j+1} + \lambda_j - \lambda_{j+1}, \Lambda_{j+1}) = \Lambda_{j+1} \ .$$

On the other hand,

$$End(\Lambda_j+) = \min(\Lambda_j + \lambda_{j+1}, \Lambda_{j+2}) = \min(\Lambda_{j+1}, \Lambda_{j+2}) = \Lambda_{j+1} \ .$$

Hence, $End(a)$ is continuous. Since it is monotone non-decreasing in the interior of each interval Ω_j, it is monotone non-decreasing along its entire domain of definition $[0, \Lambda_m)$.

The definition of $\hat{\Psi}(a)$ as a sliding window integral of a piecewise continuous function, where the window edges, a and $End(a)$, vary continuously, imply that it is also a continuous function. It is non-increasing, as can be seen by differentiating (24),

$$\frac{d\hat{\Psi}(a)}{da} = \Psi(End(a)) \cdot End'(a) - \Psi(a) \ . \tag{25}$$

In the domains where $End(a)$ is determined by the first argument in the minimum in (18), its derivative is 1. Therefore, as $End(a) > a$ and Ψ is non-increasing, Lemma 3.2-(iii), we get

$$\frac{d\hat{\Psi}(a)}{da} = \Psi(End(a)) - \Psi(a) \leq 0 \ .$$

In the domains where $End(a)$ is determined by the second argument in the minimum, $End'(a) = 0$. Consequently, since $\Psi \geq 0$, $\frac{d\hat{\Psi}(a)}{da}$ turns out to be non-positive there as well. $\square$

Assume that round i^* was the first round in which we selected in Step 4 a value of a with $End^{i^*}(a) = \Lambda^{i^*}_{m^{i^*}}$ (namely, this is the first round in which the sliding window went all the way to the right end point of the current support, $[0, \Lambda^{i^*}_{m^{i^*}})$, 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., $End^i(a) = \Lambda^i_{m^i}$ for all $i \geq i^*$. 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 3.4 *During Phase 1 the number of Θ^i-continuity intervals always decreases by one. Namely, for all i, $1 \leq i \leq i^*$, $m^i = m - i + 1$. Consequently, Phase 1 lasts no more than $m - 1$ rounds.*

Proof. Referring to the three cases that were discussed in the proof of Lemma 3.2, we note that during Phase 1 we are always in case 1, or in case 2 with $j < m^i - 1$. In either case, the value of m decreases in such rounds by 1, as shown there. Consequently, if Phase 1 covers the first $m - 1$ rounds, then during the mth round the number of continuity intervals in Θ^m is one. The definition of the function $End(a)$ implies that this round must therefore mark the beginning of Phase 2. □

Next, we turn our attention to the following important proposition.

Proposition 3.5 *In all rounds, the set of values of a that satisfy requirement (19) in Step 4 of the algorithm is nonempty and it has a maximum.*

The following sequence of lemmas provides a proof for this proposition.

Lemma 3.6 *If Proposition 3.5 holds for the first $i-1$ rounds, where i is any value in the range $1 \leq i \leq m$, then*

$$\int_{\Omega^i} \Psi^i(x)dx = \sum_{j=1}^{m} \mu_j - W_{i-1} \ ; \tag{26}$$

i.e., the total potential during the ith round equals the initial total potential minus the sum of weights of the first $i - 1$ jobs, W_{i-1}, (7).

Proof. In each round i we identify an interval $[a, End(a))$ along which the integral of the potential function Ψ^i equals w_i, (19), and then update Ψ^i into Ψ^{i+1} by extracting that interval, Step 9. Since the total potential of Ψ^1 is $\sum_{j=1}^{m} \mu_j$, see (14) and (15), equality (26) follows. □

Lemma 3.7 *If in round i*

$$\hat{\Psi}^i(0) \geq w_i \ , \tag{27}$$

where $\hat{\Psi}^i$ is defined in (24), the assertion of Proposition 3.5 holds in that round.

Proof. (The superscript i is omitted.) The definition of $End(\cdot)$ and $\hat{\Psi}(\cdot)$ imply that

$$\hat{\Psi}(\Lambda_m) = 0 \ . \tag{28}$$

As $\hat{\Psi}(\cdot)$ is continuous, Lemma 3.3, we infer by (27) and (28) that there exists at least one value of a for which (19) holds. As $\hat{\Psi}(\cdot)$ is also monotone non-increasing, there is either a unique a that satisfies (19), or a closed interval $[a_{\min}, a_{\max}]$ from which the algorithm picks the largest value $a = a_{\max}$. $\square$

Lemma 3.8 *Let*

$$\mu_j^i = \int_{\Omega_j^i} \Psi^i(x)dx \quad , \quad 1 \le j \le m^i \tag{29}$$

denote the integrals of the potential function in the ith round, $\Psi^i(x)$, along the intervals of continuity of $\Theta^i(x)$. Then

$$\sum_{j=1}^{k} \mu_j^i \ge \sum_{j=i}^{i+k-1} w_j \quad , \quad 1 \le k \le m^i - 1 \tag{30}$$

and

$$\sum_{j=1}^{m^i} \mu_j^i = \sum_{j=i}^{n} w_j \ . \tag{31}$$

During the ith round there are m^i "virtual machines" that correspond to the m^i continuity intervals of Θ^i. Those continuity intervals generalize the concept of the DPSs (*Disjoint Processor Systems*) of [6, 12]. The lemma states that the initial situation where machine weight prefixes dominate job weight prefixes, while the total sum of machine weights equals the total sum of job weights, as described in Proposition 2.4, is preserved in all rounds.

Proof. The statement is obviously true for $i = 1$ since then it agrees with Proposition 2.4. We proceed by induction to prove it for the $(i+1)$th round, assuming that it holds for the ith round. To avoid too many indices we denote all entities of the ith round with no superscript, while the entities in the subsequent round will be denoted by an apostrophe. To further simplify the notations, we concentrate on the transition from the first round to the second one, i.e., $i = 1$.

First, we assume that the first round was in Phase 1, so that the number of intervals in the second round is $m' = m - 1$. So, given that

$$\mu_1 + \cdots + \mu_k \ge w_1 + \cdots + w_k \quad , \quad 1 \le k \le m - 1 \quad \text{and} \quad \mu_1 + \cdots + \mu_m = w_1 + \cdots + w_n \ , \tag{32}$$

we need to show that if μ'_j, $1 \le j \le m' = m - 1$, are the corresponding weights in the second round, (29), then

$$\mu'_1 + \cdots + \mu'_k \ge w_2 + \cdots + w_{k+1} \quad , \quad 1 \le k \le m - 2 \quad \text{and} \quad \mu'_1 + \cdots + \mu'_{m-1} = w_2 + \cdots + w_n \ . \tag{33}$$

Let us assume that the first job was scheduled on an interval $[a, End(a))$ where $a \in \Omega_j$ for some $1 \le j \le m - 1$ ($j < m$ in view of our assumption that the first round was in Phase 1). Then after the update of the state functions, Step 9, the first $j - 1$ weights are not effected,

$$\mu'_k = \mu_k \quad , \quad 1 \le k \le j - 1 \ . \tag{34}$$

The new jth weight equals the sum of the previous jth and $(j+1)$th weights, minus w_1 (which equals the integral of Ψ^i over the interval $[a, End(a))$ that was extracted),

$$\mu'_j = \mu_j + \mu_{j+1} - w_1 \ . \tag{35}$$

The remaining weights in the new round are obtained by a left shift of the remaining weights in the previous round,

$$\mu'_k = \mu_{k+1} \quad , \quad j+1 \le k \le m-1 \ . \tag{36}$$

When $k < j$ the corresponding inequality in (33) holds due to (34), (32) and the monotonicity of the job weights,

$$\mu'_1 + \cdots + \mu'_k = \mu_1 + \cdots + \mu_k \ge w_1 + \cdots + w_k \ge w_2 + \cdots + w_{k+1} \ .$$

