Designing PTASs for MIN-SUM scheduling problems

F. Afrati

NTU Athens, Department of Electrical and Computer Engineering, Heroon Polytechniou 9, 15773, Athens, Greece

I. Milis

Athens University of Economics and Business, Department of Informatics, Patission 76, 10434 Athens, Greece

Abstract

We review approximability and inapproximability results for MIN-SUM scheduling problems and we focus on techniques for designing polynomial time approximation schemes for this class of problems. We present examples which illustrate the efficient use of the *ratio partitioning* and *time partitioning* techniques.

Key words: scheduling, MIN-SUM criteria, approximation algorithms, PTASs

1 Introduction

In the context of scheduling problems we are given, in general, a set of n jobs and a number m of machines. Each job, j, has a processing time $p_j^{(i)}$ on machine i, a release date r_j, a deadline d_j and a positive weight w_j. A job is usually processed on one machine at a time, but in the case of *multiprocessor jobs*, a job can (or need to be) processed simultaneously by several machines. A partial order, $\prec$, describes a *precedence relation* among jobs; we say that jobs are *independent* if $\prec$ is empty. Two different scheduling models can be considered: the preemptive one, where the interruption of a job execution is allowed, and the non-preemptive, where once a job starts executed it should be completed. A schedule Σ is an assignment of jobs to machines and an

⋆ This work has been partially supported by European Commission, Project APPOL-IST-1999-14084.

associated starting time S_j for each job j, such that an objective criterion is optimized. A schedule is feasible if at any time a machine processes at most one job and if $i \prec j$, then job j starts its execution after the completion of job i. By C_j we denote the time at which job j completes its execution and by D_i we denote the time by which machine i completes the execution of all jobs assigned to it. OPT denotes the objective value of an optimum schedule.

To refer to the several variants of scheduling problems, that can be obtained from the above general description, the standard three-field, $\alpha|\beta|\gamma$, notation scheme of Graham et al. [19] is used. The first field describes the machine environment. A single machine environment is denoted by $\alpha \equiv 1$, (identical) parallel machines by $\alpha \equiv P(m)$, uniformly related machines by $\alpha \equiv Q(m)$ and unrelated (parallel) machines by $\alpha \equiv R(m)$; the optional m denoting a constant number of machines. The second field of the notation describes any special conditions and constraints on the problem parameters (release dates, deadlines, precedence relation, processing times, multiprocessor jobs, preemption, e.t.c.). For multiprocessor jobs the notation fix_j is used for the variant where each job requires the simultaneous use of a prespecified set of machines and the notation $size_j$ for the variant where each job requires any set of machines of a given cardinality. In the general variant denoted by set_j, each job has a number of processing modes, each one defined by a set of machines and a processing time on this set of machines. The third field of the notation describes the optimization criterion. According to optimization criteria, scheduling problems can be classified into two broad classes: MIN-MAX criteria and MIN-SUM criteria. The most common MIN-MAX criterion is the completion time (makespan) of the schedule, $\min_\Sigma \ \max_j \ C_j$, and another is the maximum lateness, $\min_\Sigma \ \max_j \ (C_j - d_j)$. The most common MIN-SUM criterion is the total (weighted) completion time, $\min_\Sigma \ \sum_j \ w_j C_j$, and another is the total (weighted) flow time, $\min_\Sigma \ \sum_j \ w_j F_j$, where $F_j = C_j - r_j$.

It is well known that only some of the simplest variants of scheduling problems can be solved in polynomial time. For a thorough listing of polynomial solved variants as well as complexity results on scheduling problems the reader is referred to the book and a web page of Brucker [5]. Hence, most of the recent research is directed towards designing approximation algorithms for the NP-hard variants of scheduling problems. A *ρ-approximation algorithm* computes polynomially a solution within a factor ρ of the optimum one. In this setting we are interested in designing ρ-approximation algorithms with ρ as small as possible as well as on finding lower bounds for ρ. When ρ is a constant, then we say that the approximation algorithm is a *constant factor* one. We say, however, that we have a *Polynomial Time Approximation Scheme* (PTAS) if we give an algorithm which, for any fixed value of ϵ, can construct an $(1+\epsilon)$-approximation solution. If the time complexity of a PTAS is also polynomial in $1/\epsilon$, then it is called a *Fully Polynomial Time Approximation Scheme* (FPTAS). For negative approximation results, the notion of

NP-completeness is used to disprove the existence of good approximation algorithms, unless P=NP. It is well known that, there is no a FPATS for *strongly* NP-hard problems and that there is no a PTAS for MAX SNP-hard problems, unless P=NP.

Historically, MIN-MAX scheduling problems were the first investigated in the approximation algorithm framework and a lot of results have been proposed after Graham's seminal paper [18]. On the other hand, much less was known about the approximability of MIN-SUM scheduling problems until 1996. However, during the last five years, there was an explosive progress on the approximability of MIN-SUM problems, that recently led to PTASs for many of them.

In this paper we present a survey of this research on MIN-SUM scheduling focusing on PTASs design techniques. In Section 2 we review briefly the known constant approximation algorithms by focusing on the future usefulness of the proposed techniques as well as on their limitations. In Section 3 we present the recent results on PTASs and we classify the techniques used into two main categories: the *ratio partitioning* technique and the *time partitioning* technique. These techniques are presented in Sections 4 and 5, respectively, where we give examples which illustrate how each one of these techniques may be used to derive PTASs for different problem variants. Examples are taken from [3] and [1,2], respectively. We conclude with some open questions in Section 6.

2 Constant Approximation Algorithms

Although a FPTAS for $Pm \mid \mid \sum w_j C_j$ (due to Shani [34]) was known since 1976, no other approximation algorithm, for a MIN-SUM scheduling problem, has been proposed for some fifteen years. The next non-trivial approximation has been appeared in 1991 and it is due to Ravi, Agrawal and Klein [33] who gave an $O(\log n \log \sum_j w_j)$-approximation algorithm for $1|prec|\sum w_j C_j$. This result was slightly improved by Even, Naor, Rao and Schieber [14], in 1995, to an $O(\log n \log\log \sum_j w_j)$ factor. However, little was also known about the approximability of MIN-SUM scheduling problems even until 1995, despite the fact that there was an extensive literature on their polyhedral structure (especially of single machine problems). In fact, several linear programming formulations have been considered but they were only exploited either to characterize polynomial solvable special cases or to obtain strong lower bounds allowing the optimal solution of moderate sized problems using enumerative methods (see Queyranne and Schulz [32] for a thorough survey of this research area).

Philips, Stein and Wein [29], in 1995, gave the first constant approximation algorithms for a number of problems including $(8+\epsilon)$-approximation algorithms for both $1|r_j, pmtn|\sum w_jC_j$ and $P|r_j, pmtn|\sum w_jC_j)$, a $(16+\epsilon)$-approximation algorithm for $1|r_j|\sum w_jC_j$ and a $(24+\epsilon)$-approximation algorithm for $P|r_j|\sum w_jC_j$. Subsequently, Hall, Shmoys and Wein in [21] and its journal version by Hall, Schulz, Shmoys and Wein [20], improved dramatically the known approximation factors and gave algorithms for many new variants. Their results, some of which are the best known even today (see Table 1 below), were motivated by the successful work done on polyhedral structure of scheduling problems and build on earlier research on computing near-optimum solutions for MIN-MAX scheduling problems, by rounding optimum solutions to linear relaxations. In view of future results, the most important point of their work was the introduction of an *time-interval indexed* LP formulation, since it was proven later very fruitful for designing PTASs for many MIN-SUM problems.

