Approximation Schemes for Scheduling on Uniformly Related and Identical Parallel Machines

Leah Epstein* Jiří Sgall†

Abstract

We give a polynomial approximation scheme for the problem of scheduling on uniformly related parallel machines for a large class of objective functions that depend only on the machine completion times, including minimizing the l_p norm of the vector of completion times. This generalizes and simplifies many previous results in this area.

1 Introduction

We are given n jobs with processing times p_j, $j \in J = \{1, \ldots, n\}$ and m machines M_1, M_2, $\ldots$, M_m, with speeds s_1, s_2, $\ldots$, s_m. A schedule is an assignment of the n jobs to the m machines. Given a schedule, for $1 \leq i \leq m$, T_i denotes the weight of machine M_i which is the total processing time of all jobs assigned to it, and C_i denotes the completion time of M_i, which is T_i/s_i.

Our objective is, for some fixed function $f : [0, +\infty) \to [0, +\infty)$, one of the following

*`lea@math.tau.ac.il`, Department of Computer Science, Tel-Aviv University, Israel.

†`sgall@math.cas.cz`, Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic and Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Praha. Partially supported by grant A1019602 of GA AV ČR, postdoctoral grant 201/97/P038 of GA ČR, and cooperative research grant INT-9600919/ME-103 from the NSF and MŠMT CR.

(I) minimize $\sum_{i=1}^{m} f(C_i)$,

(II) minimize $\max_{i=1}^{m} f(C_i)$,

(III) maximize $\sum_{i=1}^{m} f(C_i)$, or

(IV) maximize $\min_{i=1}^{m} f(C_i)$.

Most of such problems are NP-hard, see [9, 14]. Thus we are interested in approximation algorithms. Recall that a *polynomial time approximation scheme (PTAS)* is a family of polynomial time algorithms over $\varepsilon > 0$ such that for every ε and every instance of the problem, the corresponding algorithm outputs a solution whose value is within a factor of $(1 + \varepsilon)$ of the optimum value [9].

We give a PTAS for scheduling on uniformly related machines for a rather general set of functions f, covering many natural functions studied before. Let us give some examples covered by our results and references to previous work.

Example 1. Problem (I) with $f(x) = x$ is the basic problem of minimizing the maximal completion time (makespan). It was studied for identical machines in [10, 11, 15, 12]; the last paper gives a PTAS. Finally, Hochbaum and Shmoys [13] gave a PTAS for uniformly related parallel machines. Thus our result can be seen as a generalization of that result. In fact our paper is based on their techniques, perhaps somewhat simplified.

Example 2. Problem (I) with $f(x) = x^p$ for any $p > 1$. This is equivalent to minimizing the l_p norm of the vector $(C_1, \ldots, C_m)$, i.e., $(\sum C_i^p)^{1/p}$. For $p = 2$ and identical machines this problem was studied in [6, 5], motivated by storage allocation problems. For a general p a PTAS for identical machines was given in [1]. For related machines this problem was not studied before.

Note that for $p \leq 1$ the minimization problem is trivial: it is optimal to schedule all jobs on the fastest machine (choose one arbitrarily if there are more of them). In this case a more interesting variant is the maximization version, i.e., the problem (III).

Example 3. Problem (IV) with $f(x) = x$. In this problem the goal is to maximize the time when all the machines are running; this corresponds to keeping all parts of some system alive as long as possible. This problem on identical machines was studied in [8, 7], and [16] gave a PTAS. For uniformly related machines a PTAS was given in [3].

Example 4. Scheduling with rejection. In this problem each job has associated certain penalty. The schedule is allowed to schedule only some subset of jobs, and the goal is to minimize the maximal completion time plus the total penalty of all rejected jobs. This problem does not conform exactly to any of the categories above, nevertheless our scheme can be extended to work for it as well. This problem was studied in [4] where also a PTAS for the case of identical machines is given. For related machines this problem was not studied.

Our paper is directly motivated by Alon et al. [2], who proved similar general results for scheduling on identical machines. We generalize it to uniformly related machines and a similar set of functions f. Even for identical machines, our result is stronger than that of [2] since in that case we allow a more general class of functions f.

The basic idea is to round the size of all jobs to a constant number of different sizes, to solve the rounded instance exactly, and then reconstruct an almost optimal schedule for the original instance. This rounding technique traces back to Hochbaum and Shmoys [12]. We also use an important improvement from [1, 2]: the small jobs are clustered into blocks of jobs of small but non-negligible size. The final ingredient is that of [13, 3]: the rounding factor is different for each machine, increasing with the weight of the machine (i.e., the total processing time of jobs assigned to it). Such rounding is possible if we assign the jobs to the machines in the order of non-decreasing weight. This is easy to do for identical machines. For uniformly related machines we prove, under a reasonable condition on the function f, that in a good solution the weight of machines increases with their speed, which fixes the order of machines.

2 Our assumptions and results

Now we define our assumptions on the function f. Let us note at the beginning that the typical functions used in scheduling problems satisfy all of them.