If $j \le k \le m-2$, the corresponding inequality in (33) holds due to (34)–(36) and (32),

$$\mu'_1 + \cdots + \mu'_k = \mu_1 + \cdots + \mu_{k+1} - w_1 \ge w_2 + \cdots + w_{k+1} \ .$$

The equality in (33) is proved similarly. This completes the proof for $i = 2$ if the first round was in Phase 1. If the first round was in Phase 2 the proof is very similar and should take into account the two possible cases: either $m' = m$ (if $a > \Lambda_{m-1}$) or $m' = m-1$ (if $a \le \Lambda_{m-1}$ and $End(a) = \Lambda_m$); we omit further details. □

Proof of Proposition 3.5. Lemma 3.8 implies that in all rounds

$$\hat{\Psi}^i(0) = \int_{\Omega_1^i} \Psi^i(x)dx = \mu_1^i \ge w_i \quad , \quad 1 \le i \le n \ .$$

Hence, Proposition 3.5 holds in view of Lemma 3.7. □

Theorem 3.9 *Algorithm 3.1 generates a complete and legal schedule. Moreover, the number of preemptions that are enforced by the algorithm is bounded by* $2(m-1)$.

Proof. The algorithm is well defined in view of Lemma 3.2 and Proposition 3.5. 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 $[a, End(a))$.

Next, we turn our attention to the number of preemptions. We prove that the number of segments in the schedule, $\sum_{j=1}^m k_j$ (see Definition 1.1), is bounded by $n + 2(m-1)$. Let Q_i denote the number of segments in the schedule after the assignment of the first i jobs. Then we aim at showing that $Q_n \le n + 2(m-1)$. Initially, $Q_0 = m$ since in each machine there is exactly one idle segment of duration μ_j/s_j. During Phase 1 in the algorithm, the overall number of segments may increase by no more than 2. Indeed, going back to Step 5 in the algorithm and to the decomposition (20) of the interval $[a, End(a))$, we note that the assignment of such a job may increase the number of segments only in the first and last machines, M_{j_1} and M_{j_ℓ}, while in the intermediate machines, $\{M_{j_r}\}_{1<r<\ell}$, the number of segments remains the same. During Phase 2, however, the number of segments may increase by 1 at the most. More specifically, if the interval assigned to the job is exactly the last interval of discontinuity of Θ^i, i.e., $[a, End(a)) = [\Lambda^i_{m^i-1}, \Lambda^i_{m^i})$, the number of segments does not increase; if, on the other hand, $a > \Lambda^i_{m^i-1}$ or $a < \Lambda^i_{m^i-1}$, the number of segments will increase by 1. Note that in the last round a must equal 0, as implied by condition (9). In view of all of the above, if t is the number of rounds in Phase 1, $Q_m \le Q_0 + 2t + (n - t - 1) = m + t + (n - 1)$. Since $t \le m - 1$, Lemma 3.4, we conclude that $Q_m \le m + (m - 1) + n - 1 = n + 2(m - 1)$. □

We note that this number of segments, $n + 2(m - 1)$, was shown to be minimal for some inputs for the makespan minimization problem [6].

In view of all of the above, we arrive at our final statement regarding Algorithm 3.1.

Theorem 3.10 *Algorithm 3.1 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 (13) under constraints (8)+(9).*

3.2.1 A semi-online version

Here we show that our algorithm works even when the jobs are not ordered according to a non-increasing job weight. We describe herein all the necessary modifications that need to be made in order to prove that also the semi-online version of the algorithm works.

When the jobs are not ordered, the terms *Phase 1* and *Phase 2* are no longer suitable. Instead, one should speak of rounds of *Type 1* and *Type 2*. Round number i is a round of Type 1 if the value of a that was selected in Step 4 was in the range $[0, \Lambda^i_{m^i-1}]$. Otherwise, if $a \in (\Lambda^i_{m^i-1}, \Lambda^i_{m^i})$, the round is referred to as a round of Type 2. When the jobs are ordered, all rounds of Type 1 (if any) occur first, and once a round of Type 2 occurs, all subsequent rounds are also of Type 2. This enabled the distinction between Phase 1 and Phase 2. However, when the jobs are not ordered, the two types of rounds may be interleaved. Hence, Lemma 3.4 should be modified as follows:

Lemma 3.11 *In each round of Type 1, the number of Θ^i-continuity intervals decreases by one. Consequently, there are no more than $m-1$ rounds of Type 1.*

The next lemma that needs to be modified is Lemma 3.8. Here is its modification:

Lemma 3.12 *For every round number i, $1 \le i \le n$, let $\{w^i_j\}_{1\le j\le n+1-i}$ denote the sequence of job weights that were still not scheduled thus far, ordered so that $w^i_1 \ge w^i_2 \ge \cdots \ge w^i_{n+1-i}$. Furthermore, let μ^i_j denote the integrals of the potential function in the ith round, $\Psi^i(x)$, along the intervals of continuity of $\Theta^i(x)$, see (29). Then*

$$\sum_{j=1}^{k} \mu^i_j \ge \sum_{j=1}^{k} w^i_j \quad , \quad 1 \le k \le m^i - 1 \tag{37}$$

and

$$\sum_{j=1}^{m^i} \mu^i_j = \sum_{j=1}^{n+1-i} w^i_j \ . \tag{38}$$

Proof. The statement is obviously true for $i = 1$ since then it agrees with Proposition 2.4. We proceed by induction to prove it for the $(i+1)$th round, assuming that it holds for the ith round. To that end, we denote the weight of the job that was scheduled in the ith round by w^i_s, where $1 \le s \le n+1-i$.

There are two cases to consider: either the ith round was of Type 1 or it was of Type 2. If it was of Type 1, the number of intervals in the $(i+1)$th round is $m^{i+1} = m^i - 1$ and there exists ℓ, $1 \le \ell < m^i$, such that

$$\begin{array}{lll} \mu^{i+1}_j = \mu^i_j & \quad 1 \le j \le \ell - 1 & \\ \mu^{i+1}_\ell = \mu^i_\ell + \mu^i_{\ell+1} - w^i_s & & . \\ \mu^{i+1}_j = \mu^i_{j+1} & \quad \ell + 1 \le j \le m^{i+1} & \end{array} \tag{39}$$

In light of (39), the new set of machine weights μ^{i+1}_j, $1 \le j \le m^{i+1}$, and the weights of the remaining job weights,

$$\{w^{i+1}_j\}_{1\le j\le n-i} = \{w^i_j\}_{1\le j\le n+1-i, j\ne s} \ , \tag{40}$$

obviously satisfy the required equality (38). Thus, we concentrate on proving that they satisfy the set of inequalities in (37), i.e., that

$$\sum_{j=1}^{k} \mu_j^{i+1} \ge \sum_{j=1}^{k} w_j^{i+1} \quad , \quad 1 \le k \le m^{i+1} - 1 \ . \tag{41}$$

Those inequalities hold for all $k \le \ell - 1$ because, by (39), (37) and (40),

$$\sum_{j=1}^{k} \mu_j^{i+1} = \sum_{j=1}^{k} \mu_j^{i} \ge \sum_{j=1}^{k} w_j^{i} \ge \sum_{j=1}^{k} w_j^{i+1} \ .$$

As for $\ell \le k \le m^{i+1} - 1$, (39) and (37) imply that

$$\sum_{j=1}^{k} \mu_j^{i+1} = \sum_{j=1}^{k+1} \mu_j^{i} - w_s^i \ge \sum_{j=1}^{k+1} w_j^{i} - w_s^i \ . \tag{42}$$

Now, if $s \le k+1$, the right hand side in (42) equals the sum of the k largest jobs in round $i+1$, i.e., $\sum_{j=1}^{k} w_j^{i+1}$, whence $\sum_{j=1}^{k} \mu_j^{i+1} \ge \sum_{j=1}^{k} w_j^{i+1}$ as required. If, on the other hand, $s > k+1$, we get that

$$\sum_{j=1}^{k} \mu_j^{i+1} \ge \sum_{j=1}^{k} w_j^{i} + (w_{k+1}^i - w_s^i) \ge \sum_{j=1}^{k} w_j^{i} = \sum_{j=1}^{k} w_j^{i+1} \ ;$$

namely, the set of k largest job weights remains unchanged in this case.

