Approximation Schemes for Scheduling on Parallel Machines

Noga Alon * Yossi Azar † Gerhard J. Woeginger ‡ Tal Yadid §

Abstract

We discuss scheduling problems with m identical machines and n jobs where each job has to be assigned to some machine. The goal is to optimize objective functions that solely depend on the machine completion times.

As a main result, we identify some conditions on the objective function, under which the resulting scheduling problems possess a polynomial time approximation scheme. Our result contains, generalizes, improves, simplifies, and unifies many other results in this area in a natural way.

Keywords: Scheduling theory, approximation algorithm, approximation scheme, worst case ratio, combinatorial optimization.

1 Introduction

In this paper we consider scheduling problems with m identical machines M_i, $1 \leq i \leq m$, and n independent jobs J_j, $1 \leq j \leq n$, where job J_j has *processing time* (or *length*) p_j. A *schedule* is an assignment of the n jobs to the m machines. For $1 \leq i \leq m$, the *completion time* C_i of machine M_i in some fixed schedule equals the total processing time of the jobs that are assigned to M_i. The goal is to find a schedule that optimizes an objective function that solely depends on the machine completion times. More precisely, for a fixed function $f : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$, we consider the problems of

(I). minimizing the objective function $\sum_{i=1}^{m} f(C_i)$,
(II). minimizing the objective function $\max_{i=1}^{m} f(C_i)$.

In the standard scheduling notation of Lawler, Lenstra, Rinnooy Kan and Shmoys [13], these problems are denoted by $P|\cdot|\sum f(C_i)$ and by $P|\cdot|\max f(C_i)$, respectively. Moreover, we will deal with maximization problems that are dual to the problems (I) and (II) i.e. with the

*School of Mathematical Sciences, Tel-Aviv University, Israel. Supported in part by a USA-Israeli BSF grant. E-Mail: noga@math.tau.ac.il

†Dept. of Computer Science, Tel-Aviv University, Israel. Supported in part by the Israel Science Foundation, and by the USA-Israel BSF grant. E-Mail: azar@math.tau.ac.il

‡Institut für Mathematik, TU Graz, Steyrergasse 30, A-8010 Graz, Austria. Supported by the START program Y43-MAT of the Austrian Ministry of Science. E-Mail: gwoegi@opt.math.tu-graz.ac.at

§Dept. of Computer Science, Tel-Aviv University, Israel. E-Mail: yadid@math.tau.ac.il

problem (I') of *maximizing* the objective function $\sum_{i=1}^{m} f(C_i)$ and with the problem (II') of maximizing $\min_{i=1}^{m} f(C_i)$.

Apparently, all non-trivial scheduling problems of the forms (I), (II), (I'), and (II') are NP-complete in the strong sense. Hence, research focused on obtaining polynomial time approximation algorithms for this problem, i.e. fast algorithms which construct schedules whose objective function value is not too far away from the optimum value. We say that an approximation algorithm for a minimization problem has *performance ratio* or *performance guarantee* ρ for some real $\rho \geq 1$, if it always delivers a solution with objective function value at most ρ times the optimum value. Such an approximation algorithm is then called a *ρ-approximation algorithm*. A family of polynomial time $(1+\varepsilon)$-approximation algorithms over all $\varepsilon > 0$ is called a *polynomial time approximation scheme* or PTAS, for short. For *maximization* problems, ρ-approximation algorithms with $\rho \leq 1$ are defined in a symmetric way. Moreover, a PTAS for a maximization problem is a family of polynomial time $(1-\varepsilon)$-approximation algorithms over all $\varepsilon > 0$.

1.1 Four examples

Let us give four illustrating examples for scheduling problems with objective functions of the type (I), (II), (I'), and (II'):

Example 1. Minimizing the maximum machine completion time, i.e. problem (II) with $f(x) = x$. This definitely is the most classical scheduling problem on parallel machines. Already in the 1960s, Graham [10, 11] analyzed the worst case behavior of variants of the greedy-algorithm for this problem: *List Scheduling* (LS, for short) starts with an arbitrarily ordered list of jobs and repeatedly assigns the next job in the list to a machine with currently smallest load. The *Longest Processing Time* rule (LPT, for short) first sorts the jobs by non-increasing processing time and then behaves like LS. Graham proved that LS has performance guarantee 2 and that LPT has performance guarantee 4/3. Sahni [16] constructed a PTAS for the case where the number of machines is not part of the input. Finally, Hochbaum and Shmoys [12] completely settled the approximability status of this problem by constructing a PTAS for the most general case with an arbitrary number of machines.

Example 2. Minimizing the sum of squares of the machine completion times, i.e. problem (I) with $f(x) = x^2$. This problem arises e.g. when placing a set of records on a sectored drum so as to minimize the average latency time (Cody and Coffman [4]) and in other storage allocation problems (Chandra and Wong [3]). Chandra and Wong [3] analyzed the LPT rule for this problem and proved a performance guarantee of 25/24; this result was slightly improved by Leung and Wei [15]. Avidor, Azar and Sgall [2] showed that LS has performance guarantee 4/3.