The general idea of *time indexed* LP formulation was used in the past in several forms. Potts [30] used decision variables δ_{ij}, where δ_{ij} implies that job i precedes job j in the schedule. Dyer and Wosley [13] used variables x_{jt}, where $x_{jt}=1$ means that job j completes its execution at time t. Wosley [51] and Queyranne [31] used as decision variables the completion times C_j of jobs themselves. Hall et al. [21] adopted this last formulation to obtain their results for single machine problems and proposed a different, more compact linear *time-interval indexed* LP formulation: subdivide the time horizon into the intervals $[1,1], (1, 1+\epsilon], (1+\epsilon, (1+\epsilon)^2], \dots$, where ϵ is an arbitrary small positive constant. Then, the linear program only specifies the interval in which the job is completed and since all completion times within an interval are within a $(1+\epsilon)$ factor of each other, the relative schedule within an interval will be of little consequence. This was the first use of *time partitioning.*

After the results of Hall et al. [21,20] there has been an explosion of research in this area with new algorithms of successively smaller approximation factors as well as with algorithms for new variants of MIN-SUM scheduling problems. Most of these algorithms follow the general and successful idea of problem relaxation: first an optimum solution to a relaxation of the original problem is polynomially obtained and then this solution is used to obtain an approximate solution to this problem. Types of relaxations can be distinguished into two broad classes: preemptive scheduling relaxations and several linear and convex programming relaxations. The preemptive scheduling relaxations consist of solving polynomially the preemptive variant of a non-preemptive problem, while the linear and convex programming relaxations consist of relaxing the integer constraints of an integer programming formulation of the problem. The solution of the relaxed problem is then used either to order the jobs in time (in one machine problems) or to assign the jobs to machines (in many machines problems). A series of ideas have been used for such a construction of a solution to the original problem including scheduling by completion times, scheduling by α-points as well as deterministic and randomized rounding methods. As this paper focuses on PTASs design, we summarize, in Table 1, only the best known constant approximation algorithms. Clearly we omit a large body of important work on constant approximation algorithms, but the reader can acquire an accurate view of the results in this area by considering the references given in Table 1. The reader is also referred to [41] for a elegant survey on several linear programming relaxation algorithms mainly for $1|prec|\sum w_jC_j$.

Although there is practical evidence [35,48] that linear programming relaxations can provide very good approximate solutions, all algorithms based on such relaxations suffer from a major drawback: the solution to a problem is compared to the solution of the relaxed one and, unfortunately, there is an integrality gap between these two solutions. Thus, comparing any solution of such an algorithm to a fractional one, inherits this gap. Hall et al. [20], Schulz and Skutella [36] and Chekuri and Motwani [10] argue on the fact that several relaxation methods for $1|prec|\sum w_jC_j$ can not improve on the best known 2-approximation algorithm. Moreover, Torng and Uthaisombut [46] have shown that any preemptive relaxation for $1|r_j|\sum C_j$, that uses the shortest remaining processing time (SRPT) rule to solve the relaxed problem, can not led to an approximation factor better than $e/(e-1)$, that matches the best known factor of 1.58 [11]. It seems therefore, that the relaxation approaches can not yield a PTAS for MIN-SUM scheduling problems.

3 Polynomial Time Approximation Schemes

Surprisingly, only FPTASs for *weakly* NP-hard MIN-SUM problems were known until 1988, including the Shani's FPATS for $Pm\,|\,|\sum w_jC_j$ [34] and a FPATS for $Qm|\,|\sum w_jC_j$ by Woeginger [49]. They were based on dynamic programming.

Alon, Azar, Woeginger and Yadid [4], in 1998, obtained a PTAS for $P\,|\,|\sum D_i^2$. Skutella and Woeginger [44] realized that this last result implies also a PTAS for $P\,|\frac{w_j}{p_j}=R|\sum w_jC_j$ and then in a first step they generalized this to a PTAS for $\frac{w_j}{p_j}$ ratios within a constant range. In a second step they obtained the first PTAS for a *strongly* NP-hard MIN-SUM problem ($P\,|\,|\sum w_jC_j$). Their main idea was *ratio partitioning*, i.e. the partitioning of the jobs into subsets such that the ratios $\frac{w_j}{p_j}$ of all the jobs in each subset are within a constant range. Then, using the first step, near optimum solutions are computed for all subsets. Finally, these schedules are concatenated in order of nonincreasing ratios in each machine.

Subsequently, Afrati, Bampis, Kenyon and Milis [3] used also ratio partitioning to develop a PTAS for $Rm\,|\,|\sum w_jC_j$. However, the concatenation of schedules that is used in ratio partitioning technique can not be applied in the presence of release dates. Thus the *time partitioning* technique was recalled by several researchers and it was proven powerful enough to yield PTASs for a variety of MIN-SUM scheduling problems in the presence of release dates. Several groups obtained, independently, PTASs for several variants of the average weighted completion time problem. All these results merged in to single article by eleven authors [1], where PTASs were obtained for scheduling jobs with release dates on a single machine, on identical parallel machines and on (a constant number) of unrelated machines in both preemptive and non-preemptive models. These results were a substantial progress in resolving the approximability question of MIN-SUM problems and it seems that the ideas proposed will influence the design of PTASs for other variants.

Indeed, based also on time partitioning technique and building on the ideas proposed in [1], Afrati et al. presented PTASs for $P2|fix_j|\sum C_j$ and $Pm|fix_j, pmtn|\sum C_j$ in [2], and very recently Fishkin, Jansen and Porkolab presented PTASs for $Pm|fix_j, r_j|\sum w_jC_j$ and $Pm|size_j, r_j|\sum w_jC_j$ in [15] and $Pm|set_j|\sum w_jC_j$ in [16]. Moreover, Chekuri and Khanna using some non-trivial extensions to the ideas introduced in [1], obtained PTASs for $Q|r_j|\sum w_jC_j$ and $Q|r_j, pmtn|\sum w_jC_j$ [8].

However, it seems that it is difficult to extend the techniques for approximating the average weighted completion time in the case of average weighted flow time (recall that the flow time F_j for each job is defined as $F_j = C_j - r_j$). When computing optimal solutions, minimizing the average weighted completion time is equivalent to minimizing the average weighted flow time, since $\sum r_j$ is determined by the input. However, these two objective functions behave quite differently when we look for approximate solutions. Recently, Chekuri, Khanna and Zhu [9] proposed initially a quasi-polynomial $2+\epsilon$-approximation algorithm for $1|r_j, pmtn|\sum w_jF_j$, in the special case where the ratio of the largest job processing time over the smallest one, $P = \frac{\max_j p_j}{\min_j p_j}$, is polynomially bounded in n. After this result, Chekuri and Khanna [7] gave a quasi-PTAS for the same problem, in the case where either P is polynomially bounded or the ratio of the largest job weight over the smallest one, $W = \frac{\max_j w_j}{\min_j w_j}$, is polynomially bounded. They also proved that this last quasi-PTAS becomes a PTAS in the cases where (i) both P and W are bounded, (ii) P is bounded and W is unrestricted and (iii) for each job its weight is the inverse of its processing time, i.e. $p_j = 1/w_j$ (also known as *stretch metric*).

In the third column of Table 1 we give a summary of existing (F)PTASs. We present there either the existence of a (F)PTAS or known inapproximability results in terms of either MAX SNP-hardness or existing lower bounds on the variant's approximation factor.