The first condition says that to approximate the contribution of a machine up to a small multiplicative error, it is sufficient to approximate the completion time of the machine up to a small multiplicative error. This condition is from Alon et al. [2], and it is essential for all our results.

$$\text{(F}*\text{)}: \qquad (\forall \varepsilon > 0)(\exists \delta > 0)(\forall x, y \geq 0)$$
$$(|y - x| \leq \delta x \rightarrow |f(y) - f(x)| \leq \varepsilon f(x)).$$

If we transform f so that both axes are given a logarithmic scale, the condition naturally translates into uniform continuity, as $|y - x| \leq \delta x$ iff $(1-\delta)x \leq y \leq (1+\delta)x$ iff $\ln(1-\delta) + \ln x \leq \ln y \leq \ln(1+\delta) + \ln x$, and similarly for $f(x)$. More formally, (F$*$) is equivalent to the following statement:

(F$**$) : The function $h_f : (-\infty, +\infty) \rightarrow (-\infty, +\infty)$
defined by $h_f(z) = \ln f(e^z)$
is defined everywhere and uniformly continuous.

We also need a condition that would guarantee that in the optimal schedule, the weights of the non-empty machines are monotone. These conditions are different for the cases when the objective is maximize or minimize $\sum f(C_i)$, and for the cases of min-max or max-min objectives.

Recall that a function g is convex iff for every $x \leq y$ and $0 \leq \Delta \leq y - x$, $f(x + \Delta) + f(x - \Delta) \leq f(x) + f(y)$. For the cases of min-sum and max-sum we require this condition:

(G$*$) : The function $g_f : (-\infty, +\infty) \rightarrow [0, +\infty)$
defined by $g_f(z) = f(e^z)$ is convex.

Note that $f(x) = g_f(\ln x)$. Thus, the condition says that the function f is convex, if plotted in a graph with a logarithmic scale on the x-axis and a linear scale on the y-axis. This is true for example for any non-decreasing convex function f. However, g_f is convex, e.g., even for $f(x) = \ln(1 + x)$. On the other hand, the condition (G$*$) implies that f is either non-decreasing or it is unbounded for x approaching 0.

For the case of min-max or max-min we require that the function f is bimodal on $(0, +\infty)$, i.e., there exists an x_0 such that f

is monotone (non-decreasing or non-increasing) both on $(0, x_0]$ and $[x_0, +\infty)$. (E.g., $(x-1)^2+1$ is bimodal, decreasing on $(0,1]$ and increasing on $[1,+\infty)$.) This includes all convex functions as well as all non-decreasing functions.

Note that none of the conditions above puts any constraints on $f(0)$.

Last, we need the function f to be computable in the following sense: for any $\varepsilon > 0$ there exists an algorithm that on any rational x outputs a value between $(1-\varepsilon)f(x)$ and $(1+\varepsilon)f(x)$, in time polynomial in the size of x. To simplify the presentation, we will assume in the proofs that f is computable exactly; choosing smaller ε and computing the approximations instead of the exact values always works. Typical functions f that we want to use are computable exactly, but for example if $f(x) = x^p$ for non-integral p then we can only approximate it.

Our main results are:

Theorem 2.1 *Let f be a non-negative computable function satisfying the conditions* (F$*$) *and* (G$*$). *Then the scheduling problems of minimizing and maximizing $\sum f(C_i)$ on uniformly related machines both possess a polynomial approximation scheme. The running time of these PTAS is $O(n^c p(|I|))$, where c is a constant depending on the desired precision, p is the polynomial bounding the time of the computation of f, and $|I|$ is the size of the input instance.*

Theorem 2.2 *Let f be a non-negative computable bimodal function satisfying the condition* (F$*$). *Then the scheduling problems of minimizing $\max f(C_i)$ and of maximizing $\min f(C_i)$ on uniformly related machines both possess a polynomial approximation scheme. The running time of these PTAS is $O(n^c p(|I|))$, where c is a constant depending on the desired precision, p is the polynomial bounding the time of the computation of f, and $|I|$ is the size of the input instance.*

Theorem 2.3 *Let f be a non-negative computable function satisfying the condition* (F$*$). *Then the scheduling problems of minimizing or maximizing $\sum f(C_i)$, of minimizing $\max f(C_i)$, and of maximizing $\min f(C_i)$ on identical machines all possess a polynomial approximation scheme. The running time of these PTAS is $O(n^c p(|I|))$, where c is a constant depending on the desired precision, p is the polynomial* *bounding the time of the computation of f, and $|I|$ is the size of the input instance.*

Our PTAS are running in polynomial time, but the exponent in the polynomial depends on f and ε. This should be contrasted with Alon et al [2], where for the case of identical machines they are able to achieve linear time (i.e., the exponential dependence on ε is only hidden in the constant) using integer programming in fixed dimension. It is an open problem if such an improvement is also possible for related machines.

3 Ordering of the machines

The following lemma shows why the conditions (G$*$) and bimodality are important for the respective problems; it is the only place where the conditions are used. Note that for identical machines the corresponding statements hold trivially without any condition.

