On Minimizing Average Weighted Completion Time: A PTAS for the Job Shop Problem with Release Dates*

Aleksei V. Fishkin[1], Klaus Jansen[1], and Monaldo Mastrolilli[2]

[1] Institut für Informatik und Praktische Mathematik,
Christian-Albrechts-Universität zu Kiel, Olshausenstrasse 40, 24118 Kiel, Germany,
{avf,kj}@informatik.uni-kiel.de

[2] IDSIA-Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Manno,
Switzerland, monaldo@idsia.ch

Abstract. We consider the non-preemptive job shop scheduling problem with release dates and the average weighted completion time objective. We propose here a polynomial-time approximation scheme (PTAS) for the case when the number of machines is constant and each job consists of at most a constant number of operations. This substantially improves on previous results [4], adds to a number of techniques [1,2,6,10,22], and gives some answers to the questions mentioned in [20,23].

Keywords: Approximation, PTAS, job shop, scheduling.

1 Introduction

In this paper we consider the non-preemptive job shop scheduling problem with release dates, a fixed number of machines, a fixed number of operations per job, and the average weighted completion time objective, denoted as $Jm\,|op \leq \mu, r_j|\sum w_j C_j$ [15,19]. Formally, we are given a set of m machines $M = \{1, 2, \ldots, m\}$ and a set of n jobs $J = \{1, 2, \ldots, n\}$. Each job j ($j = 1, 2, \ldots, n$) consists of $\mu \geq 2$ operations $o_{1j}, \ldots, o_{\mu j}$ that have to be processed in the given order, has a weight w_j, which defines its importance, and a release date r_j, before which it cannot be started. Each operation o_{ij} ($i = 1, 2, \ldots, \mu$) requires a machine $\tau_{ij} \in M$ and has a processing time p_{ij}. Each job can be processed only by one machine at a time and each machine can process only one job at a time. Here we assume that m and μ are fixed and the goal is to find a non-preemptive feasible schedule which minimizes *average weighted completion time* $\sum w_j C_j$, where C_j denotes the completion time of job j.

* Supported by EU-project CRESCCO, IST-2001-33135, by EU-project APPOL I & II, IST-1999-14084, IST-2001-32007, by SNSF project 21-55778.98, and grant HPRN-CT-1999-00106.

T. Ibaraki, N. Katoh, and H. Ono (Eds.): ISAAC 2003, LNCS 2906, pp. 319–328, 2003.

Previous Results. The first polynomial-time approximation scheme (PTAS) for a strongly NP-hard scheduling problem minimizing the average weighted completion time was given for scheduling jobs with no release dates on identical parallel machines $P||\sum w_jC_j$ [22]. Then recently it has been proved in [1, 6] that there are PTASs for many different variants of classical scheduling problems with release dates and the average weighted completion time objective. These include scheduling on identical parallel machines $P|r_j|\sum w_jC_j$, on related machines $Q|r_j|\sum w_jC_j$, and on a fixed number of unrelated machines $Rm|r_j|\sum w_jC_j$ with and without preemptions.

The job shop scheduling problem is an important generalization of scheduling on parallel machines. However, it seems to be harder for approximating even in the case of unit weights and no release dates. Regarding the worst-case complexity, problem $J2|op \le 2|\sum C_j$ with two machines and at most two operations per job is already strongly NP-hard [12]. This negative result was strengthened in [17], where it was proven that the general job shop scheduling problem $J||\sum C_j$ with arbitrary (not fixed) number of machines m and number of operations per job μ is MAX-SNP-hard. Contrasting this with the fact that for identical parallel machines only the general problem $P||\sum C_j$ is strongly NP-hard, while $Pm||\sum C_j$ is just weakly NP-hard, indicates that computing optimal or approximate schedules for the job shop version is likely to be much harder.

Indeed, for about 20 years only a simple $O(m)$-approximation algorithm for $J||\sum C_j$ [14] has been known. Then, in [4] it that shown that there is a $(5.78+\varepsilon)$-approximation algorithm for problem $Jm|op \le \mu, r_j|\sum w_jC_j$. Later, a general $O((\log(m\mu)/\log\log(m\mu))^2)$- approximation algorithm for problem $J|r_j|\sum w_jC_j$ was presented in [20].