Problem	Best known ρ	(F)PTAS or Lower bound
$1\|r_j\|\sum C_j$	1.58 [11]	PTAS [1]
$1\|r_j\|\sum w_jC_j$	1.69 [17]	PTAS [1]
$1\|r_j, pmtn\|\sum w_jC_j$	4/3 [38]	PTAS [1]
$1\|prec\|\sum w_jC_j$	2 [10,12,20,27]	
$1\|interval\ order\|\sum w_jC_j$	$\frac{1+\sqrt{5}}{2}+\epsilon$ [50]	
$1\|convex\ bipartite\ order\|\sum w_jC_j$	$\frac{1+\sqrt{5}}{2}+\epsilon$ [50]	
$1\|r_j, prec\|\sum w_jC_j$	$e+\epsilon$ [36]	
$1\|r_j, prec, pmtn\|\sum w_jC_j$	2 [20]	
$1\|r_j\|\sum F_j$	$O(n^{1/2})$ [24]	$\Omega(n^{1/2-\epsilon})$ [24]
$1\|r_j, pmtn\|\sum w_jF_j$	$2+\epsilon$ [9] *	quasi-PTAS [7] **
$Pm\|\ \|\sum w_jC_j$		FPTAS [34]
$P\|\ \|\sum D_i^2$	1.04 [26]	PTAS [4]
$P\|prec, p_j=1\|\sum C_j$		8/7 [22]
$P\|prec\|\sum C_j$		4/3 [22]
$P\|\ \|\sum w_jC_j$	1.21 [23]	PTAS [44]
$P\|r_j\|\sum w_jC_j$	2 [37]	PTAS [1]
$P\|r_j, pmtn\|\sum w_jC_j$	2 [37]	PTAS [1]
$P\|r_j, prec\|\sum w_jC_j$	4 [28]	
$P\|r_j, prec, pmtn\|\sum w_jC_j$	3 [20]	
$P\|r_j\|\sum F_j$	$O(\sqrt{\frac{n}{m}}\log\frac{n}{m})$ [25]	$\Omega(n^{1/3-\epsilon})$ [25]
$P\|r_j, pmtn\|\sum F_j$	$O(\log\min\{\frac{n}{m}, \log P\})$ [25]	
$P2\|fix_j\|\sum C_j$	2 [6]	PTAS [2]
$Pm\|fix_j, pmtn\|\sum C_j$		PTAS [2]
$Pm\|fix_j, r_j\|\sum w_jC_j$		PTAS [15]
$P\|fix_j, p_j=1\|\sum C_j$		$\Omega(m^{1/2-\epsilon})$ [15]
$Pm\|size_j, r_j\|\sum w_jC_j$		PTAS [15]
$P\|size_j\|\sum C_j$	32 [47]	
$Pm\|set_j\|\sum w_jC_j$		PTAS [16]
$Qm\|\ \|\sum w_jC_j$		FPATS [49]
$Q\|r_j\|\sum w_jC_j$		PTAS [8]
$Q\|r_j, pmtn\|\sum w_jC_j$		PTAS [8]
$Rm\|\ \|\sum w_jC_j$	3/2 [42]	PTAS [3,1]
$Rm\|r_j\|\sum w_jC_j$	2 [43]	PTAS [1]
$Rm\|r_j, pmtn\|\sum w_jC_j$	3 [43]	PTAS [1]
$R\|\ \|\sum w_jC_j$, $R\|\ \|\sum C_j$	3/2 [40,42]	Max SNP-hard [22]
$R\|r_j\|\sum w_jC_j$, $R\|r_j\|\sum C_j$	2 [43]	Max SNP-hard [22]
$R\|pmtn\|\sum w_jC_j$	2 [43]	
$R\|r_j, pmtn\|\sum w_jC_j$	3 [43]	

* Quasi-polynomial algorithm for poly-bounded P

** For poly-bounded P or W

Table 1. Overview of results.

It is interesting that all existing PTASs rely either on *time partitioning* or on *ratio partitioning* technique. These techniques succeeded where older techniques had failed and it is most likely that they can be used to obtain PTASs for more involved scheduling variants. Moreover, all known PTASs use a combination of elementary and well understood combinatorial ingredients: partitioning, grouping, geometric rounding, enumeration and dynamic programming. The approach taken, in these PTASs, is to sequence several transformations of the input problem. Some transformations are actual changes to simplify the input, while others are applied as thought experiments to the optimum solution to prove there is a near-optimum solution with nice structure. Each transformation potentially increases the objective value by only $1+O(\epsilon)$, that is, it produces $1+O(\epsilon)$ *loss*. Besides problem-variant-specific transformations, some general transformations are used in these techniques. Such transformations are *geometric rounding* and *time stretching*.

3.1 Geometric Rounding and Time Stretching

The first simplification creates a well-structured set of possible processing times and release dates: With $1+\epsilon$ loss, we can assume that all processing times and release dates are rounded to integer powers of $1+\epsilon$. To prove, we change the values in two steps. First multiply every release date and processing time by $1+\epsilon$; this increases the objective by $1+\epsilon$. Then *decrease* each date and time to the next *lower* integer power of $1+\epsilon$ (which is still greater than the original value). This can only improve things.

The second transformation *time stretching* creates idle time *gaps* evenly spread along the schedule. These gaps will be used conveniently in later transformations to accommodate jobs that we are moving around in order to prove that there is a near-optimum schedule with nice structure. Often, we will use time stretching to add an idle amount of ϵp_j time units before every job j. This will increase each completion time (and hence their sum) by a factor of $1+\epsilon$. According to this last transformation, we may also assume that the starting time of each job j is $S_j \geq \epsilon p_j$: If job j completed at time $t > p_j$ then it now completes at time $(1+\epsilon)t$ and therefore does not start until time $\epsilon t \geq \epsilon p_j$.

4 Ratio Partitioning

For the basic $1||\sum w_j C_j$ variant, W. Smith, in 1956, designed a very easy greedy algorithm: sequencing in order of non-increasing w_j/p_j ratios produces an optimum schedule [45]. To prove the correctness of this algorithm, observe that any schedule either satisfies Smith's rule or there are two consecutively scheduled jobs j_1, j_2 such that j_1 is scheduled before j_2 and $w_{j_1}/p_{j_1} < w_{j_2}/p_{j_2}$. Hence, if we swap jobs j_1 and j_2, the cost of the new schedule minus the cost of the old schedule is $w_{j_1}p_{j_2} - w_{j_2}p_{j_1}$ which is a negative number.

For simplicity of notation we think in terms of two machines; however, it should be clear that the analysis can be generalized for any constant number of machines. By $p_j^{(i)}$ we denote the execution time of job j on machine i. If we consider the restriction of the optimum schedule to each one of the machines, then Smith's ratio rule applies: the jobs executed on machine M_i are processed in order of non-increasing ratios $R_j^{(i)} = w_j/p_j^{(i)}$. It is therefore clear that, in the absence of release dates or any other constraint, the only difficulty is to decide, for each job, the machine which will execute it. To this end, we define a *ratio partitioning*, i.e. a partition of the range of ratios into disjoint windows such that the i^{th} window consists of ratios in the range $F_i = (a^i R_{max}, a^{i-1} R_{max}]$, where $R_{max} = \max_{i,j}\{w_j/p_j^{(i)}\}$ and $a = \epsilon^4$. Towards a further problem simplification we define a **job-type** $T_{R,R'}$ as the set of jobs having the same pair of ratios, i.e. $T_{R,R'} = \{j \mid R_j^{(1)} = R \text{ and } R_j^{(2)} = R'\}$.