Lemma 3.1 *Suppose the machines are ordered so that their speeds s_i are non-decreasing.*

Under the same assumptions as in Theorem 2.1, there exists a schedule with minimal (maximal, resp.) $\sum f(C_i)$ in which the non-zero weights of the machines are monotone non-decreasing (monotone non-increasing, resp.). I.e., for any $1 \le i < j \le m$ such that $T_i, T_j > 0$, we have $T_i \le T_j$ ($T_i \ge T_j$, resp.).

Under the same assumptions as in Theorem 2.2, there exist schedules both with minimal $\max f(C_i)$ and with maximal $\min f(C_i)$ in which the non-zero weights of the machines are either monotone non-decreasing or monotone non-increasing.

Proof: Next we prove the lemma for minimization of $\sum f(C_i)$; the case of maximization is similar. We prove that if $T_i > T_j$ then switching the assigned jobs between M_i and M_j leads to at least as good schedule. This is sufficient, since given any optimal schedule we can obtain an optimal schedule with ordered weights by at most $m-1$ such transpositions.

Denote $s = s_j/s_i$, $\Delta = \ln s$, $X = \ln C_j = \ln(T_j/s_j)$, and $Y = \ln C_i = \ln(T_i/s_i)$. From the assumptions we have $X < Y$ and $0 \le$

$\Delta < X - Y$. The difference of the original cost and the cost after the transposition is $f(C_i) + f(C_j) - f(C_i \cdot s) - f(C_j/s) = g_f(X) + g_f(Y) - g_f(X+\Delta) - g_f(Y-\Delta) \geq 0$, using convexity of g_f. Thus the transposed schedule is at least as good as the original one.

Now suppose that we are maximizing $\min f(C_i)$ and f is bimodal, first non-increasing then non-decreasing. We prove that if $T_i > T_j$ for $i < j$, then switching the assigned jobs between M_i and M_j can only improve the schedule. We have $C_j/s \leq C_i, C_j \leq C_i \cdot s$, for $s = s_j/s_i$. By bimodality we have $\min\{C_i, C_j\} \leq \min\{C_j/s, C_i \cdot s\}$, hence the transposed schedule can only be better. The cases of minimization of $\max f(C_i)$ and f first non-decreasing then non-increasing are similar, with the order reversed as needed. ∎

The lemma above implies that, depending on the type of the problem and f, we can order the machines in either non-decreasing or non-increasing order of speeds and then consider only the schedules in which the weights of the machines are non-decreasing (possibly with the exception of the empty machines).

4 Preliminaries and definitions

Let $\delta > 0$ and λ be such that $\lambda = 1/\delta$ is an even integer; we will choose it later. The meaning of δ is the (relative) rounding precision.

Given w, either 0 or an integral power of two, intuitively the order of magnitude, we will represent a set of jobs with processing times not larger than w as follows. For each job of size more than δw we round its processing time to the next higher multiple of $\delta^2 w$; for the remaining small jobs we add their processing times, round up to the next higher multiple of δw, and treat these jobs as some number of jobs of processing time δw. Now it is sufficient to remember the number n_i of modified jobs of processing time $i\delta^2 w$, for each i, $\lambda \leq i \leq \lambda^2$. Such a vector together with w is called a configuration.

In our approximation scheme, we will proceed machine by machine, and use this representation for two types of sets of jobs. First one is the set of all jobs scheduled so far; we represent them always with the least possible w (principal configurations below). The second type are the sets of jobs assigned to individual machines; we represent them with w small compared to the total processing time of their jobs (heavy configurations below), and this is sufficient to guarantee that the value of f can be approximated well.

Definition 4.1 *Let $A \subseteq J$ be a set of jobs.*

- *The* weight of a set of jobs *is* $W(A) = \sum_{j \in A} p_j$.
- *A* configuration *is a pair* $\alpha = (w, (n_\lambda, n_{\lambda+1}, \ldots, n_{\lambda^2}))$, *where* $w = 0$ *or* $w = 2^i$ *for some integer* i *(possibly negative) and* $\vec{n}$ *is a vector of nonnegative integers.*
- *A configuration* $(w, \vec{n})$ represents A *if*

  (i) no job $j \in A$ *has processing time* $p_j > w$,

  (ii) for $\lambda < i \leq \lambda^2$, n_i *equals the number of jobs* $j \in A$ *with* $p_j \in ((i-1)\delta^2 w, i\delta^2 w]$, *and*

  (iii) $n_0 = \lceil W(A')/(\delta w) \rceil$ *where* $A' = \{j \in A \mid p_j \leq \delta w\}$.
- *The* principal configuration of A *is the configuration* $\alpha(A) = (w, \vec{n})$ *with the smallest* w *that represents* A.
- *The* weight of a configuration $(w, \vec{n})$ *is defined by* $W(w, \vec{n}) = \sum_{i=\lambda}^{\lambda^2} n_i \delta^2 w$.
- *A configuration* $(w, \vec{n})$ *is called* heavy *if its weight* $W(w, \vec{n})$ *is at least* $w/2$.
- *The* successor *of a configuration* $(w, \vec{n})$ *is* $\mathrm{succ}(w, \vec{n}) = (w, \vec{n} + (1, 0, \ldots, 0))$.