If the ith round was of Type 2, the number of intervals in the $(i+1)$th round is $m^{i+1} = m^i$ and

$$\begin{array}{ll} \mu_j^{i+1} = \mu_j^i & \quad 1 \le j \le m^{i+1} - 1 \\ \mu_{m^{i+1}}^{i+1} = \mu_{m^i}^i - w_s^i & \end{array} \ . \tag{43}$$

Also here, the new set of machine weights μ_j^{i+1}, $1 \le j \le m^{i+1}$, and the weights of the remaining job weights, (40), obviously satisfy the required equality (38). Thus, we concentrate on proving that they satisfy (41). Arguing along the same lines as before, this is a straightforward consequence of (43), (37) and (40),

$$\sum_{j=1}^{k} \mu_j^{i+1} = \sum_{j=1}^{k} \mu_j^{i} \ge \sum_{j=1}^{k} w_j^{i} \ge \sum_{j=1}^{k} w_j^{i+1} \ .$$

□

Next, we should modify inequality (27) in Lemma 3.7 into $\hat{\Psi}^i(0) \ge w_1^i$ (namely, the first machine weight in each round, μ_1^i, should be at least as large as the weight of the largest job that was still not scheduled). That, in turn, proves Proposition 3.5. That summarizes all the necessary changes.

3.3 Implementation

Algorithm 3.1 maintains three state functions and, in Step 4, it needs to find a specific value of a out of a continuum of possible values. Hence, it is necessary to demonstrate how such an algorithm, that deals with non-discrete entities, may be implemented efficiently.

The three state functions may be easily represented by vectors of length m that store their discontinuities. To that end, the algorithm maintains the following variables:

1. The variable `mt` that holds the value m^i. It is initialized to `mt`$= m$.
2. The vector $\mathtt{T}[0:\mathtt{m}]$ that holds in round i the discontinuities of the timing function $\Theta^i(x)$, i.e., Λ^i_j, $1 \le j \le m^i$. It is initialized to `T[j]=`Λ_j for all $\mathtt{0} \le \mathtt{j} \le \mathtt{m}$.
3. The vector $\mathtt{G}[0:\mathtt{m}]$ that holds in round i the discontinuities of the indicator function $\Gamma(x)$. It is initialized in the same way as `T[·]`.

We note that the discontinuities of the potential function, $\Psi(x)$, coincide with those of $\Gamma(x)$ (When we talk hereinafter about discontinuities of Ψ we actually mean a transition point from one machine to another, namely, a discontinuity in Γ. Note that if two adjacent machines have the same speed, there is a discontinuity in Γ but not in Ψ; still, we count that point as a discontinuity in our discussion). During the entire execution of the algorithm, the vector `T[·]` will represent the function $\Theta(x)$ in the sense that

$$\Theta(x) = \begin{cases} x - \mathtt{T}[\mathtt{j}-1] & x \in [\mathtt{T}[\mathtt{j}-1], \mathtt{T}[\mathtt{j}]) \quad , \quad 1 \le \mathtt{j} \le \mathtt{mt} \\ 0 & \text{Otherwise} \end{cases} ,$$

see (23). The vector `G[·]`, on the other hand, represents both $\Gamma(x)$ and $\Theta(x)$ as

$$\Gamma(x) = \begin{cases} j & x \in [\mathtt{G}[\mathtt{j}-1], \mathtt{G}[\mathtt{j}]) \quad , \quad 1 \le \mathtt{j} \le \mathtt{m} \\ 0 & \text{Otherwise} \end{cases} ,$$

and

$$\Psi(x) = \begin{cases} s_j & x \in [\mathtt{G}[\mathtt{j}-1], \mathtt{G}[\mathtt{j}]) \quad , \quad 1 \le \mathtt{j} \le \mathtt{m} \\ 0 & \text{Otherwise} \end{cases} . \tag{44}$$

Assume that in a given round we identified an interval $[a, b)$ that should be extracted, see Step 9. Implementing the cut-and-shift operation, (22), on $\mathtt{T}[\cdot]$, $\mathtt{G}[\cdot]$ and `mt` is a straightforward task. Hence, we proceed to explain how to find the appropriate value of a in Step 4. To that end, we construct in each round a third vector that will represent the sliding window function $\hat{\Psi}(a) = \int_a^{End(a)} \Psi(x)dx$, (24). As shown in Lemma 3.3, $\hat{\Psi}$ is continuous and monotonic non-increasing. Its derivative however, (25), is piecewise constant and has discontinuities of three types:

- Type I. Points a in which the left end point of the sliding window, a, is a discontinuity of Ψ.
- Type II. Points a in which the right end point of the sliding window, $End(a)$, is a discontinuity of Ψ.

- Type III. Points a in which End' is discontinuous. End' has discontinuities in every point `T[j], 1` $\leq$ `j` $\leq$ `mt`, and also in internal points of the intervals `(T[j-1],T[j])` in which the two arguments in the minimum in (18) are equal. The discontinuities of the first kind, `T[j]`, are always of Type I as well, since the discontinuities of $\Theta(x)$ are always discontinuities of Ψ too.

There are no more than $m-1$ internal discontinuities of Ψ. Therefore, the number of discontinuities of Type I is no more than $m-1$, and the same holds for discontinuities of Type II, since $End(a)$ is monotone. As for discontinuities of Type III that are not also discontinuities of Type I, there are no more than $m-1$ such points because End' has no more than $m-1$ discontinuities in the interior of the intervals `(T[j-1],T[j]), 1` $\leq$ `j` $\leq$ `m`. In view of all of the above we conclude that $\hat{\Psi}$ is continuous and piecewise linear and it has no more than $3(m-1)$ singular points. Therefore, what we need to do in order to recompute $\hat{\Psi}(\cdot)$ in each round is as follows:

1. Find its set of (no more than $3(m-1)$) singularities.
2. Compute $\hat{\Psi}$ at each of these singularities, at $a=0$ and at $a=$ `G[m]` (in the latter point $\hat{\Psi}$ is always zero, (28)).