A more general problem is to minimize the sum of the p-th powers of the machine completion times, i.e. problem (I) with $f(x) = x^p$. Note that this problem is equivalent to minimizing the L_p-norm of the vector $(C_1, \ldots, C_m)$, i.e. to minimize $(\sum_{i=1}^{m} C_i^p)^{1/p}$. Chandra and Wong [3] showed that for every fixed $p \geq 1$, the LPT rule achieves a constant performance ratio whose value depends on p and may be as large as 3/2. Alon, Azar, Woeginger, and Yadid [1] finally exhibited a PTAS for every fixed $p \geq 1$. Their approach exploits the equivalence with the minimization of the L_p-norm and relies heavily on the triangle inequality.

Example 3. Scheduling machines with extensible working times, i.e. problem (I) with

$f(x) = \max\{1, x\}$. This problem arises e.g. in systems with m workers who have regular working times of one time unit and who may make (if needed) extra-paid overtime work. The goal is to assign duties to the workers so as to minimize the total cost, i.e. the fixed cost for regular working times plus the extra cost for overtime work. Dell'Olmo, Kellerer, Speranza, and Tuza [6] showed that the LPT rule has a performance guarantee of 13/11. A straightforward modification of the Hochbaum and Shmoys [12] approach yields a PTAS for this problem.

Example 4. Maximizing the minimum machine completion time, i.e. problem (II') with $f(x) = x$. This problem has applications in the sequencing of maintenance actions for modular gas turbine aircraft engines, see Friesen and Deuermeyer [8]. Deuermeyer, Friesen and Langston [7] showed that the LPT rule has a performance guarantee of 3/4; Csirik, Kellerer and Woeginger [5] tightened this analysis. Finally, Woeginger [17] gave a PTAS for this problem.

1.2 Results of this paper

We identify some conditions on functions f for which the corresponding scheduling problems (I), (II), (I'), and (II') possess a polynomial time approximation scheme. Function $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is *convex* if and only if $f(x + \Delta) + f(y - \Delta) \leq f(x) + f(y)$ holds for all $0 \leq x \leq y$ with $0 \leq \Delta \leq y - x$. Function f is *concave* if and only if $f(x + \Delta) + f(y - \Delta) \geq f(x) + f(y)$ holds for all $0 \leq x \leq y$ with $0 \leq \Delta \leq y - x$. The following condition (F$*$) will be essential throughout the paper.

(F$*$): For all $\varepsilon > 0$ there exists a $\delta > 0$ whose value depends only on ε, such that:

$$\forall x, y \geq 0, \;\; (1-\delta)x \leq y \leq (1+\delta)x \qquad \Longrightarrow \qquad (1-\varepsilon)f(x) \leq f(y) \;\leq\; (1+\varepsilon)f(x)$$

Now our main result on problems (I) and (II) is stated as follows.

Theorem 1.1 *For every non-negative and convex function f that fulfills condition (F$*$), the scheduling problem of minimizing $\sum f(C_i)$ and the scheduling problem of minimizing $\max f(C_i)$ both possess polynomial time approximation schemes. The running time of these PTAS is $O(n)$, where the hidden constant depends exponentially on the reciprocal value of the desired precision.*

Note that the time complexity of this PTAS is essentially best possible for strongly NP-complete scheduling problems: Clearly, one cannot do better than linear in the number of jobs. Moreover, the existence of a PTAS whose running time is also polynomial in the reciprocal value of the precision for a strongly NP-complete problem would imply $P = NP$; cf. page 141 of Garey and Johnson [9].

An argument that is completely symmetric to the proof of Theorem 1.1 will yield a corresponding theorem for the maximization problems (I') and (II').

Theorem 1.2 *For every non-negative concave function f that fulfills condition (F$*$), the scheduling problem of maximizing $\sum f(C_i)$ and the scheduling problem of maximizing $\min f(C_i)$ both possess a polynomial time approximation scheme with linear running time.*

The reader may want to verify that the functions $f(x) = x$, $f(x) = x^p$, and $f(x) = \max\{1, x\}$ as introduced in Examples 1–3 above are convex and satisfies condition (F$*$), and that the function $f(x) = x$ in Example 4 is concave and fulfills condition (F$*$). Hence, our Theorems 1.1 and 1.2 immediately imply the existence of a PTAS for the scheduling problems in Examples 1–4.

Our approximation scheme generalizes the schemes of Sahni [16] and of Hochbaum and Shmoys [12], and in fact it strongly builds on their ideas. However, there are several differences where a major one is dealing with so-called *small* jobs, i.e. jobs whose processing times are smaller than a certain data-dependent threshold. The algorithms in [16, 12] first remove these small jobs, then solve the remaining problem to get an intermediate auxiliary schedule, and at last greedily add the small jobs to the intermediate schedule. A similar approach *provably* does not help in proving Theorems 1.1 and 1.2 (see Example 3.1 in Section 3). Instead, we will find a way of merging the small jobs into packages of reasonable size and then work on with these packages.

Organization of the paper. In Section 2 we construct a polynomial time approximation scheme for scheduling problems (I) where the non-negative, convex function f fulfills property (F$*$). Section 3 shows how to extend these results to scheduling problems (II), (I'), and (II'). Section 4 discusses property (F$*$), and it investigates the relationship between the existence of a PTAS and the worst case behavior of the LPT rule. Finally, Section 5 contains the conclusion.