Note that given an $A \subseteq J$ and w, the definition gives a linear-time procedure that either finds the unique configuration $(w, \vec{n})$ representing it or decides that no such configuration exists.

Lemma 4.2

(i) Let $A \subseteq J$ be a set of jobs and let $(w, \vec{n})$ be any configuration representing it. Then

$$(1-\delta)W(w, \vec{n}) - \delta w \leq W(A) \leq W(w, \vec{n}).$$

(ii) Any $A \subseteq J$ has a unique principal configuration, and it can be constructed in linear time. The number of principal configuration is bounded by $(n+1)^{\lambda^2}$ and they can be enumerated efficiently.

(iii) Let $A, A' \subseteq J$ be both represented by $(w, \vec{n})$. Then for any $w' > w$ they are both represented by $(w', \vec{n}')$ for some $\vec{n}'$.

Proof: (i) Any job $j \in A$ with $p_j > \delta w$ contributes to $W(w, \vec{n})$ some r, $p_j \leq r < p_j + \delta^2 w$. Its contribution to $W(A)$ is $p_j > r - \delta^2 w \geq r - \delta p_j \geq (1-\delta) r$. The small jobs $j \in A$, $p_j \leq \delta w$, can cause a total additive error of at most δw. Summing over all jobs, the bound follows.

(ii) The principal configuration of $A = \emptyset$ has $w = 0$. For a nonempty A, find a job $j \in A$ with the largest processing time p_j, and round up p_j to a power of two to obtain w. It follows that there are at most $n+1$ possible values of w in the principal configurations. To enumerate all principal configurations with a given w, find a representation $(w, \vec{n})$ of $J(w) = \{j \in J \mid p_j \leq w\}$ and enumerate all the vectors $\vec{n}'$ bounded by $\vec{0} \leq \vec{n}' \leq \vec{n}$ (coordinatewise), such that $n'_i > 0$ for some $i > \lambda^2/2$. (Here we use the fact that λ is even.)

(iii) If $w = 0$ then $A = A' = \emptyset$ and the statement is trivial. Otherwise define

$$n_i^+ = \begin{cases} \left\lceil \frac{W(0,(n_\lambda, n_{\lambda+1}, \ldots, n_{2\lambda}, 0, \ldots, 0))}{2\delta} \right\rceil & \text{for } i = \lambda, \\ n_{2i-1} + n_{2i} & \text{for } \lambda < i \leq \lambda^2/2, \\ 0 & \text{for } i > \lambda^2/2. \end{cases}$$

It is easy to verify that if A is represented by $(w, \vec{n})$ then it is represented by $(2w, \vec{n}^+)$. Iterating this operation sufficiently many times proves that the representation with any $w' > w$ is the same for both A and A'. ■

Next we define a difference of principal configurations and show how it relates to a difference of sets. This is essential for our scheme. (It is easy to define difference of any configurations, but we do not need it.)

Definition 4.3 *Let $(w, \vec{n})$ and $(w', \vec{n}')$ be two principal configurations. Their difference is defined as follows. First, let $(w', \vec{n}'')$ be the configuration that represents the same sets of jobs as $(w, \vec{n})$ (using*

Lemma 4.2 (iii)). Now define $(w', \vec{n}') - (w, \vec{n}) = (w', \vec{n}' - \vec{n}'')$. *If* $w' < w$, *or the resulting vector has some negative coordinate(s), the difference is undefined.*

Lemma 4.4 *Let* $A \subseteq J$ *be a set of jobs and* $(w, \vec{n})$ *its principal configuration.*

(i) Let $(w', \vec{n}')$ *be a principal configuration such that* $(w', \vec{n}') - (w, \vec{n})$ *is defined. Then there exists a set of jobs* B *represented by* $(w', \vec{n}')$ *such that* $A \subseteq B \subseteq J$, *and it can be constructed in linear time.*

(ii) Let B *be any set of jobs such that* $A \subseteq B \subseteq J$, *and let* $(w', \vec{n}')$ *be its principal configuration. Then* $\gamma = (w', \vec{n}') - (w, \vec{n})$ *is defined and* $B - A$ *is represented by* γ *or* $\mathrm{succ}(\gamma)$. *Furthermore, if* $\gamma = (w', \vec{n}') - (w, \vec{n})$ *is heavy then the weight of* $B - A$ *is bounded by* $|W(B - A) - W(\gamma)| \leq 3\delta W(\gamma)$.

Proof: Let $(w', \vec{n}'')$ be the configuration representing A; it can be computed from A and w'.