Before moving on with the formal statement and proof of results we summarize the intuition in the following three informal statements which will be used to develop and analyze a PTAS

(1) We can think of each job as having its two ratios either in the same window or in adjacent windows.
(2) "Very different" job types do not affect each other.
(3) We can bound the number of jobs within each job-type by a constant number.

Now, we state and prove formally items 1,2,3.

1. We prove that for each job, either we can tell in advance in which machine it will be scheduled (we say that it is decided) or else its two ratios do not differ much. If the processing times of a job are very different on the two machines, then we can directly decide to schedule it on the machine on which it has the shortest processing time. In quantitative terms: We can assume that all jobs such that $p_j^{(1)} < \epsilon p_j^{(2)}$ are scheduled on M_1 (we say that this job is M_1-*decided*), and that all jobs such that $p_j^{(2)} < \epsilon p_j^{(1)}$ are scheduled on M_2 (we say that this job is M_2-*decided*). To prove, we show that by moving all such jobs to the machine with the small processing time, the cost does not increase much: we move each such job j in the other machine in the place so that it starts just before the earliest job which completes at time $> C_j$ (the completion time of the moved job in the original schedule). This affects the total cost by increasing the completion time of job j and by delaying the jobs after j in the machine M_1 (suppose we moved the job to M_1) However, the new completion time of job j is at most $C_j + p_j^{(1)} < C_j + \epsilon p_j^{(2)} < C_j(1+\epsilon)$. The other jobs affected are the jobs in machine M_1 which are delayed. Each such job k is delayed by at most ϵ times the sum of M_2-processing times of moved jobs which completed before time C_k in M_2; but this is at most ϵ times C_k. Thus, in this case, we can put $p_j^{(2)} = \infty$ or $p_j^{(1)} = \infty$ accordingly. Moreover, for each undecided job we now know that $p_j^{(1)} > \epsilon p_j^{(2)}$ and $p_j^{(2)} > \epsilon p_j^{(1)}$, that is $\epsilon < R_j^{(1)}/R_j^{(2)} < 1/\epsilon$. Hence, each job is either decided or it has its two ratios either in the same window or in adjacent windows.

2. "Very different" job types do not affect each other. To state formally, we rewrite the cost function as:

$$\sum_{j,k \text{ on } M_1} \mathcal{M}_1[j,k] + \sum_{j,k \text{ on } M_2} \mathcal{M}_2[j,k],$$

where

$$\forall j,k \in J, \qquad \mathcal{M}_i[j,k] = \begin{cases} p_k^{(i)} w_j & \text{if } j = k \\ \frac{1}{2} p_k^{(i)} w_j & \text{if } j \neq k \text{ and } \frac{w_k}{p_k^{(i)}} = \frac{w_j}{p_j^{(i)}} \\ p_k^{(i)} w_j & \text{if } \frac{w_k}{p_k^{(i)}} > \frac{w_j}{p_j^{(i)}} \\ 0 & \text{otherwise} \end{cases}$$

Notice that an entry $\mathcal{M}_i[j,k]$ represents the contribution to the cost function of job j, due to job k. Thus, if job j is scheduled on machine M_1, and the jobs on M_1 with ratio equal to $R_j^{(1)}$ are ordered randomly, then the expected weighted completion time of j satisfies: $w_j C_j = \sum_{k \text{ on } M_1} \mathcal{M}_1[j,k]$.

Using this formulation of the cost function, we can prove that if the ratios of two jobs are very different, then we can neglect their interaction:

1. For every pair of jobs $\{j,k\}$ such that $R_j^{(i)} < \epsilon^2 R_k^{(i)}$, we can replace $\mathcal{M}_i[j,k]$ by 0.

2. For $i = 1,2$, and for every pair of jobs $\{j,k\}$ such that j and k are either M_i-decided or undecided, and such that $R_j^{(i)} < \epsilon^4 R_k^{(i)}$, we can replace both $\mathcal{M}_1[j,k]$ and $\mathcal{M}_2[j,k]$ by 0.

Proof of 1. We prove this in two steps: First for each k on M_1, we look at the jobs j on M_1 such that $R_j^{(1)} < \epsilon^2 R_k^{(1)}$, and whose completion times satisfy $C_k < \epsilon C_j$. The presence of k before such a j has only a marginal effect on j's completion time (easy to prove). Second, we look at the pairs $\{j,k\}$ which are scheduled on M_1, such that $R_j^{(1)} < \epsilon^2 R_k^{(1)}$, but whose completion times satisfy $C_j \leq C_k/\epsilon$. The contribution to the cost function of those pairs is very little as compared to $w_k C_k$:

$$\sum_{\substack{j \ on \ M_1 \\ C_j \le C_k/\epsilon}} w_j p_k^{(1)} = \sum_{\substack{j \ on \ M_1 \\ C_j \le C_k/\epsilon}} r_j^{(1)} p_j^{(1)} p_k^{(1)} < \epsilon^2 \sum_{\substack{j \ on \ M_1 \\ C_j \le C_k/\epsilon}} w_k p_j^{(1)}$$

$$= \epsilon^2 w_k \sum_{\substack{j \ on \ M_1 \\ C_j \le C_k/\epsilon}} p_j^{(1)} \le \epsilon^2 w_k \max_{\substack{j \ on \ M_1 \\ C_j \le C_k/\epsilon}} C_j \le \epsilon^2 w_k C_k/\epsilon = \epsilon w_k C_k,$$

Proof of 2. Consider the case $i = 1$. Because $R_j^{(1)} < \epsilon^2 R_k^{(1)}$, we can replace $\mathcal{M}_1[j,k]$ by 0–according to (1) above which we just proved. If either job is M_1-decided, then it will never be placed on M_2 and so we can replace $\mathcal{M}_2[j,k]$ by 0. If both are undecided, then we have: $p_j^{(2)} > \epsilon p_j^{(1)}$ and $p_k^{(2)} < p_k^{(1)}/\epsilon$. Thus: $\frac{w_j}{p_j^{(2)}} < \frac{1}{\epsilon}\frac{w_j}{p_j^{(1)}} < \frac{\epsilon^4}{\epsilon}\frac{w_k}{p_k^{(1)}} < \frac{\epsilon^4}{\epsilon}\frac{1}{\epsilon}\frac{w_k}{p_k^{(2)}} = \epsilon^2 \frac{w_k}{p_k^{(2)}}$. Thus the condition in (1) is satisfied, hence, we can replace $\mathcal{M}_2[j,k]$ by 0.