New Results. Here we obtain a PTAS for $Jm|op \le \mu, r_j|\sum w_jC_j$. In order to be able to cope with the problem we employ the well-known input transformation technique [24] and refine some recent sophisticated approximation techniques for the average completion time objective function. These partially include the interval time-partitioning technique [4,16], the techniques of geometric rounding, time stretching, and weight-shifting [1,3], our techniques of LP relaxation and rounding [2,10], and our PTAS for the makespan version of the problem [9,18]. Our main goal is to perform several *transformations* that simplify the input without dramatically increasing the objective value such that the final result is amenable to a fast dynamic programming solution. At the beginning, we round all processing times and release dates to integer powers of $(1+\varepsilon)$. This breaks the time line into contiguous *geometrically increasing intervals* $I_x = ((1+\varepsilon)^x, (1+\varepsilon)^{x+1}]$, $x \in \mathbb{Z}$, where release dates only happen at the beginning of intervals. Furthermore, we round weights such that all w_j/p_j are distinct. Next, we define *main* and *negligible* operations of a job. Here, negligible operations are very small and can be rounded to zero, whereas main operations are very big and can rounded to some particular fractions of the total job length. This makes the problem structure much simpler. We show that any job completes within a constant number of intervals in a schedule, and partition all jobs into a constant number of subsets sharing similar characteristics, called *profile*. Our next idea is to classify the jobs as huge and tiny. In particular, a job is *huge* with respect to an interval if its length is at least ε^2/q^* times the size of the interval, and *tiny* otherwise. For one interval I_x, all huge jobs of the same profile and *size* (a power of $1+\varepsilon$) can be prioritized by decreasing weights w_j, whereas all tiny jobs of the same profile can be prioritized by decreasing ratio w_j/p_j, called a modified Smith's rule [28]. In order to prove the latter statement we define the value of parameter q^*. We first use a subroutine for "assigning" tiny jobs to intervals, and then as a subroutine for "packing" jobs in single intervals. For the first step we use an LP formulation and a special rounding procedure from [18]. For the second step, we use an adopted PTAS for the makespan version of the problem [9]. Finally, the weight-shifting technique is applied. If too many jobs are released at interval I_x, some of them have to wait to be processed. Shifting refers to the process of moving the excess jobs to the next interval I_{x+1}. We enforce special *compact instances* in which there is only a constant number of jobs at any release date. Then, we apply dynamic programming which integrates all previous components. The obtained PTAS is a sequence of the former instance transformations coupled with the dynamic programming algorithm. By an appropriate combination of these ideas and a careful analysis of the algorithm, we prove the following result:

Theorem 1. *There is a PTAS for $Jm|op \leq \mu, r_j|\sum w_j C_j$ that computes for any fixed m, μ and $\varepsilon > 0$ accuracy, $(1+\varepsilon)$-approximate solutions in $O(n \log n)$ time.*

This substantially improves on previous results [4]. Furthermore, we show that a right combination of all ideas – the input transformation technique, the idea of intervals, the idea of huge-tiny jobs, the weight-shifting technique, an LP relaxation (formulation), rounding, a PTAS for the makespan version of the problem, and dynamic programming – is quite a powerful method for the design of PTASs for problems with the average weighted completion time objective function. Indeed, these generalizations significantly add to a number of techniques [1,2,6,10, 22]. One of the most interesting aspects here is that we do not really use a PTAS for the makespan version of the problem as a subroutine, that was previously considered as an interesting question in [20], but rather as a part of our proof technique. Furthermore, following the same line of ideas we can prove the existence of PTASs for many other scheduling models, e.g. open shop, flow shop, dag shop, and their preemptive or multiprocessor versions. Accordingly, we can prove that there are PTASs for the two-machine flow shop problem $F2||\sum C_j$ and for the two-machine open shop problem $O2||\sum C_j$, that was mentioned in [23] as a result which can be the first step to understanding the approximability of shop scheduling problems with the sum of completion time objective function.

The paper is organized as follows. In Section 2 we give some simple preliminary results, which can be also found in [1,6,10]. In Sections 3 and 4 we define negligible operations and perform the main structuring step of the algorithm. In Section 4.1 we define job profiles and give some related definitions. In Section 4.2 we formulate a modified Smith's rule. In Section 5 we complete the algorithm.

