Randomized Approximation Schemes for Scheduling Unrelated Parallel Machines

Pavlos S. Efraimidis* Paul G. Spirakis

Computer Technology Institute
Department of Computer Engineering and Informatics
University of Patras
efraimid@cti.gr, spirakis@cti.gr

Abstract

The problem of Scheduling n Independent Jobs on m Unrelated Parallel Machines, when the number of machines m is fixed, is considered. The standard problem of minimizing the makespan of the schedule (*SUM*) and the bicriteria problem of scheduling with bounded makespan and cost (*SUMC*) are addressed, and randomized fully linear time approximation schemes are shown for both of them. While matching the approximation guarantee and the complexity of the best known sequential results of Jansen and Porkolab ([12]), the proposed algorithms exhibit a significantly simpler and more general rounding scheme, especially for the bicriteria *SUMC* problem, and admit simple optimal work parallelizations[1] of $O(\log n)$–time complexity. The core of the algorithms, which also draw techniques from other related works ([12], [11], [1]), is an interesting new randomized rounding procedure, the Filtered Randomized Rounding (*FRR*). In the settings of the problems considered, *FRR* boosts the deviation bounds of the rounded linear packing constraints to any given constant ratio.

Finally, the notion of *poly–bottleneck* combinatorial optimization problems is defined and used to build $O(n \log n \log \log n)$ time approximation schemes for two natural optimization versions of *SUMC*, that is minimizing the makespan when the cost of the schedule is bounded (*SUMCoptT*) and minimizing the cost when the makespan is bounded (*SUMCoptC*). These algorithms too, admit simple optimal work parallelizations.

*Financial support from the Bodosaki Foundation to perform doctoral studies is gratefully announced. Bodosaki Foundation, Leoforos Amalias 20, 10557 Athina, Greece

[1]Optimal work parallelization means that the parallel work (product of running time and number of processors) is equal to the sequential running time.

1 Introduction

Scheduling n independent jobs on m Unrelated Parallel Machines raises the problem of assigning n jobs $j = \{1, \ldots, n\}$ to m machines $i = \{1, \ldots, m\}$ in the way that each job is processed without interruption on one of the machines, and at any time, every machine processes at most one job. The processing time for job j on machine i is p_{ij}. For each schedule, the maximum load on any machine is the makespan of the schedule. The objective of the common scheduling problem *SUM* is to find a schedule of minimum makespan. In the bicriteria problem *SUMC*, the assignment of each job j to a machine i has, besides the processing time p_{ij}, a cost c_{ij}, and the objective is to find a schedule of bounded cost and makespan.

Due to its theoretical and practical importance, the problem of scheduling n independent jobs on m unrelated parallel machines has been widely studied. It is known to be NP-hard when the number of jobs n and the number of machines m are free parameters of the problem, and interestingly, it remains NP-hard even when the number of machines is defined to be $m = 2$. For the general problem, where both parameters m and n are specified as part of the problem instance, the best known results are due to Lenstra, Shmoys and Tardos. They showed in [15] a polynomial time 2-approximation algorithm for the general problem and in the same work they also proved that, unless P=NP, no approximation ratio better than 3/2 is possible. This inapproximability result raises the natural question whether better approximation results can be obtained for the case where the number m of machines is a constant.

The focus of this work are approximation algorithms for *SUM* and *SUMC* when the number of machines is fixed, and henceforth the number of machines is always assumed to be a constant. Horowitz and Sahni showed in ([11]) how to obtain for *SUM* a polynomial on n and $1/\epsilon$, $(1+\epsilon)$–approximation algorithm for any fixed $\epsilon > 0$. Such a family of algorithms is called a *fully polynomial time approximation scheme* (*FPTAS*). A polynomial on n $(1+\epsilon)$–approximation algorithm for any fixed $\epsilon > 0$ where given in [15]. Such an algorithm is called a *polynomial time approximation scheme* (*PTAS*) since its running time depends exponentially on $1/\epsilon$. Even though in this aspect the algorithm of [15] is inferior to the one of [11], it achieves a significantly smaller space complexity than the former algorithm. Kopidakis, Fayard and Zisimopoulos presented in [14] a linear time *PTAS* for *SUM*β, a restricted version of *SUM* where all processing times are within a constant factor of each other.

Recently, Jansen and Porkolab showed in [12] linear time approximation algorithms for *SUM* and *SUMC*. More precisely, they presented a linear time approximation scheme for *SUM* and an ϵ–relaxed decision procedure (*RDP*) for *SUMC*. A *RDP* is an algorithm that given a minimization problem, a constant $\epsilon > 0$, and a value d,

- either decides that there is no solution of objective value at most d.
- or returns a solution of objective value at most $(1+\epsilon)d$

The *RDP* for *SUMC* in [12] accepts as input an instance of *SUMC* and values T for makespan and C for cost, and either finds a schedule of makespan and cost at most $(1+\epsilon)T$ and $(1+\epsilon)C$ respective, or decides that there is no schedule of makespan and cost at most T and C respectively. A *randomized RDP* (*RRDP*), is a randomized algorithm that given a minimization problem, constants $\epsilon > 0$ and

$\rho: 0 < \rho < 1$, and a value d, with probability of success at least $p > \frac{1}{2}$, either decides that there is no solution of objective value at most d, or returns a solution of objective value at most $(1+\epsilon)d$.

Due to the fact that *SUMC* has two objectives, i.e. the makespan T and the cost C of the schedule, several optimization versions of *SUMC* are possible. Two natural problems are obtained by specifying an upper bound on the one objective and then optimizing the second objective under this condition. In this way, let *SUMCoptT* be the problem of finding for a given instance of *SUMC* and a specified cost value C, the schedule of minimum makespan for cost at most C. Similarly let *SUMCoptC* be the problem of finding for given *SUMC* and makespan T a schedule of minimum cost C and makespan at most T. A third option is the problem *SUMCoptTC* that optimizes a linear function of the makespan and the cost. The *RDP* for *SUM* of [12] is used in the same work for a linear time FPTAS for *SUMCoptTC*.

In the field of parallel algorithms for *SUM* and *SUMC* we are aware of the work of Serna and Xhafa, who claim in [22] a randomized $(2+\epsilon)$–approximation algorithm for *SUM* and a randomized a $(2+\epsilon)$–makespan 2–cost approximation algorithm for *SUMCoptC*. Both algorithms run in $polylog(N)$ time on $O(N)$ processors, where N is the instance size. Another related work is [4], where the authors propose a framework that achieves parallel polylog(n) time approximation algorithms for *SUM* and *SUMC*. However the results in [4] are not directly comparable with the current work since they work for any number m of machines and achieve a logarithmic performance bound.

A *randomized PTAS* (*RPTAS*) is a randomized algorithm that accepts as input a problem instance and a constant $\epsilon > 0$, runs in time polynomial on the size N of the instance, and produces as output a $(1+\epsilon)$–approximate solution with probability $p > \frac{1}{2}$.[2] If additionally the running time depends at most polynomially on $1/\epsilon$ then it is a *randomized FPTAS* (*RFPTAS*).

We first show algorithm *A-SUMβ*, a simple linear time (*RFPTAS*) for the restricted *SUMβ* problem. Algorithm *A-SUMβ* is based on approximate linear programming and standard randomized rounding, and it is meant to be a smooth introduction to the techniques used in the more involved algorithms of this work. However, algorithm *A-SUMβ* is itself an interesting result, due to its simplicity and since it matches the complexity and the performance guarantee of the best known specific algorithm for *SUMβ* of [14].

We then address the standard *SUM* problem and the bicriteria *SUMC* problem and present efficient randomized approximation algorithms for both problems. The core of our algorithms, which also draw techniques from the related works ([12], [11], [1]), is an interesting new randomized rounding procedure, the Filtered Randomized Rounding (*FRR*) technique. The striking feature of FRR is that, while rounding fractional schedules, it boosts, in the settings of the problems considered, the deviation bounds of the rounded linear packing constraints to any given constant ratio. *FRR* appears to be a general technique of independent interest that uses randomized rounding, Chernoff bounds and combinatorial arguments from [12], [11], and [1], and it should find more applications in rounding procedures for other integer linear programs. The algorithms based on *FRR* are very simple, once the general *FRR* technique has been understood.

We show algorithm *A-SUM* a linear time *RRDP* for *SUM* and use it to build a linear time *RFPTAS* for *SUM*. Similarly, we show algorithm *A-SUMC* a linear time *RRDP* for *SUMC*. These

[2] The success probability $p > \frac{1}{2}$ can be boosted to any constant probability in [p,1) by repeating the experiment a constant number of times.

results for *SUM* and *SUMC*, while matching in performance guarantee and complexity the best known sequential results of Jansen and Porkolab ([12]), exhibit, due to the *FRR* rounding technique, a significantly simpler and more general rounding scheme. This becomes especially evident on the approximation algorithm for the bicriteria *SUMC* problem.

Finally, we define the notion of *poly–bottleneck* combinatorial optimization problems and use it to build $O(n \log n \log\log n)$ time approximation schemes for two natural optimization versions of *SUMC*, that is minimizing the makespan when the cost of the schedule is bounded (*SUMCoptT*) and minimizing the cost when the makespan is bounded (*SUMCoptC*). For the combined objective problem *SUMCoptTC*, a linear time *RFPTAS* can be obtained by using algorithm *A-SUMC* within the technique of [12, Theorem 3.2].

All algorithms admit simple optimal work parallelizations which run either in $O(\log n)$ or in $O(\log n \log\log n)$ time and clearly outperform the best claimed parallel algorithms for *SUM* and *SUMCoptC* ([22]), both in the performance ratio (ϵ vs. $2+\epsilon$) and in the running time (($O(\log n)$ or $O(\log n \log\log n)$) vs. $polylog(n)$). The parallel running times are valid for the EREW PRAM, the most realistic of the PRAM models. Furthermore, since the algorithms are executed in a constant $O(1)$ or at most a very small number ($O(\log n \log\log n)$) of iterations, these algorithms imply efficient parallel algorithms on more practical parallel computation models, like the *BSP* ([24]) or the *LogP* ([2].

The algorithms of this work follow the common paradigm of calculating a fractional schedule with Linear Programming techniques and then rounding it to a near-optimal integer schedule. The linear programs that occur in our algorithms have a block-angular structure and are approximated very efficiently, sequentially or in parallel with the logarithmic-potential based price-directive decomposition method (*PDD*) of Grigoriadis and Khachiyan ([6]).

When we say that an event holds "with high probability (*whp*)" we will mean that its probability is at least $1 - 1/n^f$ for some large enough f. The abbreviation "wlog" stands for the expression "without lost of generality". $E[X_{ij}]$ represents the mean value of the random variable X_{ij}. N will always represent the size of a problem instance. The set $\{1, 2, \ldots, n\}$ will be represented as $[n]$. Section C of the Appendix provides an easy access to the important definitions and notations used in this work. The rest of this work is organized as follows. First a simple *RFPTAS* for the restricted problem *SUMβ* is shown in Sec. 2. Then the *FRR* technique is applied in Sec.3 for a *RFPTAS* for *SUM* and in Sec.4 for a *RRDP* for the bicriteria problem *SUMC*. The optimization versions of *SUMC* are treated in Sec.5.