Having identified the nodes of $\hat{\Psi}$ and its values at those nodes, we may then easily find (the maximal) point a where $\hat{\Psi}(a)=w_i$, (19), by means of a binary search of w_i in the list of values of $\hat{\Psi}$ at the nodes, followed by a linear interpolation. Hence, we proceed to discuss how we may carry out the above two tasks. The second one is easy: given a value of a it is straightforward to compute $End(a)$, according to $T[\cdot]$ and (18), and then $\hat{\Psi}(a)$, according to (44). It remains to discuss the first task above, namely, finding the singularities of $\hat{\Psi}$. The singularities of Type I are just the points `G[j], 1` $\leq$ `j` $\leq$ `m-1`. The additional singularities of Type III are easily computable from `T[·]`. Finally, the singularities of Type II are just `InvEnd(G[j]), 1` $\leq$ `j` $\leq$ `m-1`, where `InvEnd(·)` is the inverse function of $End(\cdot)$. `InvEnd(·)` is well defined everywhere apart from {`T[j], 1` $\leq$ `j` $\leq$ `mt`}. So we may find all singularities of Type II for which `G[j]` does not coincide with a discontinuity of $\Theta(x)$, `T[k]`. We claim that we may ignore at this stage points `G[j]` that do coincide with some `T[k]`. The reason is that such points give rise to two singularities of Type II: the first one is also a singularity of Type III and the second one is also a singularity of Type I. Therefore, we may ignore such points in our search of Type II singularities because they were already covered in our search for Type I and Type III singularities.

4 Examples of Target Functions

4.1 The makespan

In [6] it is shown that the optimal makespan is

$$f_{opt} := \max_{1\leq k\leq m} q_k \quad \text{where} \quad q_k = \frac{W_k}{S_k} \ , \tag{45}$$

W_k is given in (7) and S_k is the sum of the speeds of the k fastest machines, (12), $1\leq k\leq m$. We continue to prove that f_{opt} is indeed the minimum of MP with $f=\max$. First, we claim that f_{opt}

is a lower bound for the minimum: Let $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_m) \in \Omega$ and let $f_{\boldsymbol{\mu}} = \max\left(\frac{\mu_1}{s_1}, \ldots, \frac{\mu_m}{s_m}\right)$. Then $\mu_j \le f_{\boldsymbol{\mu}} \cdot s_j$ for all $1 \le j \le m$. Invoking (8)+(9), we conclude that

$$W_k \le \sum_{j=1}^{k} \mu_j \le f_{\boldsymbol{\mu}} \cdot \sum_{j=1}^{k} s_j = f_{\boldsymbol{\mu}} \cdot S_k \quad , \quad 1 \le k \le m \ .$$

We infer that $f_{\boldsymbol{\mu}} \ge q_k$ for all $1 \le k \le m$, where q_k are given in (45). This implies that

$$\min_{\boldsymbol{\mu}\in\Omega} f_{\boldsymbol{\mu}} \ge f_{opt} \ . \tag{46}$$

Next, we need to construct a solution $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_m) \in \Omega$ for which $\max\left(\frac{\mu_1}{s_1}, \ldots, \frac{\mu_m}{s_m}\right) = f_{opt}$. Algorithm 4.1 that is presented in the next section constructs such a solution (see at the end of Section 4.2).

4.2 The ℓ_p-norm

Here, we concentrate on the solution of MP where f is as in (4). 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. We begin in Section 4.2.1 with a characterization of optimal solutions of this problem. This characterization provides also a method to compute all optimal solutions. However, the run-time of this method is exponential in m. In Section 4.2.2 we describe a polynomial time algorithm that constructs an optimal solution for the problem. In analyzing that algorithm and proving its correctness, we rely upon some of the results of Section 4.2.1.

4.2.1 Optimal solutions for the ℓ_p-minimization problem

In the mathematical program MP we aim at finding a solution $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_m) \in \Omega$ that minimizes $\sum_{j=1}^{m}\left(\frac{\mu_j}{s_j}\right)^p$. Using (9) to express μ_m as a function of all other arguments, we aim at minimizing

$$g(\mu_1, \ldots, \mu_{m-1}) = \sum_{j=1}^{m-1}\left(\frac{\mu_j}{s_j}\right)^p + \left(\frac{W_m - \mu_1 - \cdots - \mu_{m-1}}{s_m}\right)^p \ , \tag{47}$$

in the domain

$$\Omega' = \left\{(\mu_1, \ldots, \mu_{m-1}) \in (\mathbb{R}^+)^{m-1} \ : \ \exists \mu_m \in \mathbb{R}^+ \text{ such that } (\mu_1, \ldots, \mu_{m-1}, \mu_m) \in \Omega\right\} \ . \tag{48}$$

Differentiating with respect to each of the $m-1$ variables we find out that the minimum occurs when

$$\frac{\partial g}{\partial \mu_j} = \frac{p\mu_j^{p-1}}{s_j^p} - \frac{p(W_m - \mu_1 - \cdots - \mu_{m-1})^{p-1}}{s_m^p} = 0 \quad , \quad 1 \le j \le m-1 \ ,$$

or

$$\mu_j = \left(\frac{s_j}{s_m}\right)^{p/(p-1)} \cdot (W_m - \mu_1 - \cdots - \mu_{m-1}) \quad , \quad 1 \le j \le m-1 \ . \tag{49}$$

The solution of this set of equations is

$$\mu_j = s_j^{p/(p-1)} \cdot \frac{W_m}{S_p[1:m]} \quad , \quad 1 \le j \le m \ , \tag{50}$$

where hereinafter

$$S_p[a:b] = \sum_{j=a}^{b} s_j^{p/(p-1)} \ . \tag{51}$$

The minimal point (50) may occur outside of Ω'. In that case, the minimum in Ω' is obtained at some point on the boundary $\partial\Omega'$. $\partial\Omega'$ is composed of $2(m-1)$ faces. $m-1$ of those faces are characterized by

$$\sum_{j=1}^{k} \mu_j = W_k \quad , \quad 1 \le k \le m-1 \ . \tag{52}$$

The other $m-1$ faces are characterized by $\mu_j = 0$, $2 \le j \le m$, where $\mu_m = W_m - \sum_{j=1}^{m-1} \mu_j$. From the convexity of the ℓ_p-norm for $1 < p \le \infty$ we may ignore the latter $m-1$ faces and restrict our attention to the first $m-1$ faces. Along the kth face $\mu_k = W_k - \sum_{j=1}^{k-1} \mu_j$ and, consequently, the function g, (47), reduces to a function of $m-2$ variables. Repeating the same computations as before, we find that the minimum along the kth face is obtained at

$$\mu_j = s_j^{p/(p-1)} \cdot \begin{cases} W_k / S_p[1:k] & 1 \le j \le k \\ \\ (W_m - W_k)/S_p[k+1:m] & k+1 \le j \le m \end{cases} \ . \tag{53}$$

In general, the minimum of g along the intersection of t faces, say, $1 \le k_1 < \cdots < k_t \le m-1$, namely, the minimum of g when (8) holds with equality for all $k \in \{k_1, \ldots, k_t\}$ and with a strict inequality for all other values of $1 \le k \le m-1$, is given by

$$\mu_j = s_j^{p/(p-1)} \cdot (W_{k_{i+1}} - W_{k_i})/S_p[k_i+1 : k_{i+1}] \quad , \quad k_i + 1 \le j \le k_{i+1} \ , \tag{54}$$

where $k_0 = 0$, $k_{t+1} = m$, $W_0 = 0$ and $0 \le i \le t$. Note that (54) agrees with (50) when the global minimum is obtained at the interior $\Omega \setminus \partial\Omega$ (i.e., when $t = 0$) and with (53) when it is obtained at the interior of one of the faces of $\partial\Omega$ ($t = 1$). Formula (54) may be used to find the global minimum in Ω using a naïve algorithm by scanning all 2^{m-1} values of $0 \le t \le m-1$ and $\{k_i\}_{1 \le i \le t}$, computing the corresponding minimum by (54), checking if that minimum is in Ω and, among those that are, selecting the minimal one.

4.2.2 A polynomial time algorithm for finding an optimal solution

Here we present a polynomial time algorithm that yields an optimal solution $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_m)$ for MP where the target function is the ℓ_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 Ω and that it is a minimal point in Ω.