(i) Since the difference is defined, $\vec{n}'' \leq \vec{n}'$. For each i, $\lambda < i \leq \lambda^2$, add $n'_i - n''_i$ jobs with processing time $p_j \in ((i-1)\delta^2 w, i\delta w]$; since $(w', \vec{n}')$ is a principal configuration we are guaranteed that a sufficient number of such jobs exists. Finally, add jobs with $p_j \leq \delta w$ one by one until n''_λ increases to n'_λ. Each added job increases the coordinate by at most 1, and we have sufficiently many of them since $(w', \vec{n}')$ is principal.

(ii) It is easy to see that $A \subseteq B$ implies that $w \leq w'$ and $\vec{n}'' \leq \vec{n}'$, thus the difference is defined. For i, $\lambda < i \leq \lambda^2$, $n'_i - n''_i$ is the number of jobs in $B - A$ with the appropriate processing times. Let W_A and W_B be the weight of jobs in A and B with $p_j \leq \delta w'$. We have $n''_\lambda - 1 < W_A/(\delta w') \leq n''_\lambda$ and $n'_\lambda - 1 < W_B/(\delta w') \leq n'_\lambda$, thus $n'_\lambda - n''_\lambda - 1 < (W_B - W_A)/(\delta w') < n'_\lambda - n''_\lambda + 1$. This is rounded to $n'_\lambda - n''_\lambda$ or $n'_\lambda - n''_\lambda + 1$, hence $B - A$ is represented by γ or γ'. By Lemma 4.2 (i) it follows that

$$(1 - \delta)W(\gamma) - \delta w' \leq W(B - A) \leq W(\gamma') = W(\gamma) + \delta w'.$$

If γ is heavy then $\delta w' \leq 2\delta W(\gamma)$, and the lemma follows. ∎

From the part (i) of the lemma it also follows that given a principal configuration, it is easy to find a set it represents: just set $A = \emptyset$.

Thus also the difference can be computed in linear time, using the procedure in the definition.

5 The approximation scheme

Given an $\varepsilon \in (0,1]$, we choose δ using (F$*$) so that $\lambda = 1/\delta$ is an even integer and

$$(\forall x, y \geq 0)(|y - x| \leq 3\delta x \rightarrow |f(y) - f(x)| \leq \frac{\varepsilon}{3} f(x)).$$

Definition 5.1 *We define the graph G of configurations as follows.*

The vertices of G are $(i, \alpha(A))$, for any $1 \leq i < m$ and any $A \subseteq J$, the source vertex $(0, \alpha(\emptyset))$, and the target vertex $(m, \alpha(J))$.

For any i, $1 \leq i \leq m$, and any configurations α and β, there is an edge from $(i-1, \alpha)$ to (i, β) iff either $\beta = \alpha$, or $\beta - \alpha$ is defined and $\mathrm{succ}(\beta - \alpha)$ *is heavy. The cost of this edge is defined as $f(W(\beta - \alpha)/s_i)$. There are no other edges.*

Definition 5.2 *Let $J_1, \ldots, J_m$ be an assignment of jobs J to machines $M_1, \ldots, M_m$. Its* representation *is a sequence of vertices of G $\{(i, \alpha_i)\}_{i=0}^m$, where $\alpha_i = \alpha(\bigcup_{i'=1}^i J_{i'})$.*

Note that we really obtain vertices of G, as $\alpha_0 = \alpha(\emptyset)$ and $\alpha_m = \alpha(J)$.

The approximation scheme performs the following steps:

(1) Order the machines with speeds either non-decreasing or non-increasing, according to the type of the problem and f so that by Lemma 3.1 there exists an optimal schedule with non-decreasing non-zero weights of the machines.

(2) Construct the graph G.

(3) Find an optimal path in G from source $(0, \alpha(\emptyset))$ to $(m, \alpha(J))$. The cost of the path is defined as the sum, maximum or minimum of the costs of the edges used, and an optimal path is one with the cost minimized or maximized, as specified by the problem.

(4) Output an assignment represented by the optimal path constructed as follows: Whenever the path contains

an edge of the form $((i-1,\alpha),(i,\alpha))$, put $J_i = \emptyset$. For every other edge, apply Lemma 4.4 (i), starting from the beginning of the path.

Lemma 4.2 (ii) shows that we can construct the vertices of G in time $O(n^{1/\lambda^2})$. Computing the edges of G and their costs is also efficient. Since the graph G is layered, finding an optimal path takes linear time in the size of G. Given a path in a graph, finding a corresponding assignment is also fast. Hence the complexity of our PTAS is as claimed.

Lemma 5.3

(i) If $\{J_i\}$ is an assignment with non-decreasing weights of the machines with non-zero weights (see Lemma 3.1), then its representation $\{(i,\alpha_i)\}_{i=0}^m$ is a path in G.

(ii) Let $\{J_i\}$ be an assignment whose representation $\{(i,\alpha_i)\}_{i=0}^m$ is a path in G and such that if $\alpha_{i-1} = \alpha_i$ then $J_i = \emptyset$. Let C be the cost of the schedule given by the assignment, and let $C^\#$ be the cost of the representation as a path in the graph. Then $|C - C^\#| \leq \varepsilon C^\#/3$.