2 The *SUMβ* Problem

SUMβ is the standard *SUM* problem with the additional constraint that the processing times do not differ with each other more than a constant factor β. More precisely, there is a constant β such that for every instance P of *SUMβ*,

$$\frac{p_{min}}{p_{max}} \geq \frac{1}{\beta} \ . \qquad (1)$$

where $p_{min} = \min_{i,j} p_{ij}$ and $p_{max} = \max_{i,j} p_{ij}$. Even though *SUMβ* is a restricted subcase of *SUM*, is has an autonomous presence in the literature and several specific approximation algorithms are known for it. The best specific sequential result is the linear time *FPTAS* of [14]. In [22] the authors claim a randomized polylog(n) time parallel algorithm for *SUM*β with approximation ratio $(2+\epsilon)$. In what follows we will show a simple linear time *RFPTAS* for *SUM*β. The algorithm admits a simple optimal work $(O(\log n))$–time parallelization on a $\frac{n}{\log n}$ processor EREW PRAM. The algorithm *SUM*β uses only a subset of the techniques that are presented in this work, and can serve as a smooth introduction to the main algorithms *A-SUM* and *A-SUMC* defined in the following Sections. Note that algorithm *A-SUM* of Sec. 3 for the *SUM* problem can also solve the *SUM*β problem. However *A-SUM*β is simpler than *A-SUM* algorithm and achieves on *SUM*β a faster, within a constant factor, running time.

2.1 Algorithm *A-SUM*β

Algorithm *A-SUM*β has a very simple structure. A fractional schedule is found with a linear programming technique, and then it is rounded to an approximate integer schedule with a standard randomized rounding technique. This approach can satisfy any given constant approximate ratio, if the number of jobs n larger than an appropriate constant n_0, which depends on m, ρ, and ϵ. For instances with number of jobs n less than n_0 the optimal schedule can be found in constant time with a brute force method.

Input: An instance of *SUM*β and constants $\epsilon > 0$ and $0 < \rho < \frac{1}{2}$.

Output: Produce a schedule, that with probability at least $(1-\rho)$ is an $(1+\epsilon)$-approximate schedule.

Step 0: *Initializations.* Let $\epsilon_2 = \epsilon_4 = \frac{\epsilon}{3}$. Set $\mu = \frac{3\ln(m/\rho)}{(\epsilon_4)^2}$ and $n_0 = m \cdot \beta \cdot \mu$.

Step 1: *Small number of jobs. IF* $(n < n_0)$ *THEN* the number of jobs n is less than a constant and hence the optimal schedule can be found in $O(1)$–time with a brute force method.

Step 2: *Integer program formulation.* The number of jobs n is $n \geq n_0$. Formulate *SUM*β as an integer linear program and relax it to a linear program.

Step 3: *Approximate LP solution.* Find a $(1+\epsilon_2)$—approximate solution to the LP with algorithm PDD.

Step 4: *Rounding.* Round the approximate fractional solution to an approximate integer schedule with randomized rounding. *END*

2.2 Analysis of algorithm *A-SUM*β

Normalization. The problem is scaled with $1/p_{max}$. The normalization is done only to simplify the analysis of the algorithm and it is not used in the algorithm. Now:

$$\forall\, i, j \;:\; \frac{1}{\beta} \;\leq\; p_{ij} \;\leq\; 1 \;. \tag{2}$$

Let $\epsilon_2 = \epsilon_4 = \frac{\epsilon}{3} = O(\epsilon)$, $\mu = \frac{3\ln\frac{m}{\rho}}{(\epsilon_4)^2}$. Let μ be the marginal mean value for the processing load on each machine and $n_0 = m \cdot \beta \cdot \mu$ be a threshold value for the number n of jobs. Note that μ and n_0 are constants.

Small number of jobs. If $n < n_0$ then there are at most m^{n_0} possible assignments of the jobs to the machines, and hence the optimal schedule one can be found in constant time with a brute force assignment. This constant time can be further reduced to a smaller fully polynomial constant time with techniques $\mathcal{T}_1$ and $\mathcal{T}_2$ of Sec. 3.3 for finding only a $(1+\epsilon)$ approximate schedule, which is adequate for the algorithm.

Fractional Schedule. The number of jobs n is assumed to be $n > n_0$. It will be shown that under this condition the makespan of the optimal schedule is large enough to guarantee with positive probability very tight randomized rounding. The integer program formulation of SUMb is:

$$
\begin{array}{ll}
min & \tau \\
s.t. & \\
 & \begin{array}{ll}
\sum_{j=1}^{n} p_{ij} x_{ij} \le \tau & (i = 1, \ldots, m) \\
\sum_{i=1}^{m} x_{ij} = 1 & (j = 1, \ldots, n) \\
x_{ij} \in \{0, 1\} & (i = 1, \ldots, m;\ j = 1, \ldots, n)
\end{array}
\end{array}
$$

For each pair (i, j) the binary variable x_{ij} is 1 if job j is assigned to machine i and 0 otherwise. Relaxing the integrality constraints on x_{ij} to $x_{ij} \ge 0$ gives the linear program *LP-SUMβ* that can be solved optimally or approximately with any of several known polynomial time algorithms for linear programming (see Sec. B of the Appendix). However *LP-SUMβ* has specific properties that can be used to achieve very efficient approximate solutions. The problem variables of *LP-SUMβ* are grouped into n independent m-dimensional simplices (blocks) and there is a constant number of positive packing constraints. These properties are exploited by the logarithmic-potential based PDD algorithm *PDD* of Grigoriadis and Khanchiyan ([6]), to approximate *LP-SUMβ* within $(1+\epsilon_1)$. This is formally stated in the following Claim which will be used throughout this work. The proof is placed in Sec. B of the Appendix.

Claim 2.1 *The linear program* LP-SUMβ *can be approximated within any constant ratio* ϵ *in* $(O(n))$*-sequential time and in* $O(\log n)$*-time on a* $O(n/\log n)$*-processor EREW PRAM.*

Let τ^* be the optimal objective value of *LP-SUMβ* and let τ_1 be the approximate objective value produced by the PDD algorithm for a given error ratio ϵ_2. Then:

$$
\begin{array}{ll}
\sum_{j=1}^{n} p_{ij} x_{ij} \le \tau_1 & (i = 1, \ldots, m) \\
\sum_{i=1}^{m} x_{ij} = 1 & (j = 1, \ldots, n) \\
x_{ij} \ge 0 & (i = 1, \ldots, m;\ j = 1, \ldots, n)
\end{array}
$$

The linear program *LP-SUMβ* is the relaxation of the integer program formulation of *SUMβ*. If the optimal objective for the integer schedule problem is *OPT* then the optimal value τ^* of the relaxed problem *LP-SUMβ* cannot be larger than *OPT*. Furthermore the approximation ratio of algorithm *PDD* guarantees that $\tau_1 \le \tau^* \cdot (1+\epsilon_2)$. Hence:

$$\tau^* \le \tau_1 \le \tau^* \cdot (1+\epsilon_2) \le OPT \cdot (1+\epsilon_2) \ . \tag{3}$$

Let d_j be the minimum processing time of job j, and let D be the sum of all d_j:

$$\forall \text{ job } j,\ d_j = \min_i p_{ij} \text{ and } D = \sum_j d_j \ . \tag{4}$$

The definition of D implies that the optimal amount of work for processing all jobs is exactly D. Simply assigning assigning every job j to the machine i that achieves the minimum processing time d_j for the job j, would give a feasible schedule of makespan at most D, even if in the worst case all jobs j would have their minimum processing time on the same machine i. Since, on the other hand, the total work of the machines is at least D, even if this work would be optimally distributed to all machines, the makespan could still not be less than D/m. Hence:

$$\frac{D}{m} \leq OPT \leq D \ . \tag{5}$$

The lower bound $\frac{D}{m}$ on the makespan is valid even if fractional schedules are admitted. This fact together with equation 3 provides a lower bound on the approximate fractional makespan τ_1:

$$\frac{D}{m} \leq \tau^* \leq \tau_1 \ . \tag{6}$$

Note that Equ. 5 and 6 are valid also for the SUM problem (the condition of Equ. 2 on the sizes of the p_{ij} has not been used for showing the equations) and they will be used in Sec. 3. The combination of Equ. 5 and Equ. 2 gives $\frac{D}{m} \geq \frac{n}{Bm}$ and hence:

$$\tau_1 \geq \frac{D}{m} \geq \frac{n}{Bm} \geq \frac{mB\mu}{Bm} \geq \mu = \frac{3\log\left(\frac{m}{\rho}\right)}{(\epsilon_4)^2} \ . \tag{7}$$

Equations 4, 7 and 3 give:

$$\frac{3\log\left(\frac{m}{\rho}\right)}{(\epsilon_4)^2} \leq \tau_1 \leq OPT \cdot (1+\epsilon_2) \ . \tag{8}$$

Rounding. The approximate fractional solution to the scheduling problem will be rounded to an approximate integer solution, with a standard randomized rounding ([20]) technique. More precisely, the appropriate rounding procedure that will be called "Exclusive" Randomized Rounding (*XRR*) is similar to Raghavan and Thompson's randomized rounding technique of Sec. 2 in [20], but generalized to the case of weighted sums of Bernulli trials. For each job j independently, exactly one of the x_{ij} is set to 1 and the rest is set to 0. Setting the variable x_{ij} to 1, represents the fact that job j is assigned to the machines i. The rounding is done in such a way, that the probability for variable x_{ij} to get the value 1 is equal to its fractional value x_{ij}. Let τ_2 be the makespan of the rounded schedule.

For each machine i, let $S_i = \sum_i p_{ij}x_{ij}$ be its processing load in the fractional schedule. The rounding procedure essentially replaces in each constraint the fractional variable x_{ij} with a Bernulli trial X_{ij} such that $E[X_{ij}] = x_{ij}$. If Ψ_i is the processing load of each machine i in the rounded schedule, then Ψ_i is the random variable $\Psi_i = \sum_i p_{ij}X_{ij}$. Since the Bernulli trials X_{ij} of the same constraint are independent with each other, the random variables Ψ_i are equal to the *Weighted Sums of independent Bernulli Trials*. By linearity of expectation, for each machine i, the mean value of its rounded load Ψ_i is equal to its load S_i in the fractional schedule:

$$\forall\, i \;:\; E[\Psi_i] = E[\sum_i p_{ij} X_{ij}] = \sum_i p_{ij} E[X_{ij}] = \sum_i p_{ij} x_{ij} = S_i \;. \tag{9}$$

This shows that the mean values of the Ψ_i satisfy the packing constraints of the fractional solution. However the random variables Ψ_i might deviate above their mean value (and beyond of course) and hence the makespan of the rounded schedule might be larger than the fractional makespan. The following bounds are a generalization of Raghavan and Spencer's Chernoff-like bounds on the tail of the distribution of the Weighted Sum of Bernulli Trials ([19], [18]). The proof is placed in Sec. A of the Appendix.