Algorithm 4.1

1. *Set $t = 0$ and $k_t = 0$ (at each stage k_t equals the number of values μ_j that were already determined).*
2. *For every $k_t + 1 \le k \le m$, compute*

$$q_k = (W_k - W_{k_t})/S_p[k_t + 1 : k] \ , \tag{55}$$

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

$$\mu_j = s_j^{p/(p-1)} \cdot (W_{k_{t+1}} - W_{k_t})/S_p[k_t + 1 : k_{t+1}] \ . \tag{56}$$

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

Comment. 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 (56), 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 ℓ_1-norm does not distinguish between such assignments. Such schedules are of-course optimal.

Example. Assume $n = 5$ jobs of weights $(w_1, \dots, w_5) = (5, 5, 3, 1, 1)$, $m = 4$ machines of speeds $(s_1, \dots, s_4) = (1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ and $p = \infty$. Then in the first round, the quotients q_k that are evaluated in Step 2 are $q_1 = \frac{5}{1}$, $q_2 = \frac{10}{3/2}$, $q_3 = \frac{13}{2}$ and $q_4 = \frac{15}{5/2}$. The maximum 20/3 is achieved at $k_1 = 2$. So we set in Step 3 $\mu_1 = 20/3$ and $\mu_2 = 10/3$. In the second round, the quotients are $q_3 = \frac{3}{1/2}$ and $q_4 = \frac{5}{1}$. The maximum 6 is obtained at $k_2 = 3$, whence we set in Step 3 $\mu_3 = 3$. The third round is the last. Here we have $q_4 = \frac{2}{1/2}$ so that $k_3 = 4$ and, consequently, $\mu_4 = 2$. The algorithm thus outputs $(\mu_1, \dots, \mu_4) = (20/3, 10/3, 3, 2)$.

Lemma 4.2 *Let $\boldsymbol{\mu} = (\mu_1, \dots, \mu_m)$ be the solution that Algorithm 4.1 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 (8)+(9). Moreover, the set of indices for which (8) holds with equality is exactly $\{k_i\}_{1 \le i \le t}$.*

Proof. We show that for every $0 \le i \le t$,

$$\sum_{j=k_i+1}^{k} \mu_j > W_k - W_{k_i} \quad , \quad k_i + 1 \le k < k_{i+1} \ , \tag{57}$$

and

$$\sum_{j=k_i+1}^{k_{i+1}} \mu_j = W_{k_{i+1}} - W_{k_i} \ . \tag{58}$$

Obviously, (57)+(58) prove all claims of the lemma. Let us fix i in the range $0 \le i \le t$ and prove (57) for that value of i. k_{i+1} was the first index that maximized the quotient $(W_k - W_{k_i})/S_p[k_i + 1 : k]$ among all $k_i + 1 \le k \le m$. Hence,

$$(W_{k_{i+1}} - W_{k_i})/S_p[k_i + 1 : k_{i+1}] > (W_k - W_{k_i})/S_p[k_i + 1 : k] \quad , \quad k_i + 1 \le k < k_{i+1} \ . \tag{59}$$

Consequently, by (56),

$$\sum_{j=k_i+1}^{k} \mu_j = \left(\sum_{j=k_i+1}^{k} s_j^{p/(p-1)} \right) \cdot (W_{k_{i+1}} - W_{k_i}) / S_p[k_i+1 : k_{i+1}] >$$

$$> \left(\sum_{j=k_i+1}^{k} s_j^{p/(p-1)} \right) \cdot (W_k - W_{k_i}) / S_p[k_i+1 : k] = W_k - W_{k_i} \quad , \quad k_i + 1 \le k < k_{i+1} \ .$$

This proves (57). The proof of (58) is similar. □

Next, we claim that $\boldsymbol{\mu}$ is optimal for $1 < p < \infty$ (the case $p = \infty$ is referred to later on).

Lemma 4.3 *Let $\boldsymbol{\mu}' = \{\mu'_j\}_{1 \le j \le m}$ be an optimal solution of* MP *for $1 < p < \infty$. Then $\boldsymbol{\mu}' = \boldsymbol{\mu}$.*

Proof. Let $\{k'_i\}_{1 \le i \le t'}$ be the indices for which the optimal solution $\boldsymbol{\mu}'$ satisfies (8) with equality. Then $\boldsymbol{\mu}'$ is given by (54) with k'_i and t' instead of k_i and t. We continue to show that $\boldsymbol{\mu}'$ coincides with $\boldsymbol{\mu}$ along the first run in $\boldsymbol{\mu}$, i.e.,

$$\mu'_j = \mu_j \quad , \quad 1 \le j \le k_1 \ . \tag{60}$$

The proof for subsequent runs is similar. Our first observation is that, by (54) and (56),

$$\mu'_1 = s_1^{p/(p-1)} \cdot W_{k'_1} / S_p[1 : k'_1] \le s_1^{p/(p-1)} \cdot W_{k_1} / S_p[1 : k_1] = \mu_1 \tag{61}$$

(the inequality in (61) stems from the fact that k_1 was chosen by the algorithm in the first round so as to maximize the quotients $q_k = W_k / S_p[1 : k]$). Moreover, as $\boldsymbol{\mu}'$ satisfies conditions (8)+(9),

$$\sum_{j=1}^{k_1} \mu'_j \ge W_{k_1} = \sum_{j=1}^{k_1} \mu_j \ . \tag{62}$$

Next, assume that (60) does not hold. Then, in view of (62),

$$\mu'_k > \mu_k \quad \text{for some} \quad 1 < k \le k_1 \ . \tag{63}$$

We continue to show that (63)+(61) imply that the solution $\boldsymbol{\mu}'$ may be improved by moving some weight from μ'_k to μ'_1, in contradiction to the optimality of $\boldsymbol{\mu}'$. To that end, define

$$M = \mu_1 + \mu_k \quad , \quad M' = \mu'_1 + \mu'_k \ , \tag{64}$$

and

$$\mu''_i = \alpha_i M' \quad \text{where} \quad \alpha_i = \frac{s_i^{p/(p-1)}}{s_1^{p/(p-1)} + s_k^{p/(p-1)}} \quad , \quad i = 1, k \ . \tag{65}$$

We show below that

$$\mu''_1 > \mu'_1 \quad , \quad \mu''_k < \mu'_k \quad \text{and} \quad \mu''_1 + \mu''_k = \mu'_1 + \mu'_k \ . \tag{66}$$

This will establish the required contradiction: by replacing in $\boldsymbol{\mu}'$ the weights on the first and kth machine, μ'_1 and μ'_k, with the newly defined weights, μ''_1 and μ''_k, we get a different solution that is still legal and it has a smaller ℓ_p-norm since

$$\left(\frac{\mu''_1}{s_1}\right)^p + \left(\frac{\mu''_k}{s_k}\right)^p < \left(\frac{\mu'_1}{s_1}\right)^p + \left(\frac{\mu'_k}{s_k}\right)^p$$