Proof: (i) For any $i = 1,\ldots,m$, if $\alpha_{i-1} = \alpha_i$ then $((i-1,\alpha_{i-1}),(i,\alpha_i))$ is an edge by definition. Otherwise by Lemma 4.4 (ii), the difference $\gamma = \alpha_i - \alpha_{i-1}$ is defined and J_i is represented by γ or $\mathrm{succ}(\gamma)$, and thus $W(J_i) \leq W(\mathrm{succ}(\gamma))$. We need to show $\mathrm{succ}(\gamma)$ is heavy. Let w be the order of α_i, and thus also of γ. If $w = 0$, the statement is trivial. Otherwise some job with $p_j > w/2$ was scheduled on one of the machines $M_1, \ldots, M_i$. Since the assignment has non-decreasing weights, it follows that $w/2 < W(J_i) \leq W(\mathrm{succ}(\gamma))$, and γ is heavy.

(ii) Let $X_i = W(\alpha_i - \alpha_{i-1})$ and let $Y_i = f(X_i/s_i)$ be the cost of the ith edge of the path. If $\alpha_{i-1} = \alpha_i$, then $J_i = \emptyset$ and $f(C_i) = Y_i = f(0)$. Otherwise by Lemma 4.4 (ii), $|W(J_i) - X_i| \leq 3\delta X_i$. Thus $|C_i - X_i/s_i| = |W(J_i)/s_i - X_i/s_i| \leq 3\delta X_i/s_i$ and by the condition (F$*$) and our choice of δ we get $|f(C_i) - Y_i| \leq \varepsilon Y_i/3$. Summing over all edges of the path we get the required bound. ∎

We now finish the proof of Theorems 2.1 and 2.2 for the minimization versions; the case of maximization is similar. Let C^* be the optimal cost, let $C^\#$ be the cost of an optimal path in G, and let C be the cost of the output solution of the PTAS. By Lemma 3.1

there exists an optimal schedule with non-decreasing weights. Thus by Lemma 5.3 (i) it is represented by a path, which cannot be cheaper than the optimal path, and by Lemma 5.3 (ii) $C^* \leq (1-\frac{\varepsilon}{3})C^{\#}$. Using Lemma 5.3 (ii) for the output assignment we get

$$C \leq (1+\frac{\varepsilon}{3})C^{\#} \leq \frac{1+\frac{\varepsilon}{3}}{1-\frac{\varepsilon}{3}}C^* \leq (1+\varepsilon)C^*.$$

Thus we have found a required approximate solution. (Note that the output solution may not have non-decreasing weights, due to rounding.)

6 Discussion

To get more insight in the meaning of the condition (F$*$), we prove the following characterization for convex functions.

Observation 6.1 *Suppose $f : [0,+\infty) \to [0,+\infty)$ is on $(0,+\infty)$ convex and not identically 0. Then f satisfies* (F$*$) *if and only if the following conditions hold:*

- *$f(x) > 0$ for any $x > 0$,*
- *for $x \to \infty$, $f(x)$ is polynomially bounded both from above and below (i.e., for some constant c, $f(x) \leq O(x^c)$ and $f(x) \geq \Omega(1/x^c)$).*
- *for $x \to 0$, $f(x)$ is polynomially bounded both from above and below (i.e., for some constant c, $f(x) \leq O(1/x^c)$ and $f(x) \geq \Omega(x^c)$).*

Proof: If $f(x) = 0$ for $x > 0$ then (F$*$) holds if and only if f is identically 0 on $(0,+\infty)$.

Otherwise let $h_f(z) = \ln f(e^z)$ be as in (F$**$), note that it is defined everywhere. Since f is convex, it has both left and right derivatives everywhere (and they are equal almost everywhere). Thus also h_f has both derivatives everywhere, and (F$**$) (i.e., uniform continuity of h_f) is equivalent to the statement that the derivatives of h_f are bounded by some constants, both from above and below.

This is equivalent to the fact that h_f is bounded by linear functions, from above and below, and for z approaching both $-\infty$ and $+\infty$. This in turn is equivalent with f being polynomially bounded, both from above and below, and for x approaching both 0 and $+\infty$. ■

This characterization is related to Conjecture 4.1 of Alon et al. [2] which we now disprove. The conjecture says that for a convex function f, and for the problem of minimizing $\sum f(C_i)$ on identical machines the following three conditions are equivalent: (i) it has a PTAS, (ii) it has a polynomial approximation algorithm with a finite performance guarantee, (iii) the heuristic LPT, which orders the jobs according to non-increasing processing times and schedules them greedily on the least loaded machine, has a finite performance guarantee.

We know that if (F$*$) holds, there is a PTAS (for a computable f) [2]. Observation 6.1 implies that if (F$*$) does not hold, then LPT does not have a finite performance guarantee; the proof is similar to Observation 4.1 of [2] (which says that no such algorithm exists for an exponentially growing function, unless $P = NP$, by a reduction to KNAPSACK).