Theorem 2.1 *Let $\lambda \geq 0$ be a positive real number and let $\alpha_1, \alpha_2, \ldots, \alpha_r$ be reals in (0,1]. Let $X_1, X_2, \ldots, X_r$ be independent Bernulli trials with $E[X_j] = p_j$. Let $\Psi = \lambda + \sum_{j=1}^r a_j X_j$. Then $E[\Psi] = \lambda + \sum_{j=1}^r a_j p_j = S$. Let $\epsilon_4 > 0$, and $T \geq S = E[\Psi] > 0$. Then*

$$Prob[\Psi > (1+\epsilon_4)T] < e^{\left(-\frac{(\epsilon_4)^2 T}{2(1+\frac{\epsilon_4}{3})}\right)} \text{ and for } \epsilon_4 < 1 \;:\; e^{\left(-\frac{(\epsilon_4)^2 T}{2(1+\frac{\epsilon_4}{3})}\right)} \leq e^{\left(-\frac{(\epsilon_4)^2 T}{3}\right)} \;. \tag{10}$$

Note 2.1 *Note that the tightness of the Chernoff bounds depends on the relative size of the maximum coefficient a_j and the mean value S of the sum of the random variables. Equation 10 can bound with positive probability, the deviation above the value $T \geq S$, by any given constant ratio, if the ratio of $\frac{S}{\max_j\{a_j\}}$ has a large enough value. This fact becomes more evident in the simplified bound for $\epsilon_4 < 1$ of the same Equation.*

Theorem 2.2 *For $\epsilon_4 \in (0,1)$ and $\rho \in (0,1)$, the makespan of the rounded schedule is, with probability at least $(1-\rho)$, not larger than $\tau_1(1+\epsilon_4)$.*

Proof: The probability that the makespan of the schedule is not more than $\tau_1(1+\epsilon_4)$ is equal to the probability that no machine load Ψ_i in the rounded schedule is larger than $\tau_1(1+\epsilon_4)$. By Equ. 10:

$$\forall\, i \;:\; P_i = \text{Prob}\{\Psi_i > \tau_1(1+\epsilon_4)\mu\} \leq e^{-\frac{(\epsilon_4)^2 \mu}{3}} \leq \frac{\rho}{m} \;. \tag{11}$$

A sufficient bound on the probability that at least one machine in the rounded schedule has load more than $\tau_1(1+\epsilon_4)$ is the sum of the probabilities P_i.

$$\text{Prob}\{\tau_2 > (1+\epsilon_4)\cdot\tau_1) = \text{Prob}\{\exists i : \Psi_i > \tau_1 \cdot (1+\epsilon_4)\} \leq \sum_i P_i \leq \rho \;. \tag{12}$$

■

If $\tau_2 \leq \tau_1(1+\epsilon_4)$ *THEN* $\tau_2 \leq OPT(1+\epsilon_2)(1+\epsilon_4) \leq (1+\epsilon)OPT$. Hence given an instance of *SUMβ* with n jobs and constants $\epsilon > 0$ and $0 < \rho < 1$, algorithm *A-SUMβ* produces with probability at least $(1-\rho)$, a schedule of makespan not larger than $(1+\epsilon)OPT$. The complexity of the *PDD* algorithm is $O\left(n\left(\frac{m}{\epsilon}\right)^2\ln\left(\frac{m}{\epsilon}\right)\right)$ on 1 processor and $O\left(\log(n)\left(\frac{m}{\epsilon}\right)^2\ln\left(\frac{m}{\epsilon}\right)\right)$ on $\frac{n}{\log n}$ processors. The *XRR* procedure needs $O(mn)$ time on 1 processor or $O(m\log n)$ time on $\frac{n}{\log n}$ processors. In the case of a small number of jobs $n < n_0$ a $(1+\epsilon)$–approximate schedule can be found in $O\left(\left[\frac{m^2\beta\log(m/\rho)}{\epsilon^3}\right]^{m+1}\right)$ time with the brute force enumeration technique $\mathcal{T}_1$ of Sec. 3.3. Putting all these complexities together gives the main Theorem of this Section:

Theorem 2.3 *Algorithm* A-SUMβ *is a* RFPTAS*for the* SUMβ *problem.* A-SUMβ *runs in* $O(n))$ *sequential time and in* $(O(\log n))$*–parallel time on a* $\frac{n}{\log n}$*–processor EREW PRAM.*

3 The *SUM* problem

In this section, we present a linear time *RFPTAS* for the problem *SUM* of scheduling n independent jobs on m unrelated parallel machines, when the number m of machines is fixed. We first show algorithm A_{SUM}, a *RRDP* for the decision version of *SUM*, and then use it to build a simple linear time *RFPTAS* for the optimization version of *SUM*.

3.1 Algorithm *A-SUM*

Algorithm *A-SUM* is based on the same paradigm used in algorithm *A-SUM*β of finding a fractional schedule and then rounding it to an approximate integer schedule. However the standard *XRR* is extended to the Filtered Randomized Rounding technique *FRR*, since standard randomized rounding cannot satisfy the tight approximation guarantee needed for the approximation scheme. The algorithm first selects a constant number of large jobs and tries every possible assignment of them on the machines. For each possible assignment φ of the large jobs, a corresponding fractional schedule of all the jobs is found with the *PDD* algorithm. Among all fractional schedules the one of minimum makespan is selected, and then it is rounded with *XRR* to an integer schedule. *Every job j in the rounded schedule, that is not a large job ($j \notin S_\ell$) and that has that has been randomly assigned to a machine i such that $p_{ij} > 1$ is called an "unlucky" job and is removed from the rounded schedule.* The result is a filtered rounded schedule that satisfies a very tight approximation ratio. All the unlucky jobs are scheduled independently, each on the machine where its processing time is minimized. A simple combinatorial argument shows that the total processing time for the final assignment of the unlucky jobs is at most a given constant fraction of the optimal makespan. The final schedule is with probability at least $(1-\rho)$, a $(1+\epsilon)$–approximate schedule.

Input: An instance of *SUM*, the constants $\epsilon : 0 < \epsilon \le 1$, and $\rho : \ 0 < \rho < 1$, and a feasible makespan value T.

Output: A schedule of makespan at most $(1+\epsilon)T$ or the problem is infeasible for T.

Step 0: *Initializations.*

Let $\epsilon_1 = \epsilon_2 = \epsilon_3 = \epsilon_4 = \epsilon_5 = \epsilon_6 = \frac{\epsilon}{7} = \Theta(\epsilon)$. $\forall j,\ d_j = \min_i\{p_{ij}\}$, $D = \sum_j d_j$, $\mu = \frac{3\ln\frac{2m}{\rho}}{(\epsilon_4)^2}$, $\xi = e^2 + \ln\left(\frac{2m}{\rho}\right)$, and $k = \left\lceil \xi\cdot\mu\cdot m\cdot\frac{m}{\epsilon_3}\right\rceil$.

Step 1: *Simple Filtering.*

$\forall i, j$: *IF* $p_{ij} > T$ *THEN* $x_{ij} = 0$

Step 2: *Large jobs.*

Let $J_\ell = \{j | d_j$ belongs to the k largest $d_j\}$ be the set of large jobs and let Φ be the set of all possible assignments of the large jobs to the machines. Let $\Phi_f$be an appropriate subset of Φ.

Step 3: *Best Fractional Schedule* x_{ij}.

$\forall$ assignment $\varphi \in \Phi_f$ do

1. Formulate the corresponding scheduling problem as an integer program *ILP-SUM*(φ).
2. Relax *ILP-SUM*(φ) to the linear program *LP-SUM*(φ).
3. Find the approximate fractional schedule with algorithm *PDD*.

Among all fractional schedules select the one of minimum makespan.

Step 4: *Filtered Randomized Rounding.*

1. Round the fractional schedule with *XRR*.
2. Filter : If a job j has been randomly assigned to a $p_{ij} > 1$ *THEN* job j is called "unlucky" and is removed from the schedule.
3. $\forall$ unlucky jobs j, assign job j to machine $i = argmin_i\{p_{ij}\}$

3.2 Analysis of algorithm *A-SUM*

In this section, it will be shown that algorithm *A-SUM* is a *RRDP* for *SUM*, that is, given an instance of the *SUM* problem and a makespan T, with probability of success at least $(1-\rho)$ it either produces a schedule of makespan at most $(1+\epsilon)T$ or decides that there is no schedule of makespan at most T. To prove this, it is sufficient to show that if T is a feasible value then algorithm *A-SUM* returns with probability at least $(1-\rho)$ a schedule of makespan at most $(1+\epsilon)T$. The input is an instance of *SUM*, the constants $\epsilon > 0$, and $\rho :\ 0 < \rho < 1$, and a feasible makespan value T.

Initializations. Let $\mu = \frac{3 \ln \frac{2m}{\rho}}{(\epsilon_4)^2}$ be a marginal mean value for the weighted sums of Bernulli Trials. As in Sec. 2 let $d_j = \min_i p_{ij}$ be the minimum processing time for job j and $D = \sum_j d_j$ be the minimum total processing time for all jobs.

Normalization. To simplify the analysis, the problem is scaled by the factor $\frac{\mu}{T}$ so that the given makespan becomes $T = \mu$. As in the Sec. 2 the scaling is done only simplify the analysis of the algorithm and it can be avoided in the real algorithm. The value of $T = \mu$, has been chosen so that if all coefficients p_{ij} would be $p_{ij} \leq 1$, then given a fractional schedule, the *XRR* rounding procedure would return an integer schedule that satisfies the required approximation guarantee of the approximation algorithm. At this point, it would be possible to proceed as in Sec. 2, that is to find an approximate fractional with the *PDD* algorithm and then *XRR* to round it an integer solution for *SUM*. However due to the potential existence of arbitrarily large coefficients $p_{ij} > 1$, the

Chernoff-bound would not guarantee good bounds on the deviations. The next steps deal exactly with this problem, the existence of large coefficients p_{ij}.

Simple Filtering. The first step is a "simple filtering" procedure, that deactivates all x_{ij} for which the corresponding p_{ij} is larger than T. This action does not influence any integer schedule of makespan at most T and hence T remains a feasible makespan value for the problem. After simple filtering, all active p_{ij} are not larger than T, but some of the p_{ij} can still be larger than 1. Now, finding a fractional approximate solution with Claim 2.1 and then applying *XRR*, would give approximate results but still far weaker performance guarantee than the required guarantee. It will be shown that it is possible to obtain fractional schedules and to round them to approximate integer schedules that with positive probability have makespan at most a factor $(1+\epsilon)$ larger than the makespan of the fractional schedule, for any given constant $\epsilon > 0$. This is accomplished with a new randomized rounding technique, called Filtered Randomized Rounding (*FRR*), that is based on standard *XRR* and on certain combinatorial arguments, similar to combinatorial arguments used in [12] and [14] to support different scheduling algorithms. Before finding an approximate fractional schedule, it is necessary to treat a constant number of large jobs. The purpose of this step will become evident in the sequel.

Large Jobs. Let k be the constant

$$k = \left\lceil \xi \cdot \mu \cdot m \cdot \frac{m}{\epsilon_3} \right\rceil \quad , \tag{13}$$

and let the k jobs with the largest's d_j (tights are resolved arbitrarily) be the "large jobs". Let J_ℓ be the set of the k large jobs:

$$J_\ell = \{j \mid d_j \text{ belongs to the } k \text{ largest } d_j\} \ . \tag{14}$$

Enumeration. Let Φ denote the set of all possible assignments φ of the large jobs $j \in J_\ell$ to the m machines. Since the number of machines m is a constant and the number k of large jobs is fixed, the cardinality of Φ is at most m^k, also a constant. Let φ^* be an assignment of the large jobs to the machines that is identical to the placement of the large jobs in an optimal schedule to the *SUM* problem. The algorithm needs to work on the assignment φ^*, and since φ^* is not known, the algorithm is executed for each of the possible assignments $\varphi \in \Phi$. The cardinality of Φ is

$$m^{O\left(\frac{m^2 \ln^2(2m/\rho)}{\epsilon^3}\right)} \ . \tag{15}$$

Even though this number is a constant for constants m and ϵ_4, it is not appropriate for a fully polynomial algorithm, since ϵ_4 appears in the exponent. In Sec. 3.3 it is shown how Φ can be replace by a smaller set, of cardinality

$$O(\min\left\{\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)} , m^{\frac{2\log(1/\epsilon)}{\epsilon}}\right\}) \ . \tag{16}$$

at the cost of introducing at most an extra error ratio $(1+\epsilon_5)\cdot(1+\epsilon_6)$ to the final solution. The rest of the analysis all approximation bounds are calculated with the assumption that the algorithm examines all assignments of Φ. This assumption is later relaxed (Equ. 33) by introducing the extra ratio $(1+\epsilon_5)\cdot(1+\epsilon_6)$ to the final approximation guarantee.