($\boldsymbol{\mu}''$ is still legal because, by (66), we increase the weight on the first machine by some constant and decrease the weight on the kth machine by the same constant, hence, we keep respecting all conditions in (8)+(9)). It thus remains only to prove (66). The equality in (66) is obvious. Regarding the two inequalities, it suffices to prove only one of them. If $M \geq M'$ then, by (65), $\mu''_k = \alpha_k M' \leq \alpha_k M$. But $\alpha_k M = \mu_k$ as implied by our definition of μ_j in the algorithm, (56), along the first run $1 \leq j \leq k_1$. Hence, by (63), we conclude that in this case

$$\mu''_k = \mu_k < \mu'_k \ . \tag{67}$$

If, on the other hand, $M < M'$ then, by (65) and (61),

$$\mu''_1 = \alpha_1 M' > \alpha_1 M = \mu_1 \geq \mu'_1 \ . \tag{68}$$

(66) now follows from (67) and (68). □

Before concluding this section we comment on the optimality for $p = \infty$. We observe that the solution $\boldsymbol{\mu}$ that Algorithm 4.1 outputs satisfies

$$\max_{1\leq k\leq m} \frac{W_k}{S_p[1:k]} = \frac{\mu_1}{s_1^{1+\frac{1}{p-1}}} \geq \frac{\mu_2}{s_2^{1+\frac{1}{p-1}}} \geq \cdots \geq \frac{\mu_m}{s_m^{1+\frac{1}{p-1}}} \ . \tag{69}$$

When $p = \infty$, (69) translates into

$$\max_{1\leq k\leq m} \frac{W_k}{S_\infty[1:k]} = \max_{1\leq k\leq m} \frac{W_k}{S_k} = \frac{\mu_1}{s_1} \geq \frac{\mu_2}{s_2} \geq \cdots \geq \frac{\mu_m}{s_m} \ ;$$

here, as in (12), $S_k = \sum_{j=1}^k s_j$. Therefore,

$$\max\left(\frac{\mu_1}{s_1}, \ldots, \frac{\mu_m}{s_m}\right) = \max_{1\leq k\leq m} \frac{W_k}{S_k} \ ,$$

which, in view of (46)+(45), shows the optimality of this solution.

In addition, we note in passing that (69) strengthens Proposition 2.2 for $p < \infty$ because it implies that

$$\frac{\lambda_1}{s_1^{\frac{1}{p-1}}} \geq \cdots \geq \frac{\lambda_m}{s_m^{\frac{1}{p-1}}} \ .$$

4.3 Threshold cost functions

Here we study the target function

$$f(\mu_1, \ldots, \mu_m) = \sum_{j=1}^m \max\left(\frac{\mu_j}{s_j}, c\right) \ . \tag{70}$$

This case, also known as *extensible bin packing* [2, 3, 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.

We begin with an algorithm to compute an optimal solution $\boldsymbol{\mu} \in \Omega$ to MP when the target function f is as above. Here W_k and S_k are as in (7) and (12).

Algorithm 4.4

```
1. Set $\mu_k = 0$ for all $1 \le k \le m$.
2. Set $W = W_m = \sum_{j=1}^{n} w_j$.
3. If $W \le c \cdot s_1$ set $\mu_1 = W$ and stop.
4. Set
      $\mu_1 = \max\left\{ c \cdot s_1, \max_{1 \le k \le m} (W_k - c \cdot S_k + c \cdot s_1) \right\} \quad , \quad W = W - \mu_1 \ .$   (71)
5. 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 $W = 0$.
```

Lemma 4.5 *The solution that Algorithm 4.4 produces is in Ω.*

Proof. In order to prove completeness, condition (9), we show that if we reach Step 4 then W must be zero at the end of the loop in Step 5 (in fact, it may become zero earlier, and then all μ_k from the next step will be zero). The initialization of μ_1, (71), implies that

$$\mu_1 \ge W_m - c \cdot S_m + c \cdot s_1 \ .$$

Consequently, at the beginning of the loop in Step 5,

$$W \le c \cdot (S_m - s_1) = c \cdot \sum_{j=2}^{m} s_j \ .$$

Hence, if in all $m-1$ rounds of the loop we execute Step 5a, the value of W at the end of the loop is zero. If, on the other hand, we execute in one of the rounds Step 5b instead, then W becomes zero at that point.

As for the legality conditions, (8), we proceed to show that $\sum_{j=1}^{k} \mu_j \ge W_k$ for an arbitrary $1 \le k \le m-1$. The statement is clear for $k = 1$, in view of (71). As for higher values of k, we separate the discussion into two cases:

$\diamond$ Case 1. In round k in the loop 5 we executed Step 5a. This implies that we executed Step 5a for all the preceding values of k as well. Consequently, $\mu_j = c \cdot s_j$ for all $2 \le j \le k$. Hence,

$$\sum_{j=1}^{k} \mu_j = \mu_1 + c \cdot (S_k - s_1) \ge W_k - c \cdot (S_k - s_1) + c \cdot (S_k - s_1) = W_k \ .$$

$\diamond$ Case 1. In round k in the loop 5 we executed Step 5b. Here, it is clear that $\sum_{j=1}^{k} \mu_j = W_m \ge W_k$. $\square$

Lemma 4.6 *The solution $\boldsymbol{\mu}$ that Algorithm 4.4 produces gives a minimum to f, (70), in Ω.*

Proof. Our first observation is that we may concentrate on solutions $\boldsymbol{\mu}' \in \Omega$ where

$$\frac{\mu_j}{s_j} \le c \quad , \quad 2 \le j \le m \ . \tag{72}$$

Indeed, if $\frac{\mu_j}{s_j} > c$ for some $j \ge 2$, then $\mu_j = cs_j + d$ where $d > 0$. In that case, if we decrease μ_j by d and increase μ_1 by d, we get another solution $\boldsymbol{\mu}''$ where $\boldsymbol{\mu}'' \in \Omega$ and $f(\boldsymbol{\mu}'') \le f(\boldsymbol{\mu}') - \frac{d}{s_j} + \frac{d}{s_1} \le f(\boldsymbol{\mu}')$. We note that the solution that Algorithm 4.4 outputs is consistent with (72).

Next, assume that $\boldsymbol{\mu}' = (\mu'_1, \ldots, \mu'_m) \in \Omega$ is a solution that satisfies (72) and $f(\boldsymbol{\mu}') < f(\boldsymbol{\mu})$, i.e., by (70),

$$\sum_{j=1}^{m} \max\left(\frac{\mu'_j}{s_j}, c\right) < \sum_{j=1}^{m} \max\left(\frac{\mu_j}{s_j}, c\right) \ . \tag{73}$$

Since both $\boldsymbol{\mu}'$ and $\boldsymbol{\mu}$ satisfy (72), we conclude by (73) that

$$\max\left(\frac{\mu'_1}{s_1}, c\right) < \max\left(\frac{\mu_1}{s_1}, c\right) \ . \tag{74}$$

This may happen only if $\mu'_1 < \mu_1$ and $\mu_1 > c \cdot s_1$. Hence, by (71),

$$\mu_1 = W_k - c \cdot S_k + c \cdot s_1 \quad \text{for some } 1 \le k \le m \ . \tag{75}$$

Concentrating on that k, we invoke (8) and (72) to conclude that

$$W_k \le \sum_{j=1}^{k} \mu'_j \le \mu'_1 + c \cdot (S_k - s_1) \ .$$

Therefore, $\mu'_1 \ge W_k - c \cdot (S_k - s_1)$. Hence, by (75), $\mu'_1 \ge \mu_1$, in contradiction to our assumption. $\square$

4.4 Separable functions

Here we consider the case where the target function is separable, namely,

$$f(\lambda_1, \ldots, \lambda_m) = \sum_{j=1}^{m} g(\lambda_j) \ , \tag{76}$$

where g is convex and monotonic. In order to solve the corresponding mathematical program MP, we may apply the polynomial time algorithm of Hochbaum and Shanthikumar [7]. That algorithm is designed to solve minimization problems of the form

$$\min_{\mathbf{x} \in D} f(\mathbf{x}) \quad , \quad D = \{\mathbf{x} \in \mathbf{R}^m : A\mathbf{x} \ge \mathbf{b}\} \ , \tag{77}$$

where f is as in (76), A is an integer matrix and D is a bounded polyhedron. The algorithm is polynomial in the size of the input, in the logarithm of the required accuracy and in

$$\Delta = \Delta(A) := \max\{|\det A_M| : A_M \text{ is a square sub-matrix of } A\} \ . \tag{78}$$

It should be noted that when f is the ℓ_p-norm, $p < \infty$, or the threshold cost function, (70), Algorithms 4.1 and 4.4 are simpler and more efficient than the general algorithm in [7].