Now consider $f(x) = x^{t(x)}$ where t is some slowly growing unbounded function; $t(x) = \log\log\log\log x$ will work. It is easy to verify that any such f is convex and does not satisfy (F$*$). However, it is possible to find a PTAS on identical machines using the integer programming approach of [2]. The function f does not satisfy (F$*$) on $[0, +\infty)$, but it satisfies (F$*$) for any interval $[0, T]$, moreover for a fixed ε the value of δ can be bounded by $\varepsilon/O(t(T))$. The PTAS algorithm now proceeds in the following way. It computes the bound on the completion time T as the sum of all processing times and chooses δ and $\lambda = 1/\delta$ accordingly. Since M is at most singly exponential in the size of the instance, λ is proportional to a triple logarithm of the instance size. Now we use the integer programming approach from [2]. Resulting algorithm has time complexity doubly exponential in λ, which is bounded by the size of the instance. Thus the algorithm is polynomial.

Let us conclude by a few remarks about the problem of minimizing $\max f(C_i)$. It is easy to approximate it for any increasing f satisfying (F$*$): just approximate the minimum makespan, and then apply f to that. Thus our extension to bimodal functions is not very strong. However, our techniques apply to a wider range of functions f. Suppose for example that the function f is increasing between 0 and 1, and then again from between c and $+\infty$, with an arbitrary behavior between 1 and c. Then it is possible to prove a weaker version of Lemma 3.1, saying that for some (almost) optimal schedule, for any $i < j$, $T_i < \mu T_j$ (if $T_i, T_j > 0$). The constant μ will depend only on the function f. This is sufficient for the approximation scheme, if we redefine the heavy edges to be the ones with weight $\mu w/2$ rather than $w/2$, and choose the other constants appropriately smaller. We omit the details and precise statement, since this extension of our results does not seem to be particularly interesting.

7 Scheduling with rejection

In this section we study the problem from Example 4 in the introduction.

Theorem 7.1 *Let f be a non-negative computable function satisfying the conditions* (F$*$) *and* (G$*$). *Then the problem of scheduling with rejection on uniformly related machine with the objection to minimize the sum of weights of rejected jobs plus $\sum f(C_i)$ possesses a polynomial approximation scheme.*

Let f be a non-negative computable bimodal function satisfying the condition (F$*$). *Then the problem of scheduling with rejection on uniformly related machines with the objection to minimize the sum of weights of rejected jobs plus* $\max f(C_i)$ *possesses a polynomial approximation scheme.*

If the machines are identical, then the same is true (in both cases above) even if f is computable and satisfies only the condition (F$*$).

The running time of all these PTAS is $O(n^c p(|I|))$, where c is a constant depending on the desired precision, p is the polynomial bounding the time of the computation of f, and $|I|$ is the size of the input instance.

The proof is a modification of our general PTAS. We give only a brief sketch. We start with the first case, i.e., the objective is penalty plus $\sum f(C_i)$.

We modify the graph G used in our PTAS in the following way. We add n auxiliary levels between any two levels of the original graph, as well as after the last level. Each level will again have nodes corresponding to all principal configurations, the target node will now be the node $\alpha(J)$ on the last auxiliary level. The edges entering the original nodes and their values will be as before. The edges entering the auxiliary levels will be as follows. There will be an edge from a configuration $(w, \vec{n})$ to $(w', \vec{n}')$ iff the following holds: $w < w'$, $(w', \vec{n}'') = (w', \vec{n}') - (w, \vec{n})$ is defined, and $n''_i = 0$ for all $i \leq \lambda^2/2$. (The last condition says that $(w', \vec{n}'')$ represents only sets of jobs with all processing times greater than $w'/2$.) The value of the edge will be the smallest total penalty of a set of jobs represented by $(w', \vec{n}'')$. Additionally, there will be edges between identical configurations, with weight 0.

The size of the graph increases by a factor of n. It is easy to construct the nodes and edges of the graph. Given an edge entering an auxiliary level, we know the number of jobs of each size that should be rejected, so we just add penalties of the appropriate number of jobs with the smallest penalties. Thus also the values of the edges can be computed efficiently.

A schedule is represented by a sequence of nodes, one on each level, such that on the ith original level we use the configuration $\alpha(A)$ for the following set of jobs A: let B be the set of jobs scheduled on the first i machines, let w be the weight of the principal configuration of B. Then A consist of B together with all jobs j rejected in the schedule such that $p_j \leq w$. On the previous auxiliary level we use the configuration $\alpha(A')$, where A' is A minus all jobs scheduled on M_i. On the last level we use the target node. The other nodes are arbitrary (we are again mainly interested in the representations that are paths in the graph).

Given a path in the graph, we can construct a schedule represented by it as follows. We follow the path from the source. Upon traversing an edge entering an auxiliary level, such that its endpoints have different configurations, we reject a subset of jobs represented by the difference of the configurations, with the smallest total penalty among such sets. As described above, this can be computed efficiently. We are guaranteed that these jobs were not scheduled or rejected so far, as their processing time is greater than the weight of the configuration at the beginning of the edge. The sum of the values of the edges entering the intermediate nodes is exactly equal to the penalty of rejected jobs, and the contribution of the machines is bounded as before. Thus the cost of the path is close to the cost of the schedule.