2 A Generic Polynomial Time Approximation Scheme

Throughout this section, we will deal with an instance I with m machines, n jobs with job processing times $p_1, \ldots, p_n$, and the objective of minimizing $\sum_{i=1}^{m} f(C_i)$ where the non-negative convex function f satisfies (F$*$). Our first goal will be to prove some simple combinatorial observations. Let

$$L \;=\; \frac{1}{m} \sum_{j=1}^{n} p_j \tag{1}$$

denote the average load of the m machines.

Observation 2.1 *If a job J_j has length $p_j \geq L$, then there exists an optimum schedule in which job J_j is the only job assigned to its machine.*

Proof. Suppose that in some optimum schedule, job J_j is assigned together with some other jobs to machine M_1. Then the completion time C_1 of machine M_1 fulfills $C_1 = p_j + \Delta$ with $\Delta > 0$. Moreover, there exists a machine, say M_2, whose completion time C_2 is strictly less than L. But now since $\Delta \leq C_2 - C_1$ then

$$f(C_1) + f(C_2) \;\geq\; f(C_1 - \Delta) + f(C_2 + \Delta) \tag{2}$$

holds by the convexity of the function f. Hence, moving all jobs except J_j from M_1 to M_2 cannot increase the objective function. ∎

With the help of Observation 2.1 we may iteratively assign the very large jobs whose length is $\geq L$: While there is a job whose length is larger than the current average load, we assign this job to a machine and remove both the job and the machine from the setting. Note that in doing this, both the optimal algorithm and the iterative process assign the large jobs to separate machines. Thus, we may assume without loss of generality that in the end no job has length larger than the average load. More specifically, from a given PTAS for the case where none of the jobs has length larger than the average, we can generate a PTAS for the general case, by assigning the large jobs to separate machines and applying the PTAS on the remaining jobs and machines.

Hence, from now on we may (and will) assume that no job has processing time greater or equal to L.

Observation 2.2 *If no job has processing time greater or equal to L, then there exists an optimum schedule in which all machine completion times C_i, $1 \leq i \leq m$, fulfill $\frac{1}{2}L < C_i < 2L$.*

Proof. Among all optimum schedules, consider a schedule that minimizes the auxiliary function $\sum_{i=1}^{m} C_i^2$. We claim that in this optimum schedule $\frac{1}{2}L < C_i < 2L$ holds for all machine completion times.

First suppose that $C_i \geq 2L$ holds for some machine M_i. Similarly as in the proof of Observation 2.1, moving an arbitrary job from M_i to the machine with smallest completion time cannot increase the convex objective function, but does decrease the auxiliary function; a contradiction. Next suppose that $C_i \leq \frac{1}{2}L$ holds for some machine M_i. Consider another machine M_j with $C_j > L$. In case M_j processes some job whose processing time is smaller than $C_j - C_i$, then moving this job from M_j to M_i decreases $\sum_{i=1}^{m} C_i^2$ but does not increase the convex objective function; another contradiction. Otherwise, M_j must process at least two jobs with processing times $\geq C_j - C_i > C_i$. Swapping one of these jobs with the set of jobs processed on M_i again decreases $\sum_{i=1}^{m} C_i^2$ and does not increase the objective function; this is the final contradiction. ∎

2.1 The transformed instance

In this section, we introduce a so-called *transformed instance* that is a kind of rounded version of the original instance I, and that can easily be solved in polynomial time. Indeed, let λ be some fixed positive integer λ. Then the transformed instance $I^{\#}(\lambda)$ of scheduling instance I is defined as follows:

- For every job J_j in I with $p_j > L/\lambda$, instance $I^{\#}(\lambda)$ contains a corresponding job $J_j^{\#}$ whose processing time $p_j^{\#}$ equals p_j rounded up to the next integer multiple of L/λ^2. Note that for the rounded processing time $p_j \leq p_j^{\#} \leq (\lambda+1)p_j/\lambda$ holds.
- Let S denote the total processing time of the jobs in I whose processing times are not greater than L/λ, and let $S^{\#}$ denote the value of S rounded up to the next integer multiple of L/λ. Then instance $I^{\#}(\lambda)$ contains $S^{\#}\lambda/L$ new jobs, each of length L/λ. Clearly, the total number of jobs cannot increase and hence is bounded by n.
- The number of machines remains the same as in instance I.

Note that in instance $I^{\#}(\lambda)$, the processing times of all jobs are integer multiples of L/λ^2. Let $L^{\#} = \frac{1}{m}\sum_{j=1}^{n} p_j^{\#}$ denote the average load of the machines in the transformed instance. By construction,

$$L \;\leq\; L^{\#} \;. \tag{3}$$

Since in I all jobs had lengths smaller than L (and since L is an integer multiple of L/λ^2), by the construction also all jobs in $I^{\#}(\lambda)$ have lengths smaller than L. From now on, jobs (in instance I and in instance $I^{\#}(\lambda)$) with processing times $> L/\lambda$ will be called *big* jobs, and jobs with processing times $\leq L/\lambda$ will be called *small* jobs.