We need to show that our mathematical program MP falls under the framework for which that algorithm applies. First, we think of the function f in MP as a function of the weights, μ_j, rather than a function of the loads, $\lambda_j = \mu_j/s_j$. Namely,

$$f(\mu_1, \ldots, \mu_m) = \sum_{j=1}^{m} g(\mu_j/s_j) \ . \tag{79}$$

This simple step is necessary so that the m restrictions on $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_m)$, (8)+(9), have integral coefficients and may be written in the form

$$\tilde{A}\boldsymbol{\mu} \geq \tilde{\mathbf{b}} \quad \text{where} \quad \tilde{A}_{i,j} = \begin{cases} 0 & 1 \leq i < j \leq m \\ 1 & 1 \leq j \leq i \leq m \\ -1 & i = m+1, 1 \leq j \leq m \end{cases} \quad \text{and} \quad \tilde{\mathbf{b}} = \begin{pmatrix} W_1 \\ \vdots \\ W_m \\ -W_m \end{pmatrix} . \tag{80}$$

Note that the mth restriction, (9), is an equality and it is represented in (80) in the last two inequalities. Relying on (9), we may restrict the variables μ_j from above as well,

$$\mu_j \leq W_m \quad , \quad 1 \leq j \leq m \ . \tag{81}$$

On the other hand, as we are interested in nonnegative solutions only, we add the set of restrictions

$$\mu_j \geq 0 \quad , \quad 1 \leq j \leq m \ . \tag{82}$$

Putting (80),(81) and (82) together we get a system of requirements of the form (77) where A is a $(3m+1) \times m$ matrix of integer entries,

$$A = \begin{pmatrix} \tilde{A} \\ -I \\ I \end{pmatrix} \tag{83}$$

and

$$\mathbf{b} = \begin{pmatrix} \tilde{\mathbf{b}} \\ \mathbf{w} \\ \mathbf{0} \end{pmatrix} \quad \text{where} \quad \mathbf{w} = \begin{pmatrix} -W_m \\ \vdots \\ -W_m \end{pmatrix} \quad \text{and} \quad \mathbf{0} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} . \tag{84}$$

The corresponding polyhedron D, (77), is bounded. It remains only to evaluate $\Delta(A)$ and to verify that it may not become too large as a function of m.

Claim 4.7 *For A in (83), $\Delta(A) = 1$.*

Proof. Let A_M be any square sub-matrix of A. We shall show that

$$\det A_M \in \{-1, 0, 1\} \ . \tag{85}$$

Assume that A_M corresponds to the selection of rows $1 \le i_1 < \cdots < i_t \le 3m+1$ and columns $1 \le j_1 < \cdots < j_t \le m$. Let us assume first that A_M has entries from the last $2m$ rows of A. Namely, there exists $0 \le s \le t-1$ so that $m+1 < i_{s+1}$. If one of these rows is identically zero in A_M or two of these rows are dependent, then $\det A_M = 0$. Otherwise, the last $t-s$ rows in A_M are of the form $(0,\ldots,0,\pm 1,0,\ldots,0)$ where the non-zero entries, ± 1, appear in different positions. Developing the determinant of A_M according to those rows, we get that $\det A_M = \pm \det A'_M$ where A'_M is a sub-matrix of dimension $s \times s$ that is contained in the first $m+1$ rows of A (namely, in $\tilde{A}$).

Hence, we may concentrate on sub-matrices of A that are contained in $\tilde{A}$. We keep denoting the row and column selections by i_k and j_k where $1 \le k \le t$, and prove our claim by induction on t. Since (85) clearly holds when $t = 1$, we proceed to describe the reduction step. If $j_1 > i_1$, it is easy to see that the first row in A_M is identically zero so that $\det A_M = 0$. If, on the other hand, $j_1 \le i_1$ there are two possibilities. If $j_2 \le i_1$ then the first two columns in A_M are equal whence $\det A_M = 0$. Otherwise, if $j_2 > i_1$, then the first row in A_M is $(1,0,\ldots,0)$. Hence, $\det A_M = \det A'_M$ where A'_M is the sub-matrix of dimension $(t-1)\times(t-1)$ that is obtained from A_M by removing its first row and first column. That sub-matrix, by the induction hypothesis, satisfies (85). $\square$

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 ℓ_2-norm in Ω coincides with the global minimum in $\mathbf{R}^m$, the machine loads indeed relate to each other like the machine speeds, see (50)). 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 $(\Lambda^1_1, \Lambda^1_2, \Lambda^1_3, \Lambda^1_4) = (10, 18, 24, 27)$.

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, Γ^1 and Θ^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 $U_{[5,15)}$. Figures 4-6 depict the three state functions after that application. We see that $\Theta^2(x)$ has $m^2 = 3$ jump discontinuities at $(\Lambda^2_1, \Lambda^2_2, \Lambda^2_3) = (8, 14, 17)$.

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 Γ^2 and Θ^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 $[5, 8)$. The resulting state functions after applying $U_{[1,9)}$ are illustrated in Figures 7-9. Now, $\Theta^3(x)$ has $m^3 = 2$ jump discontinuities at $(\Lambda^3_1, \Lambda^3_2) = (6, 9)$.

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 Θ^3; in that case $m^4 = m^3 = 2$. If, on the other hand, $w_3 \ge 0.9$, $m^4 = 1$ and then J_4 will be the first job in *Phase 2*.

Acknowledgement. We would like thank Asaf Levin for referring us to [7].