Given the optimal schedule, we need to show that it is represented by a path. As before, we may assume that the non-zero weights of machines are non-decreasing. Given the schedule, the nodes on the original levels of the graph and the preceding intermediate levels are exactly determined, and connected by edges. Given any $A \subseteq B$, it is easy to verify that the node $\alpha(A)$ of an original level is connected by a path with the node $\alpha(B)$ of the nth intermediate level after it. The total value of the edges of this path is always at most the total penalty of the jobs in $B - A$. (Note that here we may have to use more than one non-trivial edge, as we may be rejecting jobs in different weight ranges; this is the reason why we use n auxiliary levels.) The value of the machines in the optimal schedule is bounded as before, and the penalties are lower bounded by the cost of the corresponding edges. Thus the optimal schedule is not much smaller than the cost of the path. This finishes the proof for the case of minimizing the penalty plus $\sum f(C_i)$.

The case of minimizing the penalty plus $\max f(C_i)$ is somewhat different. The obstacle is that the cost of a path in the graph should be sum of the costs of edges on certain levels plus the maximum of the costs of edges on the remaining levels; for such a problem we are not able to use the usual shortest path algorithm.

Let M be some bound on $\max f(C_i)$. We use a similar graph as above, with the following modification. We include an edge entering an original node only if its value would be at most M; we set its value to 0. Now the cost of the shortest path is an approximation of the minimal penalty among all schedules with $\max f(C_i) \leq M$. More precisely, similarly as in Lemma 5.3, if there is a schedule with $\max f(C_i) \leq (1 - \varepsilon/3)M$ and total penalty P, the shortest path has cost at most P; on the other hand, from a path with cost P we may construct a schedule with $\max f(C_i) \leq (1 + \varepsilon/3)M$ and penalty P.

Now we solve the optimization problem by using the procedure above polynomially many times. Let s_{min} and s_{max} be the smallest and the largest machine speeds, respectively. Let p_{min} be the minimal processing time of a job, and let T be the total processing time of all jobs. In any schedule, any non-zero completion time is between $b = p_{min}/s_{max}$ and $B = T/s_{min}$. Now we cycle through all values $x = (1+\delta)^i b$, $i = 0, 1, \ldots$, such that $x \leq B$; the constant δ is chosen by the condition (F$*$), as in Section 5. In addition, we consider $x = 0$. The number of such x is polynomial in the size of the number B/b, which is polynomial in the size of the instance. For each x, we compute $M = f(x)$, and find a corresponding schedule with the smallest penalty P by the procedure above. (As a technical detail, we have to round each x to a sufficient precision so that the length of x is polynomial in the size of the instance; this is possible, possible to do so that the ratio between successive values of x never exceeds $1 + 2\delta$, and that is sufficient.) We chose the best of these schedules, and possibly the schedule rejecting all jobs. Since the relative change between any two successive non-zero value of x is at most 3δ, the relative change between the successive value of M is at most $\varepsilon/3$ (by our choice of δ using (F$*$)), and we cover all relevant values of M with sufficient density.

References

[1] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proc. of the 8th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 493–500. ACM-SIAM, 1997.

[2] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *J. of Scheduling*, 1:55–66, 1998.

[3] Y. Azar and L. Epstein. Approximation schemes for covering and scheduling on related machines. In *Proc. of the 1st Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX '98), Lecture Notes in Comput. Sci. 1444*, pages 39–47. Springer-Verlag, 1998.

[4] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie. Multiprocessor scheduling with rejection. In *Proc. of the 7th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 95–103. ACM-SIAM, 1996.

[5] A. K. Chandra and C. K. Wong. Worst-case analysis of a placement algorithm related to storage allocation. *SIAM J. Comput.*, 4:249–263, 1975.

[6] R. A. Cody and E. G. Coffman. Record allocation for minimizing expected retrieval costs on drum-like storage devices. *J. Assoc. Comput. Mach.*, 23:103–115, 1976.

[7] J. Csirik, H. Kellerer, and G. J. Woeginger. The exact LPT-bound for maximizing the minimum completion time. *Oper. Res. Lett.*, 11:281–287, 1992.

[8] D. K. Friesen and B. L. Deuermeyer. Analysis of greedy solutions for a replacement part sequencing problem. *Mathematics of Operations Research*, 6:74–87, 1981.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–completeness*. Freeman, 1979.

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

[11] R. L. Graham. Bounds on multiprocessor timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–429, 1969.

[12] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. Assoc. Comput. Mach.*, 34:144–162, 1987.

[13] D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17:539–551, 1988.

[14] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. Zipkin, editors, *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, pages 445–552. North-Holland, 1993.

[15] S. Sahni. Algorithms for scheduling independent tasks. *J. Assoc. Comput. Mach.*, 23:116–127, 1976.

[16] G. J. Woeginger. A polynomial time approximation scheme for maximizing the minimum machine completion time. *Oper. Res. Lett.*, 20:149–154, 1997.