Fractional Schedule. Given the assignment φ^* of the large jobs J_ℓ to the m machines, the problem of assigning the remaining jobs in an optimal way (to minimize the makespan) can be formulated as the following integer linear program *ILP-SUM*(φ^*). Let φ_i be the load on machine i due to the large jobs.

$$\begin{array}{lll} min & \tau & \\ s.t. & & \\ & \varphi_i + \sum_{j\in[n]-J_\ell} p_{ij}x_{ij} \le \tau & (i=1,\ldots,m) \\ & \sum_{i=1}^m x_{ij} = 1 & (j \in [n]-J_\ell) \\ & x_{ij} \in \{0,1\} & (i=1,\ldots,m;\ j\in[n]-J_\ell) \end{array}$$

Let τ^* be the optimal objective value of *ILP-SUM*(φ^*). Since the value T is assumed to be feasible for the problem and φ^* is assumed to be the optimal assignment of the large jobs:

$$\tau^* \le T \ . \tag{17}$$

Let *LP-SUM*(φ^*) be the linear program relaxation of *ILP-SUM*(φ^*) obtained by relaxing the integrality constraints on the variables x_{ij} to $\forall i,j:\ x_{ij}\ge 0$. The linear program *ILP-SUM*(φ^*) has a block-angular structure and as in Claim 2.1 of Sec. 2 it can be approximated with *PDD*, the logarithmic-potential based PDD algorithm of [6]. The complexity of *ILP-SUM*(φ^*) is slightly lower than *LP-SUMβ* of Sec. 2 since a constant number of jobs is already assigned (by the a priori assignment of the large jobs) and some x_{ij} have been set to 0. This difference does not increase the complexity and will be ignored in the analysis. Hence for given constant ϵ_2, the PDD algorithm will run in $O(n)$ sequential time and in $O(\log n)$ parallel time on $n/\log n$ processors, and it will produce a $(1+\epsilon_2)$–approximate fractional schedule. Let x_{ij} and τ_1 be the output produced by PDD. Then:

$$\begin{array}{ll} \varphi_i + \sum_{j\in[n]-J_\ell} p_{ij}x_{ij} \le \tau_1 & (i=1,\ldots,m) \\ \sum_{i=1}^m x_{ij} = 1 & (j \in [n]-J_\ell) \\ x_{ij} \ge 0 & (i=1,\ldots,m;\ j\in[n]-J_\ell) \end{array}$$

By the approximation guarantee of PDD and Equ. 17:

$$\tau_1 \le \tau^*\cdot(1+\epsilon_2) \le opt\cdot(1+\epsilon_2) \ . \tag{18}$$

For each $\varphi\in\Phi$ the PDD algorithm finds an approximate fractional solution to *LP-SUM*(φ). At the end the algorithm selects among all fractional schedules the one with the smallest makespan τ_2. Let $[x_{ij}]$ be a fractional solution with makespan τ_2.

$$\tau_2 = \min_{\varphi\in\Phi}\{\ \tau_\varphi \ \mid\ \tau_\varphi = \epsilon_2 - \text{approximate solution to } \textit{LP-SUM}(\varphi) \text{ found with } \textit{PDD}\} \ . \tag{19}$$

Hence:

$$\tau_2 \leq \tau_1 \leq T \cdot (1 + \epsilon_2) \ . \tag{20}$$

Rounding. The fractional schedule $[x_{ij}]$ will be rounded to an approximate integer schedule for *SUM*. Let J be set of all jobs j and J_s the set of all jobs except the large jobs $J_s = J \backslash J_\ell$ (set difference). For each job $j \in J_s$ independently, exactly one of the x_{ij} is set to 1 and the rest is set to 0, that is each job j independently is assigned to exactly one of the machines i. The probability for job j to be assigned by the rounding procedure to machine i is equal to the fractional value x_{ij}. Let τ_3 be the makespan of the rounded schedule. The rounding procedure is equivalent with replacing for $j \in J_s$ all variables x_{ij} with a corresponding Bernulli trial X_{ij}, such that $E[X_{ij}] = x_{ij}$. Since the rounding is done for each job j independently, the Bernulli trials X_{ij} of each constraint i, are independent and hence the load Ψ_i of each machine i in the rounded schedule is equal to the sum of a given positive value φ_i and the weighted sum of independent Bernulli trials $\Psi_i = \varphi_i + \sum_{j \in J'} p_{ij} X_{ij}$. Let $\xi = e^2 + \ln\left(\frac{2m}{\rho}\right)$ be an appropriate deviation ratio and let E_1 be the event:

$$\mathcal{E}_1 = \ \{ \text{ The makespan } \tau_3 \text{ of the rounded solution is } \tau_3 > \xi \cdot \tau_2 \} \ . \tag{21}$$

Proposition 3.1 *The probability of event $\mathcal{E}_1$ is at most $\rho/2$.*

Proof: The proof is a simple application of the Chernoff-like Bounds for Weighted Sums of Bernulli Trials of Theorem 2.1 and it is similar to the proof of Lem. 2.2 of Sec. 2. Since for each $i \ : \ E[\Psi_i] \leq \tau_2$:

$$\text{Prob}\{\Psi_i > \xi \cdot \tau_2\} \leq e^{\left(-\frac{\xi^2 \tau_2}{2(1+\frac{\xi}{3})}\right)} \leq \frac{\rho}{2 \cdot m} \ . \tag{22}$$

A sufficient bound on the probability of event $\mathcal{E}_1$ is the sum of the probabilities P_i:

$$\text{Prob}\{\mathcal{E}_1\} \leq \sum_i \text{Prob}\{\Psi_i\} \leq \sum_i \frac{\rho}{2 \cdot m} \leq \frac{\rho}{2} \ . \tag{23}$$

■

Let J_u be the set of unlucky jobs of the rounded schedule, that is J_u is the set of all jobs $j \in J_s$ that have been randomly assigned to a "bad" $p_{ij} > 1$.

Note 3.1 *Unlike the large jobs of J_ℓ that are statically defined for every instance of* SUM*, the notion of unlucky jobs is dynamical: Given a rounded schedule, a job is "unlucky" and belongs to J_s only if its processing time in the given rounded schedule is larger than 1 (and it does not belong to J_ℓ). Any job that does not belong to J_ℓ and for which at least one coefficient p_{ij} is larger than* 1 *can possibly be a unlucky job in a rounded schedule. Since the rounded schedules are obtained randomly from the fractional schedules, separate rounded schedules might have different sets of unlucky jobs.*

Filtering In order to achieve very tight bounds on the deviation of the makespan of the rounded schedule, <u>all unlucky jobs are simply removed from the rounded schedule</u>. The remaining schedule is called the filtered rounded schedule. For each machine i, let the random variable Ψ'_i be:

$$\Psi'_i = \varphi_i + \sum_{j \in J_s \ AND \ p_{ij} \leq 1} p_{ij} X_{ij} \ . \tag{24}$$

Ψ'_i corresponds to the load of the machine i due to all the remaining jobs, if unlucky jobs are excluded. Note that the random part of the random variables Ψ'_i is a weighted sum of Bernulli trials, where each weight is at most 1. Let τ_4 be the makespan of the filtered rounded schedule and let E_2 be the event:

$$\mathcal{E}_2 = \{ \text{ The makespan } \tau_4 \text{ of the filtered rounded schedule is } \tau_4 > (1+\epsilon_4)\cdot\tau_2 \}. \tag{25}$$

Proposition 3.2 *The probability of event $\mathcal{E}_2$ is at most $\rho/2$.*

Proof: For each machine i, its load in the filtered rounded schedule is the random variable Ψ'_i, and the probability that Ψ'_i exceeds $\tau_2 \cdot (1+\epsilon_4)$ can be bounded by the Chernoff-like bounds of Theorem 2.1.

$$\text{Prob}\{\Psi'_i > \tau_2 \cdot (1+\epsilon_4)\} < B(\tau_2, \epsilon_4) \leq \frac{\rho}{2\cdot m} \ . \tag{26}$$

The probability that at least one of the filtered loads Ψ^f_i exceeds $\tau_2 \cdot (1+\epsilon_4)$ is not larger than the sum of the individual bounds for each i:

$$\text{Prob}\{\tau_4 > \tau_2 \cdot (1+\epsilon_4)\} = \text{Prob}\{\exists\, i \ :\ \Psi'_i > \tau_2 \cdot (1+\epsilon_4)\} \leq \sum_i \left(\text{Prob}\{\Psi'_i > \tau_2 \cdot (1+\epsilon_4)\}\right) \leq \frac{\rho}{2} \ . \tag{27}$$

■

The following simple combinatorial argument will be used to handle the unlucky jobs. Similar arguments have been used in [1] and [12].

Lemma 3.1 *Let $d_1, d_2, \ldots, d_n$ be a sorted sequence of real numbers $d_1 \geq d_2 \geq \ldots \geq d_n > 0$ and let $D = \sum_{j=1}^{n} d_j$. Let p be a non-negative integer, and $\epsilon_3 > 0$ a constant. For $k = \left\lceil \frac{p}{\epsilon_3} \right\rceil$, any set S of reals $S = \{ d_i \mid i > k \}$ with $| S | \leq p$ satisfies:*

$$\sum_{d_i \in S} d_i \leq \epsilon_3 \cdot D \ . \tag{28}$$

Proof: The number of jobs n is assumed to be larger than the constant k, or else a schedule can be found by a brute force assignment. The real d_k satisfies $d_k \leq \frac{\epsilon_3}{p} \cdot D$. Since $\forall\, d_i \in S : i > k$ and the reals are sorted in decreasing order $\forall\, d_i \in S : d_i \leq d_k \leq \frac{\epsilon_3}{p} D$. Hence:

$$\sum_{d_i \in S} d_i \leq p \cdot \frac{\epsilon_3}{p} \cdot D \leq \epsilon_3 \cdot D \ . \tag{29}$$

■

Lemma 3.2 *For any set J_p of at most $p = m \cdot \xi \cdot \mu$ jobs that do not belong to J_ℓ ($J_p \bigcap J_\ell = \emptyset$), the sum of their minimum processing times d_j is at most*

$$\sum_{j \in J_p} d_j \leq \epsilon_3 \cdot \tau_2 \ . \tag{30}$$

Proof: Simple application of Lem. 3.1. ■

Let $\mathcal{E}_3$ be the event that the sum of the minimum processing times d_j of all unlucky jobs $j \in J_l$ is larger than $\epsilon_3 \cdot \tau_2$:

$$\mathcal{E}_3 = \{\sum_{j \in J_l} d_j > \epsilon_3 \cdot \tau_2\} \ . \tag{31}$$

Proposition 3.3 *The probability of event $\mathcal{E}_3$ is at most $\frac{\rho}{2}$.*

Proof: By Proposition 3.1 the probability that the non-filtered rounded schedule has makespan larger than $\xi \cdot \tau_2$ is at most $\frac{\rho}{2}$. Hence the probability that the total load on all machines in the rounded schedule exceeds $m \cdot \xi \cdot \mu$ is at most $\frac{\rho}{2}$. Since each unlucky job has processing time larger than 1, the probability that the total number of unlucky jobs in the rounded schedule is larger than $m \cdot \xi \cdot \mu$ is at most $\frac{\rho}{2}$. Lemma 3.2 proves, that for any set J_p of at most $p = m \cdot \xi \cdot \mu$ jobs such that $J_p \bigcap J_\ell = \emptyset$, the sum of their minimum processing times d_j is at most $\epsilon_3 \cdot \tau_2$. Hence if each unlucky job is assigned to the machine where its processing time is minimized, then even if in the worst case all unlucky jobs end up on the same machine the makespan of the schedule does not increase (additively) by more than $\epsilon_4 \cdot \tau_2$. ■

Final Schedule. The final schedule is obtained from the filtered schedule, by simply assigning every the unlucky job $j \in J_p$ to a machine i, where $p_{ij} = d_j$. Let τ_5 be the makespan of the final schedule.