2 Preliminaries

To simplify notation we will use throughout the paper that $1/\varepsilon$ is integer (and in particular $\varepsilon < 1/2^{m\mu}$), and use OPT to denote the objective value of the optimal schedule. For an operation o_{ij}, we use S_{ij} and C_{ij} to denote the start and completion time of o_{ij}. For a job j we will use C_j and S_j to denote the completion and start time of j, and use $\ell_j = \sum_{i=1}^{\mu} p_{ij}$ to denote the length of j. For a job set X, we will use $D(X)$ to denote the total length of the jobs in X, that is $\sum_{j\in X} \ell_j$.

As a preliminary step of our algorithm, here we perform some basic *transformations* that simplify the input problem. Many of our transformations are thought modifications applied to the optimal schedule to argue that some schedule nearly as good has very simple structure. In this case, Then, we say that *with* $1+\varepsilon$ *loss*, we can *assume* some properties in any schedule. Others our transformations are actual simplifying modifications of the instance that run in polynomial time and do not increase the objective value too much. Then, we say that *with* $1+\varepsilon$ *loss* and in $O(n)$ time, we can *enforce* some properties in an instance. For a more clear understanding, see lemmas in the next two paragraphs.

Geometric Rounding. The first simplification creates a well-structured set of lengths, release dates, and weights:

Lemma 1. *With $1+3\varepsilon$ loss and in $O(n)$ time, we can enforce all ℓ_j and r_j be integer powers of $1+\varepsilon$ and all w_j/ℓ_j be distinct.*

Proof Sketch. Multiply all r_j and ℓ_j by $1+\varepsilon$, and then *decrease* each date and length to the next *lower* integer power of $1+\varepsilon$. Since $\ell_j(1+\varepsilon) = \sum_{i=1}^{\mu}(1+\varepsilon)p_{ij}$, we decrease all $(1+\varepsilon)p_{ij}$ by the same factor as ℓ_j. The objective value of the final schedule is at most $(1+\varepsilon)OPT$. Regarding weights, we multiply all w_j by $1+\varepsilon$, and then *decrease* some of them by small values until all w_j/ℓ_j differ. This increases the objective function value by at most a factor of $1+\varepsilon$. □

This result guarantees that there are only a small number of distinct processing times and release dates to worry about, and lets us break the time line into *geometrically increasing intervals*, where release dates only happen at the beginning of intervals, that is useful for later techniques and dynamic programming.

For an arbitrary integer x, define $R_x = (1+\varepsilon)^x$. We partition $(0,\infty)$ into disjoint intervals of the form $I_x := [R_x, R_{x+1})$. We will use $|I|_x$ to refer to the size value $(R_{x+1} - R_x) = \varepsilon R_x$, which is ε times its start time. Now all release dates are of the form $R_x = (1+\varepsilon)^x$ for some integer x.

Schedule-Stretching. We can enforce that no operation starts too early or crosses too many intervals:

Lemma 2. *With $1+\varepsilon$ loss, we can assume $S_{ij} \geq \varepsilon p_{ij}$ for all jobs j.*

Proof Sketch. Multiply all C_{ij} by $1+\varepsilon$ and increase S_{ij} to match (without changing p_{ij}). The objective value of the final schedule is at most $(1+\varepsilon)OPT$, and all starting times $(1+\varepsilon)C_{ij} - p_{ij} \geq (1+\varepsilon)p_{ij} - p_{ij}$ are at least εp_{ij}. □

Lemma 3. *With $1+\varepsilon$ loss, we can assume that each operation crosses at most $s^* = \lceil \log_{1+\varepsilon}(1+\frac{1}{\varepsilon}) \rceil$ intervals.*

Proof Sketch. By the above lemma, all $C_{ij} = S_{ij} + p_{ij} \leq S_{ij} + S_{ij}/\varepsilon = S_{ij}\left(1+\frac{1}{\varepsilon}\right)$. If $S_{ij} \in [R_x, R_{x+1})$, s^* intervals following I_x cover $[S_{ij}, \left(1+\frac{1}{\varepsilon}\right) S_{ij}]$. □