Observation 2.3 *In $I^{\#}(\lambda)$, no job has processing time greater or equal to $L^{\#}$. There exists an optimum schedule in which all machine completion times fulfill $\frac{1}{2}L^{\#} < C_i^{\#} < 2L^{\#}$.* ∎

We now show two alternative methods to find an optimal solution for instance $I^{\#}(\lambda)$ in polynomial time. The first method (cf. Theorem 2.4) is based on simple dynamic programming and is fairly easy to implement. Its time complexity is a polynomial whose exponent depends on λ. The second method (cf. Theorem 2.5) is a more efficient but also more complex algorithm which is based on integer linear programming in a fixed dimension [14].

First, observe that all jobs in $I^{\#}(\lambda)$ have processing times of the form kL/λ^2 with $\lambda \leq k \leq \lambda^2$. We use the following notation to represent the input and the assignments of jobs to machines: The input jobs are represented as a vector $\vec{n} = (n_\lambda, \ldots, n_{\lambda^2})$, where n_k denotes the number of jobs whose processing time equals kL/λ^2. Notice that $n = \sum_{k=\lambda}^{\lambda^2} n_k$. An assignment to a machine is a vector $\vec{u} = (u_\lambda, \ldots, u_{\lambda^2})$, where u_k is the number of jobs of length kL/λ^2 assigned to that machine. The *length* of assignment $\vec{u}$, denoted $C(\vec{u})$, is $\sum_{k=\lambda}^{\lambda^2} u_k \cdot kL/\lambda^2$. Denote by U the set of all assignment vectors $\vec{u}$ with $\frac{1}{2}L^{\#} < C(\vec{u}) < 2L^{\#}$. It is easy to see that each assignment $\vec{u} \in U$ consists of at most 2λ jobs and that therefore $|U| \leq \lambda^{4\lambda}$ holds. By Observation 2.3, when searching for an optimum schedule we only need to consider assignment vectors in U.

First we describe the dynamic programming approach. Let V be the set of vectors corresponding to subsets of the input, i.e. $V = \{\vec{v} : \vec{0} \leq \vec{v} \leq \vec{n}\}$. For every $\vec{v} \in V$ and for every i, $0 \leq i \leq m$, we denote by $\textsc{Val}(i, \vec{v})$ the total cost of the optimum assignment of the jobs in $\vec{v}$ to i machines that only uses assignments from U; in case no such assignment exists, $\textsc{Val}(i, \vec{v}) = +\infty$. It is easy to see that $\textsc{Val}(\cdot, \cdot)$ satisfies the following recurrence:

$$\begin{aligned}
\textsc{Val}(0, \vec{0}) &= 0 \\
\textsc{Val}(1, \vec{v}) &= f(C(\vec{v})) \qquad \text{if } \vec{v} \in U \\
\textsc{Val}(1, \vec{v}) &= +\infty \qquad\;\; \text{if } \vec{v} \notin U
\end{aligned}$$

and for $i \geq 2$

$$\textsc{Val}(i, \vec{v}) \;=\; \min_{\vec{v} \geq \vec{u} \in U} \{f(C(\vec{u})) + \textsc{Val}(i-1, \vec{v} - \vec{u})\}$$

For computing the value of a pair $(i, \vec{v})$, we need to look at no more than $|U|$ possible assignments, and for each such assignment we do a constant amount of work. Thus, the total running time of the resulting dynamic programming algorithm is $O(m \cdot |V| \cdot |U|)$. Since $|V| = \prod_{k=1}^{r}(n_k + 1) \leq n^{\lambda^2}$, we have proved the following theorem.

Theorem 2.4 *For any constant $\lambda \geq 1$, an optimal solution for the instance $I^{\#}(\lambda)$ of problem $P|\cdot|\sum f(C_i)$ can be computed in polynomial time $O(mn^{\lambda^2})$.* ∎

The second method to solve instance $I^{\#}(\lambda)$ is by using integer linear programming (ILP) with a fixed number of variables (cf. Lenstra [14]). As before, U is the set of possible assignments to a machine. For each vector $\vec{u} \in U$, denote by $x_{\vec{u}}$ the numbers of machines that were assigned $\vec{u}$. In these terms, a feasible solution to the problem is a non-negative integer vector $\vec{X}$ such that $\sum_{\vec{u}\in U} x_{\vec{u}} = m$, (i.e. m machines are used) and such that $\sum_{\vec{u}\in U} x_{\vec{u}} \cdot \vec{u} = \vec{n}$ (i.e. all jobs were assigned). An optimum solution is given by the following integer system:

$$\min \sum_{\vec{u}\in U} x_{\vec{u}} \cdot f(C(\vec{u}))$$

subject to

$$\begin{aligned} &\sum_{\vec{u}\in U} x_{\vec{u}} = m \\ &\sum_{\vec{u}\in U} x_{\vec{u}} \cdot \vec{u} = \vec{n} \\ &x_{\vec{u}} \geq 0 \qquad\qquad \forall\ \vec{u} \in U \end{aligned}$$

Notice that the dimension of this integer linear program is $|U| \leq \lambda^{4\lambda}$ and that the number of constraints is $|U| + \lambda^2 - \lambda + 2$, i.e. two constants that do not depend on the input. The time complexity of Lenstra's algorithm [14] is exponential in the dimension of the program but polynomial in the logarithms of the coefficients. Therefore, its running time is $O(\log^{O(1)} n)$ where the hidden constant depends exponentially on λ. Since the program (ILP) can be constructed in $O(n)$ time, we arrive at the following theorem.