Proposition 3.4 *Let $\mathcal{E}_4$ be the event: $\mathcal{E}_4 = \mathcal{E}_2 \bigcup \mathcal{E}_3$. Then:*

1. *The probability of event $\mathcal{E}_4$ is at most ρ, and*
2. IF *event* NOT*($\mathcal{E}_4$)* THEN *the makespan τ_5 of the final schedule is not larger than $(1+\epsilon) \cdot T$.*

Proof: 1. A sufficient bound on the probability of the event $\mathcal{E}_4 = \mathcal{E}_2 \bigcup \mathcal{E}_3$ that at least one of $\mathcal{E}_2$,$\mathcal{E}_3$ is true, is the sum of their individual probabilities:

$$\text{Prob}\{\mathcal{E}_2 \bigcup \mathcal{E}_3\} \le \text{Prob}\{\mathcal{E}_2\} + \text{Prob}\{\mathcal{E}_3\} \le \rho \ . \tag{32}$$

2. If event $\mathcal{E}_4$ is *NOT TRUE* then both events $\mathcal{E}_2$ and $\mathcal{E}_3$ are *NOT TRUE* and hence by Prop. 3.2 and 3.3 the makespan τ_5 of the final schedule is:

$$(\tau_5 \le \tau_4 + \epsilon_3 \cdot \tau_2) \cdot (1+\epsilon_5) \cdot (1+\epsilon_6) \ \Rightarrow \ \tau_5 \le (1+\epsilon) \cdot T \ . \tag{33}$$

■

The set of assignments Φ_f has cardinality $\mathcal{F} = \min\left(\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}, m^{\frac{2\log(1/\epsilon)}{\epsilon}}\right)$ and it can be calculated in $\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}$ time with the brute force enumeration techniques of Sec. 3.3. Algorithm *PDD* has to be executed once for each assignment $\bar{\varphi} \in \Phi_f$. The complexity of the

PDD algorithm is $O\left(n\left(\frac{m}{\epsilon}\right)^2 \ln\left(\frac{m}{\epsilon}\right)\right)$ on 1 processor or $O\left(\log(n)\left(\frac{m}{\epsilon}\right)^2 \ln\left(\frac{m}{\epsilon}\right)\right)$ on $\frac{n}{\log n}$ processors. The *XRR* procedure has to be executed only once in $O(mn)$ time on 1 processor or in $O(m \log n)$ time on $\frac{n}{\log n}$ processors. We have proven that given an instance of *SUM*, constants $\epsilon > 0$ and $\rho \in (0,1)$, and a makespan value T, *A-SUM* runs in fully $O(n)$–sequential time or in $O(\log n)$–time on $O(n/\log n)$–processors and with probability at least $1-\rho$, either produces a schedule of makespan at most $(1+\epsilon)T$ or decides that T is not a feasible makespan value. Hence:

Theorem 3.1 *Algorithm* A-SUM *is a* $O(n)$*–time* RRDP *for* SUM*. The parallel running time of* A-SUM *is* $O(\log n)$ *on a* $O(\frac{n}{\log n})$ *processor* EREW PRAM.

The algorithm *A-SUM* can be used within a binary search procedure (Fig. 1) to build an approximation scheme for the optimization version of problem *SUM.*

Corollary 3.1 *The binary search procedure of Fig. 1 is a* $O(n)$*–time* RFPTAS *for* SUM*. Its parallel running time is* $O(\log n)$ *on a* $O(\frac{n}{\log n})$ *processor* EREW PRAM.

Proof: Let $\mathcal{P}$ be an instance of *SUM* and let T^* be its optimal makespan. Given the overall approximation guarantee ϵ, let for $\epsilon_i : i = 1, \ldots, 6$ be appropriate values such that: $\prod_{i=1}^{6}(1+\epsilon_i) \leq (1+\epsilon)$ and $\epsilon' = \frac{\epsilon}{1+\epsilon_1}$. Let $L = \lceil \log(\frac{m}{\epsilon_1}) \rceil$. Given the bound on the probability of failure, let $\rho' = \frac{\rho}{L}$. An example of appropriate values is $\epsilon_1 = \epsilon_2 = \epsilon_3 = \epsilon_4 = \epsilon_5 = \epsilon_6 = \frac{\epsilon}{9}$ and $\epsilon' = \frac{\epsilon}{1+\epsilon_1}$. By Equation 5 the optimal makespan T^* is always bounded by $\frac{D}{m} \leq T^* \leq D$. Let $\epsilon_1 : \; 0 < \epsilon_1 < \epsilon$ be the precision of the binary search procedure and let $\epsilon' = \frac{1+\epsilon}{1+\epsilon_1}$ be the error ratio for algorithm *A-SUM.* The binary search procedure of Fig. 1 seeks for the minimum value $T_1 \in [\frac{D}{m}, D]$ that satisfies ϵ'–relaxed feasibility, that is when called with parameters T_1, ϵ', and ρ', algorithm *A-SUM* returns a schedule of makespan at most $(1+\epsilon')T$. Clearly $T_1 \leq T^*$ since all feasible makespan values satisfy relaxed feasibility. Then after L steps the value T_2 found by the binary search procedure satisfies $T_2 \leq (1+\epsilon_1) \cdot T_1 \leq (1+\epsilon_1) \cdot T^*$. The makespan T of the schedule found by *A-SUM* with parameters T_2, ϵ', ρ' is at most $T \leq (1+\epsilon') \cdot T_2 \leq (1+\epsilon') \cdot (1+\epsilon_1) \cdot T^* \leq (1+\epsilon) \cdot T^*$, with probability of success at least $1-\rho$. ■

Note 3.2 *It is possible to modify algorithm* A-SUM *so that it approximates directly the optimization version of* SUM *and to avoid the binary search procedure. Instead of assuming a feasible makespan value* T *the algorithm would simply filter out* p_{ij} *such that* $p_{ij} > D$*, since the optimal schedule has makespan at most* D*. By using a slightly larger* ξ *the algorithm would find an* $(1+\epsilon)$*–approximate schedule in one execution. However the complexity of the single execution is larger than the overall complexity of the* A-SUM *based algorithm.*

3.3 Approximate enumeration

The cardinality of set Φ of assignments of "large jobs" to the machines depends exponentially on $1/\epsilon_4$. Hence, even though the size of Φ is a constant, it is not polynomial on ϵ, since $\epsilon = \Theta(\epsilon_4)$. If algorithm *A-SUM* processes each assignment $\varphi \in \Phi$ separately then the complexity of the algorithm is not fully polynomial. It will be shown, that it is possible to consider only Φ_f, a substantially

Binary Search

Input:	D, m
Output:	A $(1 + \epsilon)$ approximate schedule
Pseudocode	$L = \frac{D}{m}$, $U = D$
	Loop
	$\quad T = \frac{L+U}{2}$
	$\quad$ Run Algorithm A_{SUM} for T.
	$\quad$ Is T a feasible makespan value ?
	$\quad\quad$ NO: L = T
	$\quad\quad$ YES: U = T
	Repeat $\lceil \log(\frac{m}{\epsilon_1}) \rceil$ times
	End

Figure 1: Binary Search

smaller subset of Φ, with size polynomial on ϵ. Considering only the assignments in Φ_f will admit algorithm *A-SUM* to be fully polynomial, at the cost of introducing an arbitrarily small constant error factor to the approximation guarantee of the final solution. The set Φ_f is obtained from Φ by applying $\mathcal{T}_1$, a technique of Horowitz and Sahni ([11]) in the way it has been used in [12], and $\mathcal{T}_2$, a geometric grouping technique.

Technique $\mathcal{T}_1$. Instead of considering every individual of the possible assignments $\varphi \in \Phi$ of the large jobs to the machines it is possible to define groups of similar assignments and to consider only one of the assignments in each group. The cost is an extra constant error factor to the approximation guarantee of the final solution. For every valid assignment φ the load φ_i on machine i is : $0 \leq \varphi_i \leq D$. The interval $[0, D]$ is partitioned into $N = \frac{mk}{\epsilon_5}$ sub-intervals each of size at most $\frac{D\epsilon_5}{mk}$. Given two assignments φ and χ of Φ, if for each machine i, their respective loads φ_i and χ_i on machine i are in the same sub-interval, then the assignments φ and χ are considered to the in the same group. From all assignments that belong to the same group, the algorithm considers only one arbitrary of them. Let Φ_1 be a set of the assignments that contains exactly one assignment of each group. The cardinality of Φ_1 is $\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}$. The set Φ_1 can be generated by a simple algorithm with sequential running time $\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}$. ([12, Section 2.1]).

At least one of the assignments $\varphi \in \Phi$ will correspond to an optimal schedule. Let φ^* be one optimal assignment. Then there will be a corresponding assignment $\bar{\varphi}^*$ in Φ_1, such that $\forall\ i,\ \bar{\varphi}^*_i \leq \varphi^*_i \cdot (1 + \epsilon_5)$. Hence enumerating the assignments in Φ_1 instead of Φ improves the complexity of algorithm *A-SUM* to fully polynomial at the cost of an extra error ratio of at most $(1 + \epsilon_5)$ to the final solution.

Technique $\mathcal{T}_2$. Technique $\mathcal{T}_2$ is a simple geometric grouping technique. The interval $[0, D]$ is partitioned into the geometrically increasing sub-intervals: $[0, \epsilon D], (\epsilon D, \epsilon(1+\epsilon)D], (\epsilon(1+\epsilon)D, \epsilon(1+\epsilon)^2 D], \ldots, (\epsilon(1+\epsilon)^L D, D]$, where $L = \lceil \frac{\log(1/\epsilon_6)}{\log(1+\epsilon_6)} \rceil$. Given two assignments φ and χ of Φ, if for each machine i, their respective loads φ_i and χ_i on machine i are in the same sub-interval, then the assignments φ and χ are considered to the in the same group. From all assignments that belong to the same group, the algorithm considers only one arbitrary of them. Technique $\mathcal{T}_2$ partitions the assignments $\varphi \in \Phi$ into at most m^L groups. Let Φ_2 be the set of all representative assignments of Φ_1.

Lemma 3.3 *For every $x \in [0,1]$: $\frac{1}{2}x \leq \ln(1+x)$.*

Proof: Let $f(x) = \ln(1+x) - \frac{1}{2}x$. We want to show that $f(x) \geq 0$ for $x \in [0,1]$. Now $f(0) = 0$ and $f'(x) = \frac{1}{1+x} - \frac{1}{2}$. Since $f'(x) \geq 0$ for $x \in [0,1]$, $f(x) \geq f(0) = 0$ for $x \in [0,1]$. ■

The cardinality of Φ_2 is $O(m^L) = O(m^{\frac{\log(1/\epsilon_6)}{\log(1+\epsilon_6)}}) = O(m^{\frac{2\log(1/\epsilon_6)}{\epsilon_6}})$. The last equality is due to Lemma 3.3.
In algorithm *A-SUM* technique $\mathcal{T}_2$ is applied on top of technique $\mathcal{T}_1$, that is for all assignments in Φ_1 that belong to the same geometric group of technique $\mathcal{T}_2$ only one (arbitrary) of them is in Φ_f. For each assignment $\bar{\varphi} \in \Phi_1$ the algorithm proceeds to the main iteration only if no other assignment of the same class for Φ_2 has been processed. The number of iterations of the main part of the algorithm is now $O(\min(|\Phi_1|, |\Phi_2|)$. The cost is the extra factor $(1+\epsilon_5) \cdot (1+\epsilon_6)$ in the approximation guarantee of the final solution.