3 Main and Negligible Operations

Consider a job j. Let $\ell_j = \sum_{i=1}^{\mu} p_{ij}$ be its length. Let operations o_{ij} ($i = 1, \ldots, \mu$) be indexed by $i_1, i_2, \ldots, i_\mu$ such that $p_{i_1 j} \geq p_{i_2 j} \geq \ldots \geq p_{i_\mu j}$. Then, if there exist some $k \in \{1, \ldots, \mu\}$ such that

$$\varepsilon^{4\mu} \cdot p_{i_k j} > \sum_{s=k+1}^{\mu} p_{i_s j}, \tag{1}$$

then we select the smallest value of k and define operations $o_{i_{k+1} j}, \ldots, o_{i_\mu j}$ be *negligible*, and operations $o_{i_1 j}, \ldots, o_{i_k j}$ be *main.*

Lemma 4. *Each main operation o_{ij} has processing time $p_{ij} \geq \varepsilon^{5\mu^2} \cdot \ell_j$.*

Proof Sketch. Omitted. □

4 Main Structuring Step

Here we perform several main transformations that structure the problem. By combining several techniques we eliminate all negligible operations and round all main operations:

Lemma 5. *With $1+3\varepsilon$ loss and in $O(n)$ time, for all jobs j we can enforce all operation processing times $p_{ij} = \pi_{ij} \cdot \ell_j$, where*

$$\pi_{ij} \in \left\{ z \cdot \varepsilon^{5\mu^2+2} \mid z = 0, 1 \ldots, \frac{1}{\varepsilon^{5\mu^2+2}} \right\}.$$

Proof Sketch. Omitted. □

Similarly, we can move release dates.

Lemma 6. *With $1+\varepsilon$ loss and in $O(n)$ time, we can enforce r_j be at least $\varepsilon^{10\mu^2} \cdot \ell_j$ for all jobs j.*

Proof Sketch. Omitted. □

Finally, we can prove that no job can cross too many intervals.

Lemma 7. *With $1+\varepsilon$ loss, each job crosses at most a constant number of intervals e^*.*

Proof Sketch. Omitted. □

4.1 Profiles, Huge and Tiny Jobs, Local Profiles and Patterns

Here we introduce some important definitions. We first define job profiles. Informally, two jobs have the same profile if they have the same sequence of required processors, and although they can differ in length, their operation processing times correspond to the same sequence of length multiples. Next, we define huge and tiny jobs. Here we introduce a special parameter q^* which value is defined later in Lemma 12, respectively. Finally, we define local profiles and patterns for tiny jobs. This allows us to formulate Smith's rule for tiny jobs in Section 4.2.

Profiles. Consider a job j. By Lemma 5, each operation o_{ij} $(i = 1, \ldots, \mu)$ has processing time $\pi_{ij} \cdot \ell_j$ and requites machine τ_{ij}. Then, μ-tuples $\pi_j = (\pi_{ij})_{i=1}^{\mu}$ and $\tau_j = (\tau_{ij})_{i=1}^{\mu}$ are called the *execution* and *machine* profile of job j, respectively.

We say that two jobs have the same profile $\varphi = (\pi, \tau)$, if they have the same execution profile π and machine profile τ. Notice that two jobs of profile φ can only differ in their length and release dates. Furthermore, as a consequence of Lemma 5 we can prove the following:

Lemma 8. *The number of distinct profiles is bounded by a constant ν^*.*

Huge and Tiny Jobs. We say that a job j is *huge* in an interval I_x if its length $\ell_j \geq \varepsilon^2 |I|_{x(j)}/q^*$, and *tiny* otherwise. The value of parameter $q^* \gg 1$ is defined later in Lemma 12, respectively. We will write H_x and T_x to denote sets of huge and tiny jobs released at R_x (H for huge and T for tiny). As in Lemma 2, we use time-stretching to "clean up" a schedule:

Lemma 9. *With $1 + \varepsilon$ loss, we can assume that no tiny operation crosses an interval in a schedule.*

Furthermore, by using Lemmas 1 and 6 we can prove the following:

Lemma 10. *There is at most a constant number $z^* = O(q^*)$ of distinct sizes (ℓ_j powers of $1 + \varepsilon$) in H_x.*

Local Profiles and Patterns. Take an optimal schedule. Consider a tiny job j. Let $x(j)$ be the index for which $R_{x(j)} = r_j$. Let $y(j)$ and $z(j)$ be whose indices for which $S_j \in I_{y(j)}$ and $C_j \in I_{z(j)}$. Then, job j runs in intervals $I_{y(j)}, \ldots, I_{z(j)}$. By Lemma 9, no operation of job j crosses an interval. Hence, the set O_j of all operations $o_{1j}, \ldots, o_{\mu j}$ "splits" into a constant number of subsets, $O_j^{y(j)}, \ldots, O_j^{z(j)}$, where each subset $O_j^x \subseteq O_j$ consists of operations which "fall" into interval I_x, for $x = y(j), \ldots, z(j)$.