3. We can bound the number of jobs within each job-type by a constant number. Consider a job-type $T_{R,R'}$ and the sum of the weights, $W_{R,R'}$, of jobs in $T_{R,R'}$. We group small jobs together to build larger compound jobs. Thus, we can assume that every job of a job-type has weight at least $\epsilon^2 W_{R,R'}$, and hence each job-type has at most $\frac{1}{\epsilon^2} = O(1)$ elements. More specifically, we group greedily so that the sum of weights of the tiny jobs in each group (which now represents the weight of the compound job) is between $\epsilon^2 W_{R,R'}$ and $2\epsilon^2 W_{R,R'}$. There are two points to discuss: a) The cost does not change much by this grouping. For each group (in machine i with ratio R say), the contribution to the cost function before the grouping was $\sum_{m,n \ jobs \ in \ group_k} R p_m^{(i)} p_n^{(i)}$. This quantity, though, is bounded on both sides by $R(\sum_{n \ job \ in \ group_k} p_n^{(i)})^2$ and $R\frac{1}{2}(\sum_{n \ job \ in \ group_k} p_n^{(i)})^2$ (upper and lower bound respectively). Hence it is at most $\frac{4}{R}\epsilon^4 W_{R,R'}^2$. Similarly the jobs in this job type contribute to the total cost by a quantity which is at least $\frac{1}{2R} W_{R,R'}^2$. There are at most $\frac{1}{\epsilon^2}$ compound jobs, hence the total shift from the original cost is at most $\frac{4}{\epsilon^2 R}\epsilon^4 W_{R,R'}^2$ which is less than $\epsilon \frac{1}{2R} W_{R,R'}^2$. b) Clearly a schedule that uses the compound jobs results in a cruder fractioning of the jobs in the job type among the machines. To take care of that we stretch time by $1+\epsilon$ and this stretching is enough to accommodate the compound jobs.

4.1 Example: $R2||\sum w_j C_j$

We first simplify the input of the problem by applying geometric rounding on the parameters defining the jobs. Then, we identify the jobs which are M_1-decided or M_2-decided. We replace all irrelevant processing times (i.e. $p_j^{(2)}$ if j is M_1-decided and $p_j^{(1)}$ if j is M_2-decided) by ∞, which means that the corresponding ratio becomes equal to 0. Then we do all the simplifications of the previous subsection, namely calculating first the $\mathcal{M}_i[j,k]$ and then putting accordingly some of them equal to 0.

Note that, the set of undecided jobs having at least one ratio in a specific window has size $O(\log^2(1/\epsilon)/\epsilon^2)$: Roughly, because of geometric rounding, we have only a constant number of different ratios within a window and, because undecided jobs have their ratios either in the same window or in adjacent windows, there are only a constant number of job types that may contain a job of a particular ratio in one of the machines. Moreover, there is a constant number of jobs within each job type.

Summing up the above analysis: a) Each job type has the two ratios either in the same window or in adjacent windows and b) Job types affect each other only in the stretch of two windows and c) there is a constant number of undecided jobs with at least one ratio in a specific window. An immediate consequence of that is dynamic programming with respect to the windows of ratios. In each stage i of the dynamic programming, we have fixed the schedule for jobs with both ratios in windows $F_1, \ldots F_i$ and keep all possible schedules of jobs with at least one ratio in window F_i. Thus, each stage runs in $O(1)$ time.

5 Time Partitioning

For an arbitrary integer x, we define $R_x = (1+\epsilon)^x$. We partition the time interval $(0,\infty)$ into disjoint intervals $I_x = [R_x, R_{x+1})$. We will use I_x to refer to both the interval and the size $(R_{x+1} - R_x)$ of the interval. We will often use the fact that $I_x = \epsilon R_x$, i.e., the length of an interval is ϵ times its start time. We consider an optimum schedule and focus on the schedule of one machine. These are easy to prove but useful facts:

- Moving a job j inside the interval in which it runs produces a $1+O(\epsilon)$ loss because it will only increase its completion time by at most I_x which is less than ϵR_{x+1} or less than ϵC_j.
- A job can not be arbitrarily large: In the worst case, a job starts at time ϵp_j and completes at time $\epsilon p_j + p_j$, hence, it crosses at most $s = \lceil \log_{1+\epsilon} \frac{\epsilon p_j + p_j}{\epsilon p_j} \rceil = \lceil \log_{1+\epsilon}(1+\frac{1}{\epsilon}) \rceil$ intervals.
- If a job j crosses several intervals, we know that it crosses at most s intervals, and then we can accumulate idle time for all s intervals in the beginning of job j, and we know that the relative size of the 1st interval to the size of the sth interval is $\geq \frac{I_x}{I_{x+s}} = \frac{1}{(1+\epsilon)^{s-1}} = \epsilon$.
- By time stretching we can create, in each interval an extra idle gap of size ϵI_x.

In what follows in this section, we assume unit weights.

5.1 Large and Small Jobs

In many cases, jobs that are much smaller than the interval in which they run are essentially negligible and easy to deal with. The difficulty comes from jobs that are *large*—taking up a substantial portion of the interval. *Small* jobs are a lot like fractional jobs in linear relaxation methods, and fractional solutions are usually easier to find. We assume, here, that small jobs are defined to be those with $p_j < \epsilon I_x$, where I_x is the interval where job j runs.

- We may restrict attention to schedules in which no small job crosses an interval: By time stretching, add in each interval a gap of ϵI_x and finish the single small job that may be crossing an interval.
- There is a constant number of large jobs in an interval, namely at most $\frac{1}{\epsilon}$.

Interestingly, we can almost get rid altogether of large jobs (except a constant number of them). We can do that by moving most large jobs in larger intervals without much degrading the quality of the schedule. Thus, we end up with an overall constant number of large jobs:

Lemma 1 *With a $1 + O(\epsilon)$ loss, we assume:*

1. For each job j, $S_j \geq \min\{\frac{p_j}{\epsilon^2}, \epsilon^7 OPT\}$.

2. All jobs are small in the intervals they run except at most $1/\epsilon^7$.

Proof: The proof of (2) follows from (1): Note that $S_j \geq p_j/\epsilon^2$ means that job j is small when it runs. Also, all jobs starting after time $\epsilon^7 OPT$ are at most $1/\epsilon^7$. The small jobs trivially satisfy this condition in (1). We move around large jobs in larger intervals so that they are now small in the interval that they run. We do that only for large jobs that complete before $\epsilon^7 OPT$.

We move a job that completes before time $(1 + \frac{1}{\epsilon^2})p_j$ in the first gap after time $t = C_j/\epsilon^4$. We have to worry about three considerations:

1. There is space large enough to accommodate all jobs that are landed on a particular gap. This is so because the gap at time t is equal to $\epsilon^2 t$, hence, at least equal to C_j, thus can hold all jobs which complete before time C_j.

2. There is not some hugely long crossing job in this interval. This is guaranteed by the fact that any crossing job may cross at most s intervals. Thus, we put the moved job in the beginning of the crossing job and still it is in an interval where it is considered small. This is so because the intervals along which a job

stretches differ in size by a factor at most $1/\epsilon$.

3. The added cost is not too much: We need to bound the added cost of the large jobs we moved forward. The last interval from which we advance jobs ends at time $t = \epsilon^7 OPT$. Thus the total completion time of the advanced jobs is $\sum_{R_x<t} \frac{1}{\epsilon}\frac{(1+\epsilon)R_x}{\epsilon^4} \leq \frac{t}{\epsilon^5}\sum_{i\geq 0} 1/(1+\epsilon)^i = \frac{t}{\epsilon^5}\frac{(1+\epsilon)^2}{\epsilon} = \epsilon \cdot \mathrm{OPT}(1+\epsilon)^2 \leq 2\epsilon\mathrm{OPT}$ as desired. ∎

5.2 *Example:* $1|\, r_j \,|\sum C_j$

We apply geometrical rounding to all variables of the input of the problem. Thus, jobs are released only in the end of intervals. The preemptive variant of $1|\, r_j \,|\sum C_j$ can be solved optimally by SPRT rule. Supposing that, in the optimum schedule of the non preemptive problem all jobs are small, we can set release dates equal to $\max\{r_j, \frac{p_j}{\epsilon^2}\}$. According to Lemma 1, this does not change the feasibility of a nearly optimum schedule. We can now use SPRT rule to obtain an optimum preemptive schedule and then use the gaps in each interval to finish up the preempted jobs. These are small, so they need no more than ϵI_x to finish up. Running SPRT followed by finishing up preempted jobs corresponds to running SPT rule.