Note 3.3 *Due to technique $\mathcal{T}_2$ the number of assignments that are processed is* polynomial on m. *However $\mathcal{T}_2$ depends on $\mathcal{T}_1$ since all assignments of Φ_2 are generated from the assignments of Φ_1. Technique $\mathcal{T}_1$ depends exponentially on the number of machines m and it is the only component of the whole approximation scheme that is not polynomial on m. Hence a way to directly generate the assignments of Φ_2 would turn* A-SUM *into a polynomial time algorithm for any numbers n of jobs and m of machines.*

4 Makespan and Cost

The bicriteria problem *SUMC* generalizes the standard *SUM* problem, in that processing a job j on machine i incurs a cost of c_{ij} and hence there are two optimization criteria in *SUMC*, the makespan and the cost. As in the previous sections, the number of machines m is assumed to be a constant. We show algorithm *A-SUMC*, a fully linear time *RRDP* for *SUMC*. Given an instance of SUMC and values T and C, an ϵ–relaxed decision procedure for SUMC returns a schedule of cost at most $(1+\epsilon)C$ and makespan at most $(1+\epsilon)T$, or decides that there is no schedule of cost at most C and makespan at most T. The probability of failure is at most the given constant ρ.

4.1 Algorithm *A-SUMC*

The structure of Algorithm *A-SUMC* is almost identical to that of algorithm *A-SUM*, with a small number of straightforward adaptations. The definition of the measures d_j and D is extended to cover the bicriteria nature of *A-SUMC*. A constant number of jobs, the large jobs, are selected. For each possible assignment of the large jobs, a fractional schedule is calculated with algorithm

PDD. The best fractional solution is rounded randomly to an integer schedule. All unlucky jobs of the rounded schedule are removed and then assigned, each on the machine where its measure d_j is minimum. The result is a $(1+\epsilon)$–approximate schedule, if the problem is feasible and if the randomized algorithm did not fail.

Input: An instance of *SUMC*, the constants $\epsilon : 0 < \epsilon \leq 1$, and $\rho :\ 0 < \rho < 1$, and the values T for makespan and C for cost.

Output: With probability of success at least $(1-\rho)$, a schedule of makespan at most $(1+\epsilon)T$ and cost at most $(1+\epsilon)C$ or the problem is infeasible for makespan T and cost C.

Step 0: *Initializations.*
Let $\epsilon_1 = \epsilon_2 = \epsilon_3 = \epsilon_4 = \epsilon_5 = \epsilon_6 = \frac{\epsilon}{9} = \Theta(\epsilon)$. $\forall j,\ d_j = \min_i\{p_{ij} + c_{ij}\}$, $D = \sum_j d_j$,
$\mu = \frac{3\ln\frac{2m+2}{\rho}}{(\epsilon_4)^2}$, $\xi = e^2 + \ln\left(\frac{2m+2}{\rho}\right)$, and $k = \left\lceil \xi \cdot \mu \cdot (m+1) \cdot \frac{m+1}{\epsilon_3} \right\rceil$.

Step 1: *Normalization.* The processing times p_{ij} are scaled by $\frac{\mu}{T}$ and the costs c_{ij} by $\frac{\mu}{C}$. Now the problem is to decide if there is a schedule of makespan and cost bounded by μ.

Step 2: *Simple Filtering.*
$\forall i, j$: *IF* $(p_{ij} > T$ *OR* $c_{ij} > C)$ *THEN* $x_{ij} = 0$

Step 3: *Large jobs.*
Let $J_\ell = \{\ j\ ;|\ d_j$ belongs to the k largest $d_j\}$ be the set of large jobs and let Φ be the set of all possible assignments of the large jobs to the machines. Let Φ_f be an appropriate subset of Φ.

Step 4: *Best Fractional Schedule* x_{ij}.
$\forall$ assignment $\varphi \in \Phi_f$ do

1. Formulate the corresponding scheduling problem as an integer program *ILP-SUMC*(φ).
2. Relax *ILP-SUMC*(φ) to the linear program *LP-SUMC*(φ).
3. Find the approximate fractional schedule with algorithm *PDD*.

Among all fractional schedules select the one of minimum makespan and cost.

Step 5: *Filtered Randomized Rounding.*

1. Round the best fractional schedule with *XRR*.
2. Filter : If a job j has been randomly assigned to a $p_{ij} > 1$ or a $c_{ij} > 1$ *THEN* job j is called "unlucky" and it is removed from the schedule.
3. $\forall$ unlucky jobs j, assign job j to machine $i = \text{argmin}_i\{p_{ij} + c_{ij}\}$

4.2 Analysis of algorithm *A-SUMC*

The input to algorithm *A-SUMC* for *SUMC*, is an instance of *SUMC*, the values T for the makespan and C for the cost, the constant approximation ratio $\epsilon > 0$ and the maximum probability of failure $\rho \,:\, 0 < \rho < 1$.

Normalization. Let μ be the marginal mean value:

$$\mu = \frac{3 \cdot \ln\left(\frac{2m+2}{\rho}\right)}{(\epsilon_4)^2} \quad . \tag{34}$$

By scaling the processing times p_{ij} by $\frac{\mu}{T}$ and the costs c_{ij} by $\frac{\mu}{C}$ the problem becomes to decide if there is a schedule of makespan and cost bounded by μ.

Simple Filtering. As in Sec. 3, a simple filtering procedure sets for any pair (i, j) such that $p_{ij} > \mu$ or $c_{ij} > \mu$ the corresponding $x_{ij} = 0$. Since μ is assumed to be a feasible bound for the problem, the simple filtering process does not modify its feasibility. Let

$$\xi = e^2 + \ln\left(\frac{2m+2}{\rho}\right) \quad , \tag{35}$$

$$p = \mu \cdot \xi \cdot (m+1) \;\text{ , and} \tag{36}$$

$$k = \left\lceil (\mu \cdot \xi \cdot (m+1)) \cdot \left(\frac{m+1}{\epsilon_3}\right) \right\rceil \quad . \tag{37}$$

The measure d_j for each job j is now defined in a way to cover both the processing times p_{ij} and the costs c_{ij}. The sum of all d_j is defined as before to be D.

$$\forall \text{ job } j,\; d_j = \min_i \{p_{ij} + c_{ij}\} \text{ and } D = \sum_j d_j \quad . \tag{38}$$

Large Jobs. Let the k jobs corresponding to the k largest d_j be the set J_ℓ of the "large jobs":

$$J_\ell = \{j \mid d_j \text{ belongs to the } k \text{ largest } d_j\} \quad . \tag{39}$$

Enumeration. Let Φ denote the set of all possible assignments t of the large jobs $j \in J_\ell$ to the machines. The cardinality of Φ is m^k, a constant. The algorithm can examine separately each of the assignments $\varphi \in \Phi$ in $m^{O(\frac{(m+1)^2 \log^2(\frac{2m+2}{\rho})}{\epsilon^3})}$ steps.

Approximate Enumeration. As in Sec. 3 the set of assignments considered can be substantially reduced from Φ to $\Phi_f \subseteq \Phi$ with techniques of Sec. 3.3. The cost is an extra arbitrarily small error factor $(1+\epsilon_5)\cdot(1+\epsilon_)$ to the approximation guarantee for the final solution. As in the analysis of algorithm *A-SUM*, the rest of the analysis of *A-SUMC* is done with the assumption that all possible assignments in Φ are examined by the algorithm. This assumption is relaxed in the final Equation 48, by introducing the extra factor $(1+\epsilon_5)\cdot(1+\epsilon_6)$ to the approximation guarantee.

Approximate Fractional Schedule Let φ^* be an assignment of the large jobs as they appear in an optimal schedule. Given the assignment φ^*, the problem of assigning the remaining jobs in an optimal way (to minimize the maximum of makespan and cost) can be formulated as the following integer linear program $ILP - SUMC(\varphi^*)$:

$$
\begin{array}{lll}
min & \tau & \\
s.t. & & \\
 & \varphi_i^* + \sum_{j\in[n]-J_\ell} p_{ij}x_{ij} \le \tau & (i=1,\ldots,m) \\
 & \varphi_c^* + \sum_{j\in[n]-J_\ell, i\in[m])} c_{ij}x_{ij} \le \tau & \\
 & \sum_{i=1}^m x_{ij} = 1 & (j \in [n]-J_\ell) \\
 & x_{ij} \in \{0,1\} & (i=1,\ldots,m;\ j\in[n]-J_\ell)
\end{array}
$$

Let τ^* be the optimal objective value of *ILP-SUMC*(φ^*). Since the value μ is assumed to be a feasible bound on the makespan:

$$\tau^* \le \mu = T = C \ . \tag{40}$$

Let *LP-SUMC*(φ^*) be the linear program relaxation of *ILP-SUMC*(φ^*) obtained by relaxing the integrality constraints on the variables x_{ij} to $\forall i,j:\ x_{ij}\ge 0$. The linear program *LP-SUMC*(φ^*) has a block-angular structure and can be approximated with the logarithmic-potential based PDD algorithm of [6] as *LP-SUM*(t^*) in Sec. 3. Given the constant ϵ_2, the PDD algorithm will produce a $(1+\epsilon_2)$–approximate fractional schedule. Let x_{ij} and τ_1 be the output produced by PDD. Then:

$$
\begin{array}{ll}
\varphi_i^* + \sum_{j\in[n]-J_\ell} p_{ij}x_{ij} \le \tau_1 & (i=1,\ldots,m) \\
\varphi_c^* + \sum_{j\in[n]-J_\ell, i\in[m])} c_{ij}x_{ij} \le \tau_1 & \\
\sum_{i=1}^m x_{ij} = 1 & (j \in [n]-J_\ell) \\
x_{ij} \in [0,1] & (i=1,\ldots,m;\ j\in[n]-J_\ell)
\end{array}
$$

By the approximation guarantee of *PDD* and Equ. 40:

$$\tau_1 \le \tau^* \cdot (1+\epsilon_2) \le \mu \cdot (1+\epsilon_2) \ . \tag{41}$$

For each assignment $\varphi \in \Phi$ the PDD algorithms finds an approximate fractional solution to *LP-SUMC*(φ). At the end the algorithm selects the among all fractional solutions the one with the best objective value τ_2.

$$\tau_2 = \min_{\varphi\in\Phi} \{\ \tau_\varphi \ \mid\ \tau_\varphi = \epsilon_2 - \text{approximate solution to } \textit{LP-SUMC}(\varphi) \text{ found with } \textit{PDD}\} \ . \tag{42}$$

Hence:

$$\tau_2 \le \tau_1 \le \mu\cdot(1+\epsilon_2) \ . \tag{43}$$

Rounding. As in Sec. 3 the best fractional solution found is rounded to an approximate schedule, with Filtered Randomized Rounding (*FRR*). First the fractional schedule is rounded to an integer schedule with standard *XRR*. Let τ_3 be the maximum of the makespan and the cost of the rounded schedule.