Now assume that tiny job j has some profile $\varphi = (\pi, \tau)$, where two μ-tuples $\pi = (\pi_i)_{i=1}^{\mu}$ and $\tau = (\tau_i)_{i=1}^{\mu}$. Then, we have that $\pi_i = \pi_{ij}$ and $\tau_i = \tau_{ij}$ for all operations $o_{ij} \in O_j$. Informally, we can say that the operations of set O_j "form" profile $\varphi = (\pi, \tau)$. If we restrict ourselves to the operations of set O_j^x, we can define a $2|O_j^x|$-tuple $\bar{\varphi}^x = (\bar{\pi}^x, \bar{\tau}^x)$ such that $\bar{\pi}_i^x = \pi_{ij}$ and $\bar{\tau}_i^x = \tau_{ij}$ for all operations $o_{ij} \in O_j^x$. In this case, we can also say that the operations of set O_j^x "form" *local profile* $\bar{\varphi}^x$ in interval I_x, $x = y(j), \ldots, z(j)$. In other words,

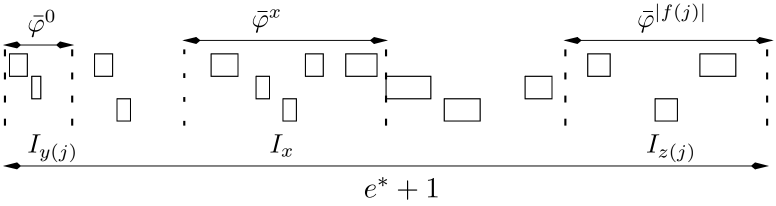

Fig. 1. Local profiles $\bar{\varphi}^0, \bar{\varphi}^1, \ldots, \bar{\varphi}^{|f(j)|}$ of job j

operations of tiny jobs "locally" form "profiles". Notice that operations of two tiny jobs with different profiles can "form" the same local profile in an interval.

We say that a tiny job j has *pattern* $f(j) =< \bar{\varphi}^0, \bar{\varphi}^1, \ldots, \bar{\varphi}^{|f(j)|} >$ in a schedule if starting in interval $I_{y(j)}$ it completes in interval $I_{y(j)+|f(j)|} = I_{z(j)}$, and in each interval $I_{y(j)+k}$ the operations of tiny job j "form" local profile $\bar{\varphi}^k$, for $k = 0, \ldots, |f(j)|$. For an illustration see Figure 1. Notice some local profiles $\bar{\varphi}^k$ can be empty, but the combination of all local profiles gives the profile of tiny job j. By Lemma 7 any tiny job crosses at most a constant number e^* of intervals. By Lemma 8 there is at most a constant number ν^* of profiles. Hence, we can prove the following:

Lemma 11. *There is at most a constant number $\bar{\nu}^*$ of distinct local profiles, and at most a constant number f^* of distinct patterns.*

4.2 Scheduling Tiny Jobs: Smith's Rule

We first need to introduce some notations. Consider an optimal schedule. Then, for a tiny job j we can define two indices $x(j) \leq y(j)$ and pattern $f(j)$ such that job j is released at $R_{x(j)}$, starts at $S_j \in I_{y(j)}$ having pattern $f(j)$ and completes at $C_j \in I_{y(j)+|f(j)|}$.