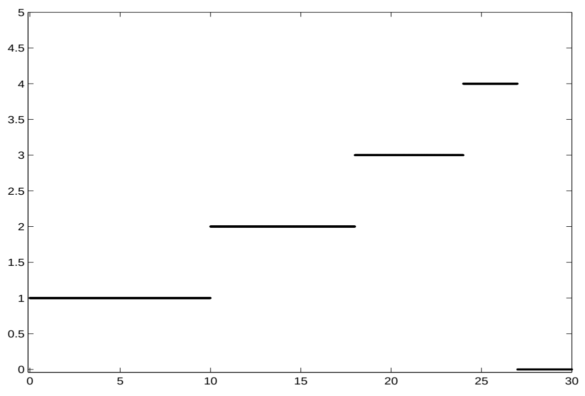

Figure 1: $\Gamma^1(x)$

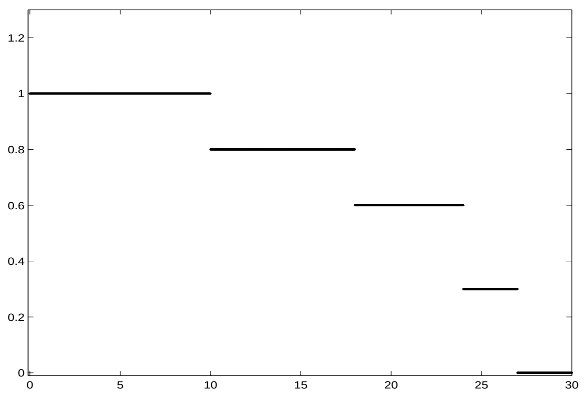

Figure 2: $\Psi^1(x)$

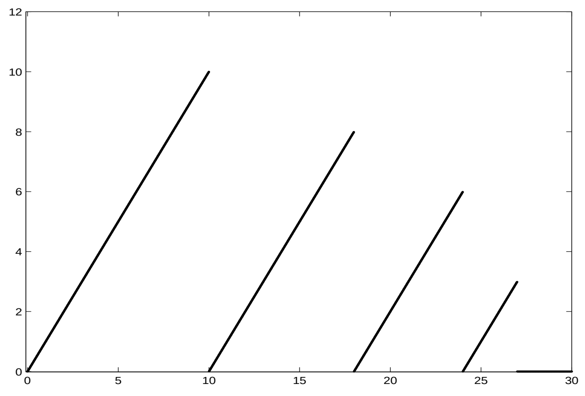

Figure 3: $\Theta^1(x)$

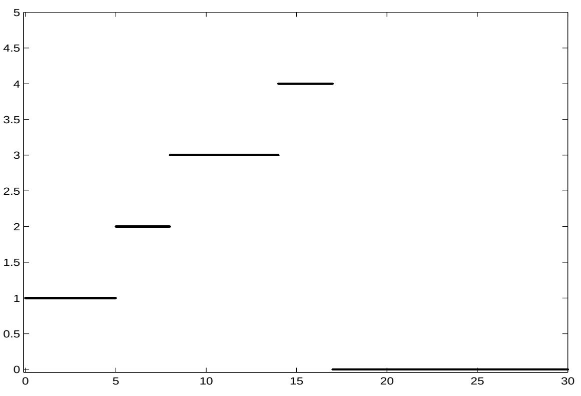
Figure 4: $\Gamma^2(x)$

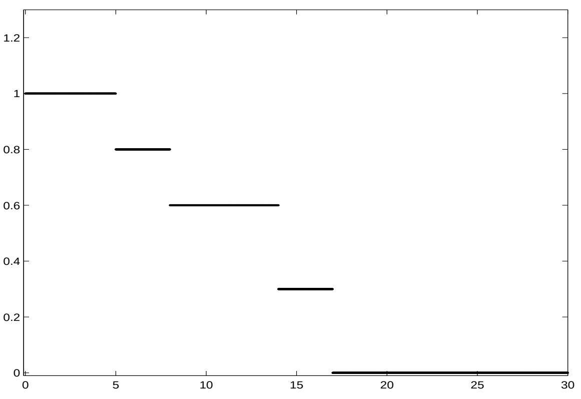
Figure 5: $\Psi^2(x)$

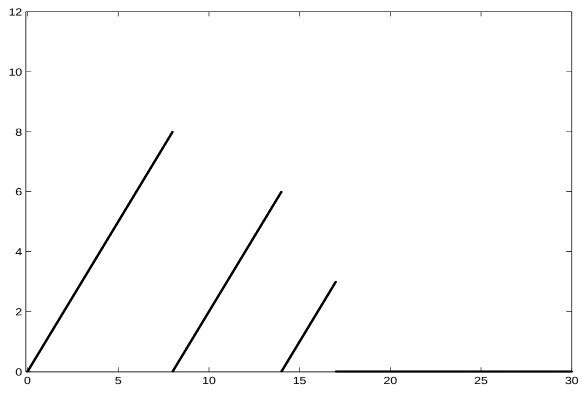
Figure 6: $\Theta^2(x)$

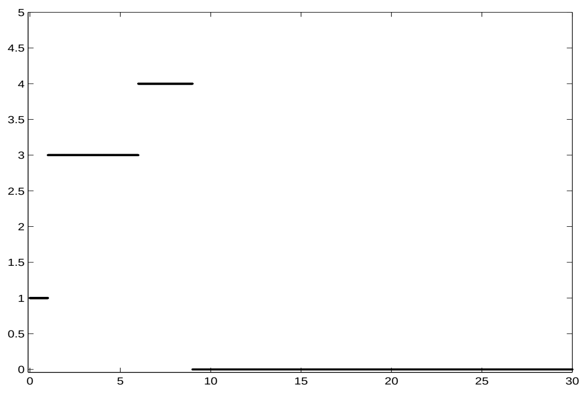

Figure 7: $\Gamma^3(x)$

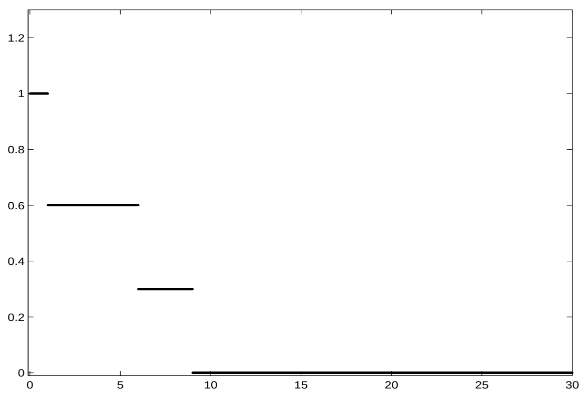

Figure 8: $\Psi^3(x)$

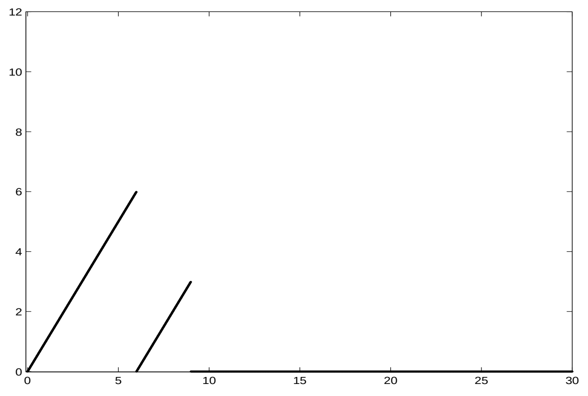

Figure 9: $\Theta^3(x)$

References

[1] N. Alon, Y. Azar, G. Woeginger, T. Yadid, Approximation schemes for scheduling on parallel machines, Journal of Scheduling 1 (1998) 55–66.

[2] E. G. Coffman, Jr., George S. Lueker, Approximation algorithms for extensible bin packing, in: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01), 2001, pp. 586–588.

[3] P. Dell'Olmo, H. Kellerer, M. G. Speranza, Zs. Tuza, A 13/12 approximation algorithm for bin packing with extendable bins, Information Processing Letters 65 (1998) 229–233.

[4] P. Dell'Olmo, M. G. Speranza, Approximation algorithms for partitioning small items in unequal bins to minimize the total size, Discrete Applied Mathematics 94 (1999) 181–191.

[5] L. Epstein, J. Sgall, Approximation schemes for scheduling on uniformly related and identical parallel machines, in 7th Annual European Symposium on Algorithms (ESA'99), 1999, pp. 151–162.

[6] T. Gonzalez, S. Sahni, Preemptive scheduling of uniform processor systems, Journal of the ACM 25 (1978) 92–101.

[7] D. S. Hochbaum, J. G. Shanthikumar, Convex separable optimization is not much harder than linear optimization, Journal of the ACM 37 (1990) 843–862.

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

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

[10] E. C. Horvath, S. Lam, R. Sethi, A level algorithm for preemptive scheduling. Journal of the ACM 24 (1977) 32–43.

[11] J. W. S. Liu, A. T. Yang, Optimal scheduling of independent tasks on heterogeneous computing systems, in Proceedings ACM National Conference, volume 1, ACM, 1974, pp. 38–45.

[12] H. Shachnai, T. Tamir, G. J. Woeginger, Minimizing makespan and preemption costs on a system of uniform machines, in Proceedings of the 10th Annual European Symposium on Algorithms (ESA'02), 2002, pp. 859–871.