Theorem 2.5 *For any constant $\lambda \geq 1$, an optimal solution for the instance $I^{\#}(\lambda)$ of problem $P|\cdot|\sum f(C_i)$ can be computed in time $O(n)$.* ∎

2.2 Transformed instance versus original instance

In this section, we study the relationships between schedules for instance I and schedules for instance $I^{\#}(\lambda)$.

Lemma 2.6 *Let Σ be a schedule for instance I with machine completion times C_i, where $\frac{1}{2}L < C_i < 2L$. Then there exists a schedule $\Sigma^{\#}$ for instance $I^{\#}(\lambda)$ with machine completion times $C_i^{\#}$, where*

$$C_i - \frac{1}{\lambda}L \ \leq\ C_i^{\#} \ \leq\ \frac{\lambda+1}{\lambda}C_i + \frac{1}{\lambda}L \tag{4}$$

Proof. In schedule Σ, replace every big job J_j by its corresponding job $J_j^{\#}$ in $I^{\#}(\lambda)$. This may increase some machine completion times by a factor of at most $(\lambda+1)/\lambda$, but it never can decrease a machine completion time. Next, rearrange on every machine M_i the jobs in

such a way that the small jobs are processed in the end, and let s_i denote their total size. Clearly, $\sum_{i=1}^{m} s_i = S$. There exist integer multiples $s_i^{\#}$ of L/λ for $1 \leq i \leq m$, such that

$$|s_i - s_i^{\#}| \;\leq\; L/\lambda \qquad \text{and} \qquad \sum_{i=1}^{m} s_i^{\#} \;=\; S^{\#}. \tag{5}$$

Removing the small jobs from the current schedule and putting $s_i^{\#}\lambda/L$ jobs of length L/λ instead of them on machine M_i yields the desired schedule $\Sigma^{\#}$. ∎

Lemma 2.7 *Let $\Sigma^{\#}$ be a schedule for instance $I^{\#}(\lambda)$ with machine completion times $C_i^{\#}$, where $\frac{1}{2}L^{\#} < C_i^{\#} < 2L^{\#}$. Then there exists a schedule Σ for instance I with machine completion times C_i where*

$$\frac{\lambda}{\lambda+1}C_i^{\#} - \frac{2}{\lambda}L \;\leq\; C_i \;\leq\; C_i^{\#} + \frac{1}{\lambda}L. \tag{6}$$

Moreover, such a schedule Σ can be computed in $O(n)$ time.

Proof. In schedule $\Sigma^{\#}$, replace every big job $J_j^{\#}$ by its corresponding job J_j in I. This cannot increase the machine completion times; any machine completion time that is decreased is decreased by at most a factor of $\lambda/(\lambda+1)$. Next, let $s_i^{\#}$ denote the number of jobs of length L/λ that are processed on machine M_i in schedule $\Sigma^{\#}$; rearrange on every machine the jobs in such a way that these small jobs are processed in the end. Below, we will show how to partition the small jobs in instance I into m parts $S_1^*, \ldots, S_m^*$ such that the total size of the jobs in part S_i^* is between $(s_i^{\#} - 2)L/\lambda$ and $(s_i^{\#} + 1)L/\lambda$. Then replacing on every machine M_i the $s_i^{\#}$ jobs of length L/λ by the small jobs in S_i^* yields the desired schedule that fulfills inequality (6).

In order to construct the partition $S_1^*, \ldots, S_m^*$, one proceeds as follows. In the first partitioning step, greedily assign to every set S_i^* a number of small jobs from I until the total size of assigned jobs exceeds $(s_i^{\#} - 2)L/\lambda$. Note that the total size of jobs assigned to S_i^* is bounded by $(s_i^{\#} - 1)L/\lambda$. Since

$$\sum_{i=1}^{m}(s_i^{\#} - 1)L/\lambda \;\leq\; S^{\#} - mL/\lambda \;\leq\; S, \tag{7}$$

one can complete the first partitioning step without running out of jobs. In the second partitioning step, greedily add the remaining small jobs to the sets S_i^*, but without letting the total size assigned to S_i^* exceed $(s_i^{\#} + 1)L/\lambda$. It can be easily shown that indeed all small jobs are assigned in the second partitioning step. ∎

2.3 The approximation scheme

Finally, we will put everything together and derive the desired PTAS. Consider some fixed real number ε, $0 < \varepsilon < 1$. Let $\delta > 0$ be a real number that fulfills

$$f(y) \;\leq\; (1 + \frac{\varepsilon}{3})f(x) \qquad \text{for all } x, y \geq 0 \text{ with } (1-\delta)x \leq y \leq (1+\delta)x. \tag{8}$$

Note that by condition (F$*$) such a δ always exists. Define a positive integer $\lambda^* := \lceil 5/\delta \rceil$. For a given instance I of $P|\cdot|\sum f(C_i)$, the approximation scheme performs the following three Steps (S-1) through (S-3).