Smith's rule. Let k and j be two tiny jobs with $x(k) \leq x(j)$ (here $r_k \leq r_j$) and $\frac{w_j}{\ell_j} < \frac{w_k}{\ell_k}$ (see Lemma 1). We say that tiny jobs *obey* Smith's rule if $y(k)+|f(k)| \leq y(j) + |f(j)|$ ($C_k \leq C_j$) for all such pairs of jobs j and k. In other words, if the two jobs are available in an interval, then job k of greater value w_k/ℓ_k completes not later than job j with respect to intervals $I_{y(k)}$, $I_{y(j)}$ and patterns $f(k)$, $f(j)$. Here we are interested in the following result:

Lemma 12. *The value of parameter $q^* = O(\varepsilon, m, \mu)$ can be defined such that with $1 + 7\varepsilon$ loss, for each profile φ we can assume that tiny jobs of profile φ obey Smith's rule in a schedule.*

Proof Sketch. Omitted. $\square$

5 Weight-Shifting and Dynamic Programming

Weight-Shifting. Assume that at some release date R_x we have a lot of huge jobs (H_x) and tiny jobs (T_x). Which jobs can wait until the next interval? Take one profile φ. The jobs of $H_x(\varphi)$ having the same size must complete by decreasing weights w_j. By Lemmas 10 and 12, there is at most a constant number of such sizes. By Lemma 12, the jobs of $T_x(\varphi)$ must complete by decreasing w_j/ℓ_j. By Lemma 7, all jobs that start in I_x must complete within the next e^* intervals. We only select the jobs that can be potentially scheduled in I_x:

Lemma 13. *With $1+O(\varepsilon)$ loss and in $O(n \log n)$ time, we can enforce $D(T_x) \leq t^* \cdot |I|_x$ and $|H_x| \leq H^*$ at each release date R_x, where t^* and H^* are some constant.*

At each release date R_x we partition the ordered set of tiny jobs $T_x(\varpi)$ into subsets of roughly equal size $\approx \varepsilon^2 |I|_x / 2q^*$ (but less than $\varepsilon^2 |I|_x / q^*$). Then, we merge the jobs of each such subset into a new tiny job of profile φ.

Lemma 14. *With $1+O(\varepsilon)$ loss and in $O(n \log n)$ time, we can enforce $|T_x| \leq T^*$ and $|H_x| \leq H^*$ at each release date R_x, where T^* and H^* are some constant.*

Finally, by the ideas of Lemmas 9, 7 and the results of Lemma 14 we can prove the following:

Lemma 15. *With $1 + \varepsilon$ loss, each job completes within d^* intervals after its release, where d^* is some constant.*

Dynamic Programming. We partition the time line into a sequence of *blocks*, where each block i consists of d^* consecutive intervals. The basic idea is to use dynamic programming with blocks as units. The jobs of block i run either in block i or $i+1$. A *pseudo-schedule* S_i describes a possible placement of the jobs of blocks i. The dynamic programming entry $E(i, S_i)$ stores the minimum weighted completion time achievable by completing all jobs released before or in block i while leaving pseudo-schedule S_i for block $i+1$. Given all table entries for block $i-1$, the values for block i can be computed as follows.

$$E(i, S_i) = \min_{S_{i-1}} \{E(i-1, S_{i-1}) + W(i, S_{i-1}, S_i)\},$$

where $W(i, S_{i-1}, S_i)$ is the minimum weighted completion time achievable by scheduling the jobs in intervals of block i with respect to the *incoming* pseudo-schedule S_{i-1} and the *outgoing* pseudo-schedule S_i, respectively. Both the feasibility test and computation of $W(i, S_{i-1}, S_i)$ can be done in $O(1)$ time. Since there are at most $O(n)$ blocks the entire table $E(\cdot)$ can be computed in $O(n)$ time. The combination of all above steps and the dynamic programming algorithm gives a PTAS with $O(n \log n)$ running time. This completes the proof of Theorem 1.