Our only problem, now, is that the optimum schedule may require some jobs to run when they are large. By Lemma 1, there is still a nearly optimum solution with only a constant number of large jobs. We can schedule these large jobs by complete enumeration and schedule the rest using SPT rule with the new release dates as above. Again, using SPT rule for the small jobs, essentially amounts to using SPRT rule (which still remains optimum in the presence of the fixed large jobs) to obtain a preemptive solution and then finishing up the jobs.

5.3 *Example:* $P2|fix_j|\sum C_j$

The jobs are dedicated: each job j requires for its execution the simultaneous availability of a prespecified subset of machines $\tau_j = \{M_1, \ldots, M_m\}$ for p_j time units. The set τ_j is called the *type* of job j. We restrict attention here to the case with two machines. The methods and techniques though extend easily to the case of any fixed number of machines. For this problem, we have two kinds of monoprocessor jobs, the 1-jobs and the 2-jobs dedicated to run on machine M_1 and M_2 respectively. Biprocessor 12-jobs are dedicated to both machines.

In what follows, we actually describe a reduction from the preemptive case to the non-preemptive case. This goes through for any fixed number of machines, only that for $M > 2$, the preemptive case is not polynomial and is solved by a PTAS.

Besides geometric rounding, we need some more transformations. We begin with the following lemma which states that jobs scheduled to run in parallel have comparable processing times.

We first state a few useful facts:

- If p_{max} is the maximum processing time of a set of jobs and ℓ and C is the length and the cost respectively of a schedule of this set of jobs, then, $\ell < 2\sqrt{p_{max}C}$. At least $\frac{\ell}{2p_{max}}$ jobs will be executed after time $\frac{\ell}{2}$, hence the cost is at least: $C \geq \frac{\ell}{2p_{max}}\frac{\ell}{2}$, hence the inequality.
- Suppose each processing time is $< \epsilon^{10}$, then the makespan is $\ell < 2\sqrt{p_{max}C} < 2\sqrt{\epsilon^{10}C^2}$ hence $\ell < 2\epsilon^5 C$.

Lemma 2 *With a $1 + O(\epsilon)$ loss, we never have two jobs scheduled in parallel whose processing times differ by a factor greater than $\frac{1}{\epsilon^5}$.*

Proof: Let j be a job executed w.l.o.g. on machine M_1, and let A be the set of jobs scheduled in parallel with j on machine M_2 with processing times $< \frac{1}{\epsilon^5}p_j$. Recall that, since the single-machine-jobs on each machine are scheduled in the order of increasing processing times, the jobs in A form an interval.

The intuition of this proof is that if jobs in A stretch in a small time interval ($< \epsilon p_j$), then by time stretching we move the large job by at most ϵp_j and the additional cost is small compared to the processing time p_j of job j. If the jobs in A stretch in a larger time interval, then, the number of jobs in A is comparitively large, hence we move job j in a gap of a larger interval and the additional cost is small compared to the sum of completion times of jobs in A. The technical details are now given. We consider the following two cases:

a) $\sum_{i \in A} p_i < \epsilon p_j$. We use time stretching to move starting time of j to $S_j + \epsilon p_j$, thus jobs in A and job j do not run in parallel any more.

b)$\sum_{i \in A} p_i \geq \epsilon p_j$. Put the job j in the *gap* so that it completes at time $\frac{C_j}{\epsilon^2}$. Let S_j be the starting time of job j in the initial schedule. Since for every $i \in A$, $p_i < \epsilon^5 p_j$, we get that at least $\frac{1}{2\epsilon^4}$ jobs in A complete in the time-interval $[S_j + \frac{\epsilon p_j}{2}, S_j + \epsilon p_j]$. Then $\sum_{i \in A} C_i > (S_j + \frac{\epsilon p_j}{2})\frac{1}{2\epsilon^4} > \frac{\epsilon}{2}\frac{1}{2\epsilon^4}(S_j + p_j) = \frac{1}{4\epsilon^3}C_j$. Thus, $\frac{C_j}{\epsilon^2} < 4\epsilon \sum_{i \in A} C_i$. Summing over all jobs j, we get an increase of at most $4\epsilon OPT$. ∎

The following lemma states that if there is a *gap* in the values of the processing times of the jobs in the sense we can partition the set of jobs into "long" and "short" jobs then al short jobs are scheduled before any long job is scheduled.

Lemma 3 *If the set of jobs $\mathcal{T}$ can be partitioned into subsets L and S such that $\max_{j \in S} p_j < \epsilon^5 \min_{j \in L} p_j$. Then, with $1+\epsilon$ loss, in any optimal schedule, all jobs in S are scheduled before any job in L is scheduled.*

Proof: Observe that, because of lemma 2 above, we can view a near-optimal schedule as a series of time inervals, each time interval accomodating either only jobs in L or only jobs in S. Suppose there are two consecutive time intervals so that a set of L jobs L' is scheduled before a set of S jobs S'. If we swap the schedules in those time intervals so that S' is scheduled before L', then we get a better schedule. We compute the change in cost. The jobs of L' are delayed by $\mathcal{S}$ while the jobs of S' are moved up by $\mathcal{L}$ ($\mathcal{S}$ and $\mathcal{L}$ are the length of the time interval in which the jobs in S' and L' respectively are scheduled) Thus the change of the cost is ($|L'|$, $|S'|$ is the number of jobs in S' and L' respectively).

$$\mathcal{S}|L'| - \mathcal{L}|S'| \leq (\frac{2}{\min_{j \in L} p_j} - \frac{1}{\max_{j \in S} p_j})\mathcal{L}\mathcal{S}$$

$$\leq (2 - \frac{\min_{j \in L} p_j}{\max_{j \in S} p_j})\frac{\mathcal{L}\mathcal{S}}{\min_{j \in L} p_j}.$$

By the hypothesis we have that $\frac{\min_{j \in L} p_j}{\max_{j \in S} p_j} > \frac{1}{\epsilon^5}$, hence the upper bound on the change of cost is a negative number for $\epsilon < 1/2$. ∎

However this lemma cannot be used unless there is such a gap. It turns out that there is an "almost gap". Namely, a separation result allows for partitioning the jobs into "long" jobs and "short" jobs with a few (negligible) "medium" jobs in between. UB denotes the upper bound to the optimal cost obtained by processing all jobs sequentially. Note that this upper bound is within a factor of 2 of the optimal.

Definition: *Let L_i denote the set of jobs with processing time $p_j \geq \epsilon^{5i+5} UB$, M_i denote the set of jobs j with processing time $\epsilon^{5i+10} UB \leq p_j < \epsilon^{5i+5} UB$, and S_i denote the remaining jobs.*

Lemma 4 *If $L_1 \neq \emptyset$, then there is some $i < \log_{1+\epsilon} \frac{1}{\epsilon^{10}}$ such that $OPT(M_i \cup L_i) \leq (1+\epsilon^2) OPT(L_i)$. We call this i the* seperation index.