Let $\xi = e^2 + \ln\left(\frac{2m+2}{\rho}\right)$. Let $\mathcal{E}_1$ be the event:

$$\mathcal{E}_1 = \ \{\text{ In the rounded solution } \tau_3 > \xi\cdot\mu\}. \ . \tag{44}$$

Proposition 4.1 *The probability of event $\mathcal{E}_1$ is at most $\rho/2$.*

Proof: Similar to Proposition 3.1. ■

For each i, j let the processing time p_{ij} be called "bad" if $p_{ij} > 1$ and the cost c_{ij} be called "bad" if c_{ij}. In the rounded schedule a job j is called unlucky, if it does not belong to the large jobs J_ℓ and if it has been randomly assigned to a machine i such that p_{ij} is bad or c_{ij} is bad, that is $x_{ij} = 1$ with $p_{ij} > 1$ or $c_{ij} > 1$. Let J_u be the set of unlucky jobs in the rounded schedule. All the unlucky jobs are removed from the rounded schedule. The remaining schedule is called the filtered rounded schedule. In the filtered rounded schedule let $\tau_4 = \max\{\text{makespan}, \text{cost}\}$. Let $\mathcal{E}_2$ be the event:

$$\mathcal{E}_2 = \ \{ \text{ In the filtered rounded schedule } \tau_4 > (1+\epsilon_4) \cdot \tau_2 \}. \tag{45}$$

Proposition 4.2 *The probability of event $\mathcal{E}_2$ is at most $\rho/2$.*

Proof: Similar to Proposition 3.2. ■

Lemma 4.1 *For any set of at most p jobs that do not belong to J_ℓ, the sum of their d_j is at most $\epsilon_3 \cdot \tau_2$.*

Proof: Simple application of Lem. 3.1. ■

Let $\mathcal{E}_3$ be the following event concerning the sum of the d_j of all unlucky jobs $j \in J_u$:

$$\mathcal{E}_3 = \{ \sum_{j \in J_u} d_j > \epsilon_3 \cdot \tau_2 \} \ . \tag{46}$$

Proposition 4.3 *The probability of event $\mathcal{E}_3$ is at most $\frac{\rho}{2}$.*

Proof: Similar to Prop. 3.3. ■

Final Schedule. The final schedule is obtained from the filtered schedule, by simply assigning every unlucky job $j \in J_p$ to a machine i, where $p_{ij} = d_j$. Let τ_5 be the maximum of the makespan and the cost of the final schedule.

Proposition 4.4 *Let $\mathcal{E}_4$ be the event: $\mathcal{E}_4 = \mathcal{E}_2 \bigcup \mathcal{E}_3$. Then:*

1. *The probability of event $\mathcal{E}_4$ is at most ρ, and*
2. IF *event* NOT*($\mathcal{E}_4$)* THEN *in the final schedule τ_5 is not larger than $(1+\epsilon) \cdot T$.*

Proof: 1. A sufficient bound on the probability of the event $\mathcal{E}_4 = \mathcal{E}_2 \bigcup \mathcal{E}_3$ that at least one of $\mathcal{E}_2$, $\mathcal{E}_3$ is true, is the sum of their individual probabilities:

$$\text{Prob}\{\mathcal{E}_2 \bigcup \mathcal{E}_3\} \leq \text{Prob}\{\mathcal{E}_2\} + \text{Prob}\{\mathcal{E}_3\} \leq \rho \ . \tag{47}$$

2. If event $\mathcal{E}_4$ is *NOT TRUE* then both events $\mathcal{E}_2$ and $\mathcal{E}_3$ are *NOT TRUE* and then by Prop. 4.2 and 4.3 the value τ_5 of the final schedule is:

$$(\tau_5 \leq \tau_4 + \epsilon_3 \cdot \tau_2) \cdot (1+\epsilon_5) \cdot (1+\epsilon_6) \ \Rightarrow \ \tau_5 \leq (1+\epsilon) \cdot \mu \ . \tag{48}$$

■

The set of assignments Φ_f has cardinality $\mathcal{F} = \min\left(\left(\frac{(m+1)\log((2m+2)/\rho)}{\epsilon}\right)^{O(m)}, (m+1)^{\frac{2\log(1/\epsilon)}{\epsilon}}\right)$ and it can be calculated in $\left(\frac{(m+1)\log((2m+2)/\rho)}{\epsilon}\right)^{O(m)}$ time with the brute force enumeration techniques of Sec. 3.3. Algorithm *PDD* has to be executed once for each assignment $\bar{\varphi} \in \Phi_f$. The complexity of the *PDD* algorithm is $O\left(n\left(\frac{m}{\epsilon}\right)^2 \ln\left(\frac{m}{\epsilon}\right)\right)$ on 1 processor or $O\left(\log(n)\left(\frac{m}{\epsilon}\right)^2 \ln\left(\frac{m}{\epsilon}\right)\right)$ on $\frac{n}{\log n}$ processors. The *XRR* procedure has to be executed only once in $O(mn)$ time on 1 processor or in $O(m \log n)$ time on $\frac{n}{\log n}$ processors. We have proven that given an instance of *SUMC*, constants $\epsilon > 0$ and $\rho \in (0,1)$, and values values T and C for makespan and cost respectively, *A-SUMC* runs in fully $O(n)$–sequential time or in $O(\log n)$–time on $O(n/\log n)$–processors and, with probability at least $1-\rho$, either produces a schedule of makespan at most $(1+\epsilon)T$ and cost $(1+\epsilon)C$, or decides that there is no schedule of makespan at most T and cost at most C. Hence:

Theorem 4.1 *Algorithm* A-SUMC *is a $O(n)$–time* RRDP *for* SUMC. *The parallel running time of* A-SUMC *is $O(\log n)$ on a $O(\frac{n}{\log n})$ processor* EREW PRAM.

5 Optimizations of SUMC

The *SUMC* problem has two separate objectives, the makespan and the cost, and this fact gives rise to more than one optimization problems for *SUMC*. Two natural cases are *SUMCoptT* and *SUMCoptC*, which are obtained by specifying an upper bound on the one objective and then optimizing the second objective under this condition. In this way, *SUMCoptT* is the problem of finding for a given instance of *SUMC* and a specified cost value C, the schedule of minimum makespan for cost at most C. Similarly *SUMCoptC* is the problem of finding for given *SUMC* and makespan T a schedule of minimum cost C and makespan at most T. A third optimization problem is *SUMCoptTC*, which optimizes a linear combination of both criteria, makespan and cost, for example $d \cdot T + C$, for $d > 0$. This problem is discussed in [23] for an unrestricted number of machines. For a fixed number of machines, Jansen and Porkolab present in [12] a simple approximation scheme for *SUMCoptTC* based on a relaxed decision procedure for *SUMC*. Algorithm *A-SUMC* can be used within the algorithm of [12].

In this section, we show how an ϵ–relaxed decision procedure for *SUMC* can be used to build approximation schemes for the ϵ-relaxed versions of *SUMCoptT* and *SUMCoptC*. Given an instance of SUMCoptT and the bound on the cost C, the ϵ–relaxed SUMCoptT problem is to find a schedule of cost at most $(1+\epsilon)C$ and makespan at most $(1+\epsilon)T$, if T is the optimal makespan of *SUMCoptT* if the cost is restricted to be at most C. In the same way, given an instance of *SUMCoptC* and the bound on the makespan T, the ϵ–relaxed SUMCoptC problem is to find a schedule of makespan at most $(1+\epsilon)T$ and cost at most $(1+\epsilon)C$, if C is the optimal cost of *SUMCoptC* when the makespan is restricted to be at most T.

Proposition 5.1 *For any feasible solution to SUMC, the two objectives of SUMC, makespan and cost, have values that are always within a linear factor of one of the weights in the original description of the problem instance.*

Proof: Let p_{max} be the maximum processing time that appears in an optimal solution to SUMC. Then the makespan of the optimal schedule is at least p_{max} and at most $n \cdot p_{max}$. There are at most $m \cdot n$ different possibilities for p_{max}, since p_{max} has to be one of the specified p_{ij}. In the same way let c_{max} be the maximum cost that appears in an optimal solution to SUMC. Then the cost of the optimal schedule is at least c_{max} and at most $n \cdot c_{max}$. There are at most $m \cdot n$ different possibilities for c_{max}, since c_{max} has to be one of the specified c_{ij}. ■

Definition 5.1 *We say that a combinatorial optimization problem has the* poly–bottleneck[3] *property, if its optimal objective value is always within a polynomial factor of one of its input elements (weights). Unweighted problems are assumed to have elements of weight* 1.

Fact 5.1 SUMC *has the* poly–bottleneck *property.*

The following Lemma uses algorithm *A-SUMC* and the *poly–bottleneck* property of *SUMCoptT* to build an approximation algorithm for it (*SUMCoptT*). In the same way an approximation algorithm for *SUMCoptC* can be build. Furthermore the technique can be generalized to any problem that has the *poly–bottleneck* property, if a (relaxed) decision procedure is given for the problem.

Lemma 5.1 *Let P be an instance of the* SUMCoptT *problem for given cost C, approximation ratio $\epsilon > 0$ and let $\rho :\ 0 < \rho < 1$ be the maximum probability of failure. Let N be the size of the problem. There is a simple two-level binary search procedure that with probability at least $(1-\rho)$ gives a $(1+\epsilon)$ approximate solution for P in $O(\log N)$ steps. The binary search procedure calls algorithm* A-SUMC *once at each step.*

Proof: The following 2-level binary search procedure achieves the result claimed in the Lemma. The procedure seeks for the minimum makespan value T, such algorithm *A-SUMC* returns a $(1+\epsilon)$–approximate schedule of the corresponding problem *SUMC* with parameters T and C.

Step 1: *Weights.* Let W be the set of numbers that contains for each pair (i, j), the values p_{ij} and $n \cdot p_{ij}$. The cardinality of W is $w = |W| \leq 2 \cdot m \cdot n$.

Step 2: *Sorting.* Sort the elements of W. Let the sorted list be $w_1 \geq w_2 \geq \ldots \geq w_w$.

Step 3: *Indexed binary search.*

1. *LET* $l = 1,\ u = w$
2. $x = \frac{l+u}{2}$
3. *IF SUMC* is feasible for cost C and makespan w_x
4. *THEN* $u = x$ *ELSE* $l = x$;

[3] The term "bottleneck" has been used by Hochbaum and Schmoys in [8] for a class of graph optimization problems. An important property of the bottleneck problems of [8], is that the value of the optimal solution is always one of the (edge) weights in the original specification of the instance of the problem.

5. Repeat from step 2 for at most $\lceil log(2mn) \rceil$ times.

The output x is such that : $w_x \leq T^* < w_{x+1}$.

Step 4: *Standard binary search.*

1. *LET* $L = w_x$, $U = w_{x+1}$
2. $T = \frac{L+U}{2}$
3. *IF SUMC* is feasible for cost C and makespan w_x
4. *THEN* $U = T$ *ELSE* $L = T$;
5. Repeat from step 2 for at most $\lceil log(mn/\epsilon) \rceil$ times.

The sorting procedure of step 2 needs $O(n \log n)$ time. The binary search of steps 3 and 4 executes $O(\log(n/\epsilon))$ loops. The main cost of each loop is the execution of algorithm *SUMC*. Given that the probability of failure of the whole procedure is bounded by ρ, each individual execution of *SUMC* is done with probability of failure at most $\frac{\rho}{\lceil log(2mn) \rceil + \lceil log(mn/\epsilon) \rceil}$. Under this condition the complexity of *SUMC* is $O(n \log \log n)$. Hence the complexity of the overall approximation scheme for *SUMCoptC* is $O(n \log n \log \log n)$. The asymptotic performance of the algorithm can slightly be improved to $O(n \log n)$ by using the deterministic *RDP* for *SUMC* of [12] instead of algorithm *A-SUMC*. ■

References

[1] A.K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis. Scheduling independent multiprocessor tasks. In *5th Annual European Symposium on Algorithms*, pages 1–12, 1997.

[2] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: Towards a realistic model of parallel computation. In *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993.

[3] J. Diaz, M. Serna, P. Spirakis, and J. Toran. *Paradigms for Fast Parallel Approximability.* Cambridge University Press, 1997.

[4] P.S. Efraimidis and P.G. Spirakis. Very fast, sequential and parallel, approximations to hard combinatorial optimization problems. Technical Report TR99.06.01, Computer Technology Institute, June 1999.
(http://students.ceid.upatras.gr/∼efraimid/index.html).

[5] M. Grigoriadis and L. Khanchiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM J. Optimization*, 4(1):86–107, 1994.

[6] M. Grigoriadis and L. Khanchiyan. Coordination complexity of parallel price-directive decomposition. *Mathematics of Operations Research*, pages 317–327, 1996.