References

1. F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Millis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings 40th IEEE Symposium on Foundations of Computer Science*, pages 32–43, 1999.
2. F. Afrati, E. Bampis, A. V. Fishkin, K. Jansen, and C. Kenyon. Scheduling to minimize the average completion time of dedicated tasks. In *Proceedings 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, LNCS 1974, Springer Verlag, pages 454–464, 2000.
3. F. Afrati and I. Milis. Designing PTASs for MIN-SUM scheduling problems. In *Proceedings 13th International Symposium on Fundamentals of Computation Theory*, LNCS 2138, pages 432–444. Springer Verlag, 2001.
4. S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. In *Proceedings 23rd International Colloquium on Automata, Languages and Programming*, LNCS 1099, pages 646–657. Springer Verlag, 1996.
5. C. Chekuri. *Approximation algorithms for scheduling problems.* PhD thesis, Department of Computer Science, Stanford University, 1998.
6. C. Chekuri and S. Khanna. A PTAS for minimizing weighted completion time on uniformly related machines. In *Proceedings 28th International Colloquium on Automata, Languages and Programming*, LNCS 2076, pages 848–861. Springer Verlag, 2001.
7. C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. In *Proceedings of 8th Annual ACM-SIAM Symposium on discrete Algorithms*, pages 609–618, 1997.
8. B. Chen, C. N. Potts, and G. J. Woeginger. *Handbook of combinatorial optimization (D.-Z. Du and P. M. Paradalos eds.)*, chapter A review of machine scheduling: complexity, algorithms and approximability, pages 21–169. Kluwer, 1998.
9. A. V. Fishkin, K. Jansen, and M. Mastrolilli. Grouping techniques for scheduling problems: Simpler and faster. In *Proceedings 9th Annual European Symposium*, LNCS 2161, pages 206–217, Arhus, 2001. Springer Verlag.
10. A. V. Fishkin, K. Jansen, and L. Porkolab. On minimizing average weighted completion time of multiprocessor tasks with release dates. In *Proceedings 28th International Colloquium on Automata, Languages and Programming*, LNCS 2076, pages 875–886, Crete, 2001. Springer Verlag.
11. M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* Freeman, San Francisco, CA, 1979.
12. M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operation Research*, 1:117–129, 1976.
13. L. A. Goldberg, M. Paterson, A. Srinivasan, and E. Sweedyk. Better approximation guarantees for job-shop scheduling. In *Proceedings 8th Symposium on Discrete Algorithms*, pages 599–608, 1997.
14. T. Gonzales and S. Sahni. Flowshop and jobshop schedules: Complexity and approximation. *Operations Research*, 26:36–52, 1978.
15. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic scheduling: A survey. *Annals of Discrete Mathematics*, 287–326, 1979.
16. L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average time: Off-line and on-line algorithm. *Mathematics of Operation Research*, pages 513–544, 1997.

17. H. Hoogeveen, P. Schuurman, and G. Weoginger. Non-approximability results for scheduling problems with minsum criteria. In *Proceedings 6th Conference on Integer Programming and Combinatorial Optimization*, LNCS 1412, pages 353–366. Springer Verlag, 1998.
18. K. Jansen, R. Solis-Oba, and M. Sviridenko. Makespan minimization in job shops: A polynomial time approximation scheme. In *Proceedings 31st Annual ACM Symposium on Theory of Computing*, pages 394–399, Atlanta, 1999. To appear in SIAM Journal on Discrete Mathematics.
19. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operation Research and Management Science*, chapter Sequencing and scheduling: Algorithms and complexity, pages 445–522. North-Holland, Amsterdam, 1993.
20. M. Queyranne and M. Sviridenko. New and improved algorithms for minsum shop scheduling. In *Proceedings 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 871–878, 2000.
21. A. S. Schulz. *Polytopes and scheduling.* PhD thesis, Technical University of Berlin, Germany, 1996.
22. P. Schuurman and G. J. Woeginger, Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2: 203–213, 1999.
23. P. Schuurman and G. J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2:203–213, 2000.
24. P. Schuurman and G. J. Woeginger. Approximation schemes – a tutorial. To appear in the book Lectures on Scheduling, edited by R. H. Moehring, C. N. Potts, A. S. Schulz, G. J. Woeginger and L. A. Wolsey., 2002.
25. D. B. Shmoys. Using linear programming in the design and analysis of approximation algorithms: Two illustrative examples. In *Proceedings 1st International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, LNCS 1444, pages 15–32. Springer Verlag, 1998.
26. D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal of Computing*, 23:617–632, 1994.
27. M. Skutella. *Approximation and randomization in scheduling.* PhD thesis, Technical University of Berlin, Germany, 1998.
28. W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistic Quarterly*, 3:59–66, 1956.
29. E. Torng and P. Uthaisombut. Lower bounds for SRPT-subsequence algorithms for non-preemptive scheduling. In *Proceedings 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 973–974, 1999.