(S-1) Compute the instance $I^{\#}(\lambda^*)$ from I.

(S-2) Solve instance $I^{\#}(\lambda^*)$ to optimality; call the resulting schedule $\Sigma^{\#}$.

(S-3) Transform schedule $\Sigma^{\#}$ into a schedule Σ for the original instance I as described in the proof of Lemma 2.7.

It is straightforward to perform Step (S-1) in linear time. By Theorem 2.5 and by Lemma 2.7, also Steps (S-2) and (S-3) can be implemented in $O(n)$ time.

Now let OPT denote the value of the objective function in the optimum schedule of I, let OPT$^{\#}$ denote the value of the objective function for the optimum schedule of $I^{\#}(\lambda^*)$, and let APPROX denote the value of the objective function of the schedule for I that is computed in Step (3). First, we claim that

$$\text{OPT}^{\#} \;\leq\; (1+\frac{\varepsilon}{3})\text{OPT} \tag{9}$$

holds: Indeed, by Observation 2.2, there exists an optimum schedule for I whose machine completion times C_i fulfill $\frac{1}{2}L < C_i < 2L$. Then by Lemma 2.6, there exists a schedule for $I^{\#}(\lambda^*)$ with machine completion times $C_i^{\#}$ such that

$$|C_i - C_i^{\#}| \;\leq\; \frac{1}{\lambda^*}C_i + \frac{1}{\lambda^*}L \;\leq\; \frac{3}{\lambda^*}C_i \;\leq\; \delta C_i. \tag{10}$$

By (8), $f(C_i^{\#}) \leq (1+\frac{\varepsilon}{3})f(C_i)$. Summing up this inequality over all machines, one gets that the objective value of the schedule for $I^{\#}(\lambda^*)$ in Lemma 2.6 is at most $(1+\frac{\varepsilon}{3})$OPT. This implies the correctness of (9).

Next, we claim that

$$\text{APPROX} \;\leq\; (1+\frac{\varepsilon}{3})\text{OPT}^{\#}. \tag{11}$$

Indeed, by Observation 2.3, there exists an optimum schedule for $I^{\#}(\lambda^*)$ whose machine completion times $C_i^{\#}$ fulfill $\frac{1}{2}L^{\#} < C_i^{\#} < 2L^{\#}$. Then by Lemma 2.7, there exists a schedule for the original instance I with machine completion times C_i such that

$$|C_i - C_i^{\#}| \;\leq\; \frac{1}{\lambda^*+1}C_i^{\#} + \frac{2}{\lambda^*}L \;\leq\; \frac{1}{\lambda^*}C_i^{\#} + \frac{2}{\lambda^*}L^{\#} \;\leq\; \frac{5}{\lambda^*}C_i^{\#} \;\leq\; \delta C_i^{\#}. \tag{12}$$

Similarly as above, inequality (8) now yields the correctness of (11). Finally, combining inequalities (9) and (11) yields that

$$\text{APPROX} \;\leq\; (1+\frac{\varepsilon}{3})^2 \cdot \text{OPT} \;\leq\; (1+\varepsilon)\text{OPT}. \tag{13}$$

Summarizing, the procedure (S-1) through (S-3) finds in polynomial time a schedule for I whose objective value is at most a factor of $(1+\varepsilon)$ away from the optimum. This completes the proof of Theorem 1.1 for problem $P|\cdot|\sum f(C_i)$.

3 Discussion of the PTAS

In this section, we explain how to carry over the PTAS in Section 2 from problem (I) to problems (II), (I') and (II').

First let us take a look at problem (II), i.e. minimizing $\max_{i=1}^{m} f(C_i)$. It is easy to see that an analogous statement as in Observation 2.2 still holds true for problem (II), i.e. there is an optimum schedule in which all C_i are between $\frac{1}{2}L$ and $2L$. The transformed instance is formulated as in Section 2.1 and solved in $O(n)$ time similarly as in Theorem 2.5. The only difference is that one has to use a slightly different objective function in the integer linear program (ILP):

$$\min\ z$$

subject to

$$\begin{aligned}
&\sum_{\vec{u}\in U} x_{\vec{u}} \;=\; m \\
&\sum_{\vec{u}\in U} x_{\vec{u}} \cdot \vec{u} \;=\; \vec{n} \\
&x_{\vec{u}} \;\leq\; m \cdot y_{\vec{u}} \\
&z \;\geq\; y_{\vec{u}} \cdot f(C(\vec{u})) \\
&0 \leq x_{\vec{u}},\ x_{\vec{u}} \text{ integer} \qquad \forall\ \vec{u} \in U \\
&y_{\vec{u}} \in \{0,1\} \qquad \forall\ \vec{u} \in U
\end{aligned}$$

The 0-1 variables $y_{\vec{u}}$ indicate whether the assignment $\vec{u}$ is used ($y_{\vec{u}} = 1$) or not ($y_{\vec{u}} = 0$). In case the assignment $\vec{u}$ is used, then the term $y_{\vec{u}} \cdot f(C(\vec{u}))$ contributes to the objective function. Finally, the arguments in Section 2.3 can be used to show that Steps (S-1)–(S-3) yield a $(1+\varepsilon)$-approximation algorithm.