Proof: The proof is a simple algebraic manipulation. Observe that $L_i = L_1 \cup M_1 \cup M_2 \cup \ldots \cup M_{i-1}$. Taking the inequality $OPT(L_1) > \epsilon^{10} OPT$ and, i inequalities, one for each i: $OPT(M_i \cup L_i) > (1+\epsilon^2) OPT(L_i)$ or equivalently $OPT(L_1 \cup M_i \cup M_1 \cup M_2 \cup \ldots \cup M_{i-1}) > (1+\epsilon^2) OPT(L_1 \cup M_1 \cup M_2 \cup \ldots \cup M_{i-1})$

and multiplying the rhs parts and the lhs parts yields the inequality: $OPT > OPT(L_1 \cup M_1 \cup \ldots \cup M_i) > (1+\epsilon^2)^i \epsilon^{10} OPT$, i.e. $(1+\epsilon^2)^i \epsilon^{10} < 1$. Consequently, $i < \log_{1+\epsilon^2} \frac{1}{\epsilon^{10}}$. ∎

Combining the results in lemmas 3 and 4, we get the following:

Lemma 5 *Let i be the separation index. Then, with a $1+\epsilon$ loss, we schedule all jobs in S_i before all jobs in $L_i \cup M_i$.*

Proof: We will show that the cost is not increased much. By lemma 3, we know that there is a schedule with a $1+\epsilon$ loss from the optimal, which schedules first the jobs in S_i and then the jobs in L_i. The cost of this schedule is $C(S_i \cup L_i) = OPT(S_i) + OPT(L_i) + M^{S_i}|L_i|$ where M^{S_i} is the maximum makespan of the optimum schedule in S_i.

The cost of a schedule which schedules first S_i and then $M \cup L_i$ is $OPT(S_i) + OPT(M_i \cup L_i) + M^{S_i}|M_i \cup L_i|$. Since, we already know that $OPT(M_i \cup L_i) < (1+\epsilon)OPT(L_i)$ (by Lemma 4), we need to bound $M^{S_i}|M_i|$.

Let $p_{\min}^{L_i}$ be the shortest processing time of a job in L_i, then:

$$\frac{|L_i|^2}{4} p_{\max}^{S_i} < \frac{|L_i|^2}{4} p_{\min}^{L_i} < OPT(L_i) < \epsilon^2 OPT(L_i) \qquad (1)$$

hence $|L_i| < 2\epsilon \sqrt{\frac{OPT(L_i)}{p_{\max}^{S_i}}}$. From the observation in the beginning of this subsection: $M^{S_i} < 2\sqrt{p_{\max}^{S_i} OPT(S_i)}$, thus $M^{S_i}|L_i|$ is at most $2\epsilon OPT$. ∎

The above transformations together with a polynomial algorithm for the preemptive variant [2] yield a PTAS for this problem. We develop the algorithm in two stages. First suppose there are no jobs with processing time $> \epsilon^{10} OPT$. Then we get the algorithm:

(1) Solve the preemptive problem with at most $1+\epsilon$ loss.
(2) Reorder jobs in each type by non-decreasing processing time.
(3) Reorder to obtain a schedule with at most one preemption per interval.
(4) Move large uniprocessor jobs in long intervals so that starting time for uniprocessor jobs is greater than $\frac{1}{\epsilon^2} p_j$, according to Lemma 1.
(5) Consider each preempted job, j, (it is always a single-machine job) and finish it at the time of preemption.

The analysis of this algorithm is straightforward from the above lemmas.

Now, the overall algorithm is:

(1) For every $i < \log_{1+\epsilon} \frac{1}{\epsilon^{10}}$, partition the jobs into L_i, M_i, S_i.
(2) Construct a non-preemptive schedule of S_i as above.
(3) Construct an optimal non-preemptive schedule of $L_i \cup M_i$ by exhaustive search.
(4) Concatenate the schedule of S_i and the schedule of $L_i \cup M_i$.
(5) Choose the best schedule over all choices of i.

6 Open Questions

The recent activity on MIN-SUM scheduling problems has led to resolving many of the complexity questions, as is shown in Table 1. Several open questions can be picked up by looking at this table. Schuurman and Woeginger in [39] provide also an excellent listing of ten open problems in scheduling theory, including MIN-SUM scheduling problems. In [39], they also discuss related MIN-SUM scheduling problems that we do not address here such as shop scheduling problems and scheduling problems with communication delays.

However, we believe that some of the most interesting open questions can be classified into three directions.

(i) The variants with precedence constraints are still far from being resolved. Any ρ-approximation algorithm, with $\rho < 2$ or any negative result for $1|prec|\sum w_j C_j$ or $1|prec|\sum C_j$ will be of great interest. Note that at least four different 2-approximation algorithms are known (see Table 1) for this variant. Moreover, Woeginger [50], gave a proof that eight special cases of $1|prec|\sum w_j C_j$ (including $1|prec|\sum C_j$) all have the same approximation threshold. Thus, what we can, possibly, expect is a negative result like the following.

Conjecture 1 *There is no $(2-\epsilon)$-approximation algorithm for $1|prec|\sum C_j$.*

(ii) The unrelated machines case, with the number of machines as part of the problem's instance, is also of interest. We know that $R|\,|\sum C_j$ and hence $R|r_j|\sum C_j$ are MAX SNP-hard [22] and that their weighted variants can be approximated within a factor of 3/2 [40,42] and 2 [43], respectively. We also know that their weighted preemptive variants can be approximated within a factor of 2 and 3 [43], respectively. However, the existence of an approximability threshold or a PTAS for these preemptive weighted or even un-weighted variants, is an open question. Last but not least it is still open whether $R|pmnt|\sum C_j$ is NP-hard or polynomial time solvable. It seems therefore that there is enough room for improvements here and that any conjecture will be very risky.

(iii) Neither PTASs nor constant approximation algorithms are known for the case with objective function the total (weighted) flow time. Such a result is not however expected for the non preemptive variants due to known non-approximability results [24,25]. Recent work in [9] and [7] (see the discussion in Section 3) for $1|r_j, pmtn|\sum w_j F_j$ provides a strong evidence that the following conjecture holds.

Conjecture 2 *There is a PTAS for* $1|r_j, pmtn|\sum w_j F_j$.

For the parallel machines preemptive unweighted variant $P|r_j, pmtn|\sum F_j$ an $O(\log \min\{\frac{n}{m}, \log P\})$-approximation algorithm is known [25], whereas no negative result has been shown. It would be interesting to have either an better approximation factor or a negative result for this variant or even for $P2|r_j, pmtn|\sum F_j$.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, M. Sviridenko: Approximation schemes for minimizing average weighted completion time with release dates. In Proceedings of 40th FOCS (1999) 32–43.

[2] F. Afrati, E. Bampis, A. V. Fishkin, K. Jansen, C. Kenyon: Scheduling to minimize the average completion time of dedicated tasks. In Proceedings of 20th FSTTCS (2000) 454-464.

[3] F. Afrati, E. Bampis, C. Kenyon, I. Milis: Scheduling to minimize the weighted sum of completion times. Journal of Scheduling **3** (2000) 323–332.

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

[5] P. Bruker: Scheduling Algorithms. Springer, 1998. See also http://www.mathematik.uni-osnabrueck.de/research/OR/class/

[6] X. Cai, C. Y. Lee, C. L. Li: Minimizing total completion time in two-processor task systems with prespecified processor allocations. Naval Research Logistics, **45** (1998) 231–242.

[7] C. Chekuri, S. Khanna: Approximation Schemes for Preemptive Weighted Flow Time. Electronic Colloquium on Computational Complexity (ECCC) 8(065) (2001).

[8] C. Chekuri, S. Khanna: A PTAS for Minimizing Weighted Completion Time on Uniformly Related Machines. In Proceedings of 28th ICAPLP (2001) 848-861.