[7] M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, editors. *Probabilistic Methods for Algorithmic Discrete Mathematics.* Springer, 1998.

[8] D. Hochbaum and D. B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 3(33):533–550, July 1986.

[9] D. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 3(1):144–162, January 1987.

[10] D.S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company, 1997.

[11] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23:317–327, 1976.

[12] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *ACM Symposium on Theory of Computing*, pages 408–417, 1999.

[13] H. Karloff. *Linear Programming.* Birkhäuser, 1991.

[14] Y. Kopidakis, D. Fayard, and V. Zissimopoulos. Linear time approximation schemes for parallel processor scheduling. In *8th IEEE Symposium on Parallel and Distributed Processing*, pages 482–485, 1996.

[15] J.K. Lenstra, D.B. Shmoys, and È. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[16] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[17] S. Plotkin, D. Shmoys, and È. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[18] P. Raghavan. *Randomized Rounding and Discrete Ham-Sandwich Theorems: Provably Good Algorithms for Routing and Packing Problems.* PhD thesis, Computer Science Division, UC Berkeley, 1986.

[19] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.

[20] P. Raghavan and C.D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[21] A. Schrijver, editor. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.

[22] M. Serna and F. Xhafa. Approximating scheduling problems in parallel. In *EuroPAR 97*, 1997.

[23] D. Shmoys and È. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, A62:461–474, 1993.

[24] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

APPENDIX

A Proof of Theorem 2.1

This section shows the proof of Theorem 2.1 for Chernoff-like bounds.

Theorem A.1 *(Theorem 2.1). Let $\lambda \geq 0$ be a positive real number and let $\alpha_1, \alpha_2, \ldots, \alpha_r$ be reals in (0,1]. Let $X_1, X_2, \ldots, X_r$ be independent Bernulli trials with $E[X_j] = p_j$. Let $\Psi = \lambda + \sum_{j=1}^{r} a_j X_j$. Then $E[\Psi] = \lambda + \sum_{j=1}^{r} a_j p_j = S$. Let $\delta > 0$, and $T \geq S = E[\Psi] > 0$. Then*

$$Prob[\Psi > (1+\delta)T] < e^{\left(-\frac{(\delta)^2 T}{2(1+\frac{\delta}{3})}\right)} \text{ and for } \delta < 1 \ : \ e^{\left(-\frac{(\delta)^2 T}{2(1+\frac{\delta}{3})}\right)} \leq e^{\left(-\frac{(\delta)^2 T}{3}\right)} . \qquad (49)$$

Theorem 2.1 is a simple generalization of the following Theorem of Raghavan and Spencer on Chernoff-like bounds on the tail of the distribution of the Weighted Sum of Bernulli Trials ([19], [18]).

Theorem A.2 *([19, Theorem 1]). Let $\alpha_1, \alpha_2, \ldots, \alpha_r$ be real numbers in (0,1]. Let $X_1, X_2, \ldots, X_r$ be independent Bernulli trials with $E[X_j] = p_j$. Let $\Psi = \sum_{j=1}^{r} a_j X_j$. Then $E[\Psi] = \sum_{j=1}^{r} a_j p_j = S$. Let $\delta > 0$, and $S = E[\Psi] > 0$. Then*

$$Prob[\Psi > (1+\delta)S] < \left[\frac{e^{(\delta)}}{(1+\delta)^{(1+\delta)}}\right]^{S} \quad . \tag{50}$$

The following technical Lemma will used to replace the right hand side of Equ. 50 with an easier to handle expression.

Lemma A.1 *([7, Page 200, Lemma 2.4]). For all $x \geq 0$,*

$$(1+x)\ln(1+x) - \geq 3x^2/(6+2x) \ . \tag{51}$$

Proof: Let $f_1(x) = (6 + 8x + 2x^2)\ln(1+x) - 6x - 5x^2$. We want to show that $f_1(x) \geq 0$ for all $x] \geq 0$. Now $f_1(0) = 0$, and $f_1'(x) = 4f_2(x)$ where $f_2(x) = (2+x)\ln(1+x) - 2x$. It suffices to show that $f_2(x) \geq 0$ for all $x \geq 0$. Now $f_2(0) = 0$, and $f_2'(x) = (1+x)^{-1} + \ln(1+x) - 1$. Now $f_2''(0) = 0$, so it suffices to show that $f_2''(0) \geq 0$ for all $x \geq 0$. But $f_2''(x) = x(1+x)^{-2} \geq 0$, and so we are done. ■

The following Lemma is obtained from Lemma A.1 and Theorem A.2:

Lemma A.2 *For $\delta > 0$ and $S \geq 0$*

$$\left[\frac{e^{(\delta)}}{(1+\delta)^{(1+\delta)}}\right]^{S} \leq e^{\left(-\frac{(\delta)^2 S}{2(1+\frac{\delta}{3})}\right)} \quad , \textit{ and for } \delta < 1 : e^{\left(-\frac{(\delta)^2 S}{2(1+\frac{\delta}{3})}\right)} \leq e^{\left(-\frac{(\delta)^2 S}{3}\right)} \quad . \tag{52}$$

Proof: Let $B(S,\delta) = e^{\left(-\frac{(\delta)^2 S}{2(1+\frac{\delta}{3})}\right)}$. Let $\lambda \geq 0$ be a positive real number and let $\alpha_1, \alpha_2, \ldots, \alpha_r$ be reals in (0,1]. Let $X_1, X_2, \ldots, X_r$ be independent Bernulli trials with $E[X_j] = p_j$. Let $\Psi = \lambda + \sum_{j=1}^{r} a_j X_j$. Then $E[\Psi] = \lambda + \sum_{j=1}^{r} a_j p_j = S$. Let $\delta > 0$, and $T \geq S = E[\Psi] > 0$.

Let $\Psi' = \sum_{j=1}^{r} a_j p_j$ and $E[\Psi'] = \sum_{j=1}^{r} a_j p_j = S'$. Then

$$\begin{aligned} \text{Prob}\{\Psi > (1+\delta)T\} = {} & \text{Prob}\{\lambda + \Psi' > (1+\delta)T\} = \\ \text{Prob}\{\Psi' > (1+\delta)T - \lambda\} \leq {} & \text{Prob}\{\Psi' > T - \lambda + T\cdot\delta\} = \\ \text{Prob}\{\Psi' > S'(1 + \frac{T-\lambda-S'}{S'} + \frac{T}{S'}\cdot\delta)\} < {} & \quad B(S', \tfrac{T-\lambda-S'}{S'} + \tfrac{T}{S'}\cdot\delta) \ . \end{aligned}$$

The last step is to show that

$$B(S', \frac{T-\lambda-S'}{S'} + \frac{T}{S'}\cdot\delta)\cdot\delta) \leq B(T,\delta) \ , \tag{53}$$

which can be done be using the definition of $B(S,\delta)$ and the fact that $S' + \lambda = S \leq T$. ■

B Approximating the Linear Programs with *PDD*

The linear programs that occur in algorithms A-$SUM\beta$, A-SUM and A-$SUMC$ respectively, have the block-angular structure of the following linear program $\mathcal{L}$.

$$\begin{array}{lll} & \text{Linear Program } \mathcal{L} & \\ min & \lambda & \\ s.t. & & \\ & \sum_{j=1}^{n} p_{ij} x_{ij} \le \lambda & (i = 1, \ldots, m) \\ & \sum_{i=1}^{m} x_{ij} = 1 & (j = 1, \ldots, n) \\ & Z \ge 0,\ x_{ij} \ge 0 & (i = 1, \ldots, m;\ j = 1, \ldots, n) \end{array}$$

An important property of $\mathcal{L}$ is that the problem variables are grouped into n independent m-dimensional simplices (blocks) and that it has a constant number of positive packing constraints that can be considered as the coupling constraints. The objective is to compute a block-feasible solution that uses a scalar multiple of the given m vector of resources. Since $\mathcal{L}$ is a linear program it can be optimally solved with any of the known polynomial time algorithms for linear programming ([13], [21]). If however, an $(1+\epsilon)$–approximate solution is good enough for the application (and this is the case for the applications of this work), then there are very fast algorithms for linear programs like $\mathcal{L}$ ([5],[17],[6]). The most efficient for the needs of this work is the logarithmic-potential based PDD algorithm of Grigoriadis and Khanchiyan, which approximates the LP relaxation of the problem within $(1+\epsilon_1)$ in $O(n)$ time. Furthermore it admits a simple optimal work parallelization of $O(\log n)$–running time.

Claim B.1 *(Claim 2.1). The linear program* LP-SUMβ *can be approximated within any constant ratio ϵ in $(O(n))$–sequential time and in $O(\log n)$–time on a $O(n/\log n)$–processor EREW PRAM.*

Proof: According to Theorem 3 of [6], the PDD algorithm will find a $(1+\epsilon_1)$-approximate solution for LP-SUMb in $O(m(\epsilon_1^{-2}\ln\epsilon_1^{-1}+\ln m))$ iterations. Each iteration requires $O(m\ln\ln(m/\epsilon_1))$ operations, or $O(\ln m\ln\ln(m/\epsilon_1))$ operations in parallel on a $m/\ln m$ - processor EREW PRAM. Each iteration also requires n parallel unrestricted block optimizations performed to a relative accuracy of $O(\epsilon_1)$. At each iteration the algorithm calculates m sums of n numbers for estimating the current load on each coupling constraint. The addition can be done in $O(mn)$ time on 1 processor or in $O(m\log n)$ time on $O(\frac{n}{\log n})$ processors. For the block problems in LP, there are simple block solvers that find in $O(m)$ time the optimal solution to the block problem, even though a ϵ_1-approximate solution would be sufficient. Hence the algorithm runs in $O(\log n)$ time on $O(n/\log n)$ processors and produces a $(1+\epsilon_2)$ approximate solution to *LP-SUMβ*. ∎

C Definitions and notation

This Section summarizes the terminology and the notation used in the paper.

C.1 Definitions

Definition C.1 Polynomial Time Approximation Scheme - PTAS*: An algorithm that accepts as input a problem instance and a constant* $\epsilon > 0$*, runs in time polynomial on the instance size* N*, and produces as output a* $(1+\epsilon)$*–approximate solution.*

Definition C.2 Fully Polynomial Time Approximation Scheme - FPTAS*: A* PTAS *whose running time is polynomial on both the instance size and the constant* $\epsilon > 0$*.*

Definition C.3 Randomized PTAS - RPTAS*: A randomized algorithm that accepts as input a problem instance and a constant* $\epsilon > 0$*, runs in time polynomial on the size* N *of the instance, and produces as output a* $(1+\epsilon)$*–approximate solution with probability* $p > \frac{1}{2}$*.*

Definition C.4 randomized FPTAS *(*RFPTAS*). A* RPTAS *whose running time is polynomial on both the instance size and the constant* $\epsilon > 0$*.*

Definition C.5 *(*Relaxed Decision Procedure - RDP*). A* RDP *is an algorithm that given a minimization problem and and a value* d*, an* ϵ*-relaxed decision procedure*

- *either decides that there is no solution of objective value at most* d*.*
- *or returns a solution of objective value at most* $(1+\epsilon)d$

Definition C.6 *(*Randomized Relaxed Decision Procedure - RRDP*). A (*RRDP*), is a randomized algorithm that given a minimization problem and and a value* d*, with probability of success at least* $p > \frac{1}{2}$*,*

- *either decides that there is no solution of objective value at most* d*,*
- *or returns a solution of objective value at most* $(1+\epsilon)d$*.*

Definition C.7 The *poly–bottleneck* property. *We say that a combinatorial optimization problem has the* poly–bottleneck *property, if its optimal objective value is always within a polynomial factor of one of its input elements (weights). Unweighted problems are assumed to have elements of weight 1.*

C.2 Problem Definitions

Definition