For the maximization problems (I') and (II'), again a statement as in Observation 2.2 holds true, i.e. there always is an optimum schedule in which all C_i are between $\frac{1}{2}L$ and $2L$ (recall that now we are dealing with *concave* functions). The transformed instance is formulated as in Section 2.1. It can be solved (with an appropriately modified objective function) in $O(n)$ time via Lenstra's algorithm as in Theorem 2.5. Similarly as in Section 2.3, one can argue that Steps (S-1)–(S-3) always yield a schedule that is at most a factor of $(1-\varepsilon)$ away from the optimum. The details are left to the reader. This completes the proof of Theorem 1.2.

Finally, let us show that a straightforward application of the methods of Sahni [16] and of Hochbaum and Shmoys [12] cannot yield a proof of Theorems 1.1 and 1.2. This method in [16, 12] first removes the small jobs that are smaller than some threshold τ, then solves the remaining problem to get an intermediate schedule, and at last greedily adds the small jobs to this intermediate schedule.

Example 3.1 *Consider an instance with $m = 3$ machines, 6 jobs with lengths $13, 9, 9, 6, 6, 6$ and $10/\tau$ tiny jobs, each of length $\tau/2$; note that the overall length of the tiny jobs is 5. The objective is to minimize $\sum C_i^2$. It is straightforward to verify that the optimum schedule*

assigns to every machine a total processing time of 18*: three jobs of length 6 to machine* M_1*, two jobs of length 9 to machine* M_2*, and the remaining jobs to machine* M_3*. The optimum objective value equals 972. If one removes the tiny jobs, then the unique optimum schedule for the jobs* $13, 9, 9, 6, 6, 6$ *is to assign* 13 *and* 6 *to machine* M_1*,* 9 *and* 6 *to* M_2*, and* 9 *and* 6 *to* M_3*. No matter, how one adds the tiny jobs to this intermediate schedule, one can never reach a schedule with objective value better than* $973\frac{1}{2}$*.*

4 Approximation schemes, condition (F∗), and the LPT

In this section, we investigate the scheduling problem (I), i.e. problem $P|\cdot|\sum f(C_i)$. We discuss the impact of property (F∗) for functions on the approximability behavior of problem $P|\cdot|\sum f(C_i)$. Moreover, we discuss the relationship between the existence of a PTAS and the worst case behavior of the LPT rule.

First, let us consider the function $f(x) = 2^x$, $x \in \mathbb{R}_0^+$. It can easily be seen that function 2^x is non-negative and convex, but does not satisfy property (F∗). Hence, the following result demonstrates that our main result in Theorem 1.1 does not hold true without imposing condition (F∗).

Observation 4.1 *Unless* $P = NP$*, the problem* $P|\cdot|\sum 2^{C_i}$ *does not allow a polynomial time approximation algorithm with finite performance guarantee. Especially, the problem does not allow a PTAS.*

Proof. Suppose that there exists a polynomial time approximation algorithm with finite performance guarantee $\rho > 1$. Define an integer $\alpha = \lceil \log_2 \rho \rceil + 2$. For an integer Z and for a set $z_1, \ldots, z_n$ of positive integers with $\sum_{j=1}^{n} = 2Z$, it is NP-complete to decide whether the z_j can be partitioned into two sets whose elements sum up to Z (see Garey and Johnson [9]).

Define a corresponding scheduling instance with $m = 2$ machines and n jobs J_j, $1 \leq j \leq n$, with lengths $p_j = \alpha z_j$. It is easy to see that exactly one of the following two cases holds: (i) The z_j can be partitioned into two sets whose elements sum up to Z and, hence, the optimum objective value equals $2^{\alpha Z+1}$. (ii) The z_j can *not* be partitioned into two sets whose elements sum up to Z and, hence, the optimum objective value is at least $2^{\alpha Z+\alpha}$. Since the approximation algorithm could distinguish between these two cases in polynomial time, it would be able to solve the NP-complete partition problem in polynomial time. The claim follows. ∎

Note that the non-approximability result in Observation 4.1 does *not* only rely on the exponential growth rate of function 2^x: An analogous statement can be proved e.g. for function $f(x) = (x-1)^2$. We do not know whether Theorem 1.1 still holds true without imposing convexity of function f.

Next let us discuss the relationship between the existence of an approximation scheme for a scheduling problem, and between the behavior of the LPT rule (cf. Section 1.1) for this problem. We offer the following conjecture.

Conjecture 4.2 *Let* f *be a non-negative, convex function* $\mathbb{R}_0^+ \to \mathbb{R}_0^+$ *and let* $P|\cdot|\sum f(C_i)$ *be the corresponding scheduling problem. Then the following three statement are all equivalent.*

(i) $P|\cdot|\sum f(C_i)$ *possesses a polynomial time approximation algorithm with finite performance guarantee.*

(ii) For $P|\cdot|\sum f(C_i)$*, the LPT rule has finite performance guarantee.*

(iii) $P|\cdot|\sum f(C_i)$ *possesses a PTAS.*

We are able to prove this conjecture for the case where f is not only convex but also satisfies property (F$*$). In this case, Theorem 1.1 implies the truth of (iii) and (i), and the following observation yields the truth of (ii).

Observation 4.3 *Let* f *be a non-negative, convex function* $\mathbb{R}_0^+ \to \mathbb{R}_0^+$ *that satisfies property (F*$*$*). Then the LPT rule for problem* $P|\cdot|\sum f(C_i)$ *has finite performance guarantee.*

Proof. Let $L = \frac{1}{m}\sum_{j=1}^{n} p_j$. Since the LPT rule assigns every job with processing $\geq L$ to its own machine, we may assume by arguments that are similar to those applied at the beginning of Section 2 that no job has processing time greater or equal to L. Next, it can be shown that in the schedule produced by LPT, every machine completion time C_i fulfills $\frac{1}{2}L < C_i < 2L$. By Observation 2.2, an analogous inequality holds for the machine completion times in an optimum schedule. Consequently, every machine completion time in the LPT schedule is at most a factor 4 away from the corresponding machine completion time in the optimum schedule.

Now consider condition (F$*$) with $\varepsilon = 1$: We get that there exists a $\delta > 0$, such that for all $y \leq (1+\delta)x$ the inequality $f(y) \leq 2f(x)$ holds. Define k as the smallest integer that fulfills $(1+\delta)^k \geq 4$. Then for all $y \leq 4x$ the inequality $f(y) \leq 2^k f(x)$ holds. Combining this inequality with the above discussion on the LPT schedule, one gets that the LPT algorithm indeed has finite performance guarantee 2^k. ∎

For general functions f, the worst case behavior of the LPT rule does not seem to be a good indicator for the approximability of a scheduling problem: It is easy to find non-convex functions f for which problem $P|\cdot|\sum f(C_i)$ possesses a PTAS or is even solvable in polynomial time, whereas the LPT rule does not have finite performance guarantee; take e.g. $f(x) = \ln(1+x)$. Moreover, there exist non-convex functions f for which problem $P|\cdot|\sum f(C_i)$ does not possesse a PTAS (unless P=NP) whereas the LPT rule does have finite performance guarantee; take e.g. the function f with $f(1) = 1$ and $f(x) = 2$ for $x \neq 1$.

5 Conclusion

We have shown that for a large set of scheduling problems on parallel machines whose objective functions solely depend on the machine completion times, there exist polynomial time approximation schemes. The running times of our algorithms are linear in the number of jobs and machines. Our result covers as special cases the approximation schemes of Sahni [16], Hochbaum and Shmoys [12], Alon, Azar, Woeginger, and Yadid [1], and Woeginger [17].

We pose as an open problem to derive similar results for scheduling problems on parallel machines whose objective functions solely depend on the *job* completion times.

References

[1] N. Alon, Y. Azar, G.J. Woeginger, and T. Yadid, Approximation schemes for scheduling, in *Proceedings of the 8th Symposium on Discrete Algorithms 1997*, 493–500.

[2] A. Avidor, Y. Azar, and J. Sgall, Ancient and new algorithms for load balancing in the l_p norm, In *Proc. 9th ACM-SIAM Symp. on Discrete Algorithms 1998*, to appear.

[3] A.K. Chandra and C.K. Wong, Worst-case analysis of a placement algorithm related to storage allocation, *SIAM Journal on Computing* **4**, 1975, 249–263.

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

[5] J. Csirik, H. Kellerer and G.J. Woeginger, The exact LPT-bound for Maximizing the Minimum Completion Time, *Operations Research Letters* **11**, 1992, 281–287.

[6] P. Dell'Olmo, H. Kellerer, M.G. Speranza, and Zs. Tuza, Partitioning items in a fixed number of bins to minimize the total size, manuscript, University of Brescia, Italy, 1996.

[7] B.L. Deuermeyer, D.K. Friesen and M.A. Langston, Scheduling to maximize the minimum processor finish time in a multiprocessor system, *SIAM J. Alg. Disc. Meth.* **3**, 1982, 190–196.

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

[9] M.R. Garey and D.S. Johnson, *Computers and Intractability*, Freeman and Company, San Francisco, 1979.

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

[11] R.L. Graham, Bounds on multiprocessing timing anomalies, *SIAM J. Applied Mathematics* **17**, 1969, 416–429.

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

[13] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, Sequencing and scheduling: algorithms and complexity, in *Handbooks in Operations Research and Management Science, Vol. 4*, North Holland, 1993, 445–522.

[14] H.W. Lenstra, Integer programming with a fixed number of variables, *Mathematics of Operations Research* **8**, 1983, 538–548.

[15] J.Y.T. Leung and W.D. Wei, Tighter bounds on a heuristic for a partition problem, *Information Processing Letters* **56**, 1995, 51–57.

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

[17] G.J. Woeginger, A polynomial time approximation scheme for maximizing the minimum machine completion time, *Operations Research Letters* **20**, 1997, 149–154.