[9] C. Chekuri, S. Khanna, An Zhu: Algorithms for minimizing weighted flow time. In Proceedings of 33rd STOC (2001) 84-93.

[10] C. Chekuri, R. Motwani: Minimizing weigthed completion time on a single machine. In Proceedings of 10th SODA (1999) 873–874.

[11] C. Chekuri, R. Motwani, B. Natarajan, C. Stein: Approximation techniques for average completion time scheduling. In Proceedings of 8th SODA (1997) 609–618.

[12] F. A. Chudak, D. S. Hochbaum: A half-integer linear programming relaxation for scheduling precedence-constrained jobs on a single machine. Operations Research Letters **25** (1999) 199–204.

[13] M. E. Dyer, L. A. Wosley: Formulating the single machine sequencing problem with release dates as a mixed integer program. Discrete Applied Math. **26** (1990) 255–270.

[14] G. Even, J. Naor, S. Rao, B. Schieber: Divide-and-conquer approximation algorithms via spreading matrices. In Proceeding of 36th FOCS (1995) 62–71.

[15] A. V. Fishkin, K. Jansen, L. Porkolab: On minimizing average weigthed time completion time of multiprocessor tasks with release dates. In Proceedings of 28th ICALP (2001) 875–886 .

[16] A. V. Fishkin, K. Jansen, L. Porkolab: On minimizing average weigthed time completion time: A PTAS for scheduling general multiprocessor tasks. In Proceedings of 13th FCT-Workshop on Efficient Algorithms (WEA), LNCS-2138 (2001) 495-507.

[17] M. X. Goemans, M. Queyranne, A. S. Shulz, M. Skutella, Y. Wang: Single machine scheduling with release dates. SIAM Journal on Discrete Mathemetics, to appear (see http://www.math.tu-berlin.de/~skutella/publications.html).

[18] R.L. Graham: Bounds on certain multiprocessor anomalies. Bell Systems Tech. J. **45** (1996) 1563–1581.

[19] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan: Optimization and approximation in deterministic sequencing and scheduling. Ann. Discrete Math. **5** (1979) 287–326.

[20] L. A. Hall, A. S. Schulz, D. B. Shmoys, J. Wein: Scheduling to minimize average completion time: on-line and off-line approximation algorithms. Mathematics of Operations Research **22** (1997) 513–544.

[21] L.A. Hall, D.B. Shmoys, J. Wein: Scheduling to minimize average completion time: on-line and off-line algorithms. In Proceedings of 7th SODA (1996) 142–151.

[22] H. Hoogeveen, P. Schuurman, G.J. Woeginger: Non-approximability results for scheduling problems with minsum criteria. In Proceedings of 6th IPCO, LNCS 1412 (1998) 353–366.

[23] T. Kawaguchi, S. Kyan: Worst case bound of an LRF schedule for the mean weighted flow-time problem. SIAM J. on Computing **15** (1986) 1119–1129.

[24] H. Kellerer, T. Tautenhahnm, G. J. Woeginger: Approximability and nonappproximability results for minimizing total flow time on a single machine. In Proceedings of 28th STOC (1996) 418–426.

[25] S. Leonardi, D. Raz: Approximating total flow time on parallel machines. In Proceedings of 29th STOC (1997) 110-119.

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

[27] F. Margot, M. Queyranne, Y. Wang: decompositions, network flows, and a precedence constrained single machine scheduling problem. Report No. 2000-29, Department of Mathematics, University of Kentucky, Lexington.

[28] A. Munier, M. Queyranne, A. S. Schulz: Approximation bounds for a general class of precedence-constrained parallel machine scheduling problems. In Proceedings of 6th IPCO, LNCS 1412 (1998) 367–382.

[29] C. Philips, C. Stein, J. Wein: Scheduling job that arrive over time. In Proceedings of 4th WADS, LNCS 955 (1995) 290–301.

[30] C.N. Potts: An algorithm for the single machine sequencing problem with precedence constraints. Math. Programming Stud. **13** (1980) 78–87.

[31] M. Queyranne: Structure of a simple scheduling polyhedron. Math. Programming **58** (1993) 263–285.

[32] M. Queyranne, A. S. Shulz: Polyhedral approaches to machine scheduling. Preprint No. 408/1994, Department of Mathematics, Technical University Berlin, 1994.

[33] R. Ravi, A. Agrawal, P. Klein: Ordering problems approximated: single processor scheduling and interval graph completion. In Proceedings of 18th ICALP, LNCS 510 (1991) 751–762.

[34] S. Sahni: Algorithms for scheduling independent tasks. J. ACM **23** (1976) 116–127.

[35] M. W. P. Savelsbergh, R. N. Uma, J. M. Wein: An experimental study of LP-based approximation algorithms for scheduling problems. In Proceedings of 9th SODA (1998) 453–462.

[36] A.S. Schulz, M. Skutella: Random-based scheduling: New approximations and LP lower bounds. In Proceedings of RANDOM'97, LNCS 1269 (1997) 119-133.

[37] A.S. Schulz, M. Skutella: Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria. In Proceedings of 5th ESA, LNCS 1284 (1997) 416-429.

[38] A. S. Schulz, M. Skutella: The Power of alpha-Points in Preemptive Single Machine Scheduling. Journal of Scheduling, to appear (see http://www.math.tu-berlin.de/~skutella/publications.html).

[39] P. Schuurman, G.J. Woeginger: Polynomial time approximation algorithms for machine scheduling: Ten open problems. J. of Scheduling **2** (1999) 203-213.

[40] J. Sethuraman, M. S. Squillante: Optimal scheduling of multiclass parallel machines. In Proceedings of 10th SODA (1999) 963–964.

[41] D. B. Shmoys: Using linear programming in the design and analysis of approximation algorithms: Two illustrative problems. In Proceedings of APPROX'98, LNCS 1444 (1998) 15-32.

[42] M. Skutella: Semidefinite relaxations for parallel machine scheduling. In Proceedings of 39th FOCS (1998) 472–481.

[43] M. Skutella: Convex quadratic programming relaxations for network scheduling problems. In Proceedings of 7th ESA, LNCS 1643 (1999) 127–138.

[44] M. Skutella, G.J. Woeginger: A PTAS for minimizing the weigthed sum of job completion times on parallel machines. In Proceedings of 31th STOC (1999) 400–407.

[45] W. Smith: Various optimizers for single-stage production. Naval Res. Logist. Quart. **3** (1956) 59-66.

[46] E. Torng, P. Uthaisombut: Lower bounds for srpt-subsequence algorithms for nonpreemptive scheduling. In Proceedings of 10th SODA (1999) 973–974.

[47] J. Turek, W. Ludwing, J. Wolf, P. Yu: Scheduling parallel tasks to minimize average responce times. In Proceedings of 5th SODA (1994) 112-121.

[48] R. N. Uma, J. Wein: On the relationship between combinatorial and lp-based approaches to NP-hard scheduling problems. In Proceedings of 7th IPCO (1999) 394–408.

[49] G.J. Woeginger: When does a dynamic programming formulation guarantee the existence of an FPTAS? In Proceedings of 10th SODA (1999) 820-829.

[50] G.J. Woeginger: On the approximability of average completion time scheduling under precedence constraints. In Proceedings of 28th ICALP (2001) 887-897.

[51] L. A. Wosley: Mixed integer programming formulations for production planning and scheduling problems. Invited talk at 12th International Symposium on Mathematical Programming, MIT, Cambriddge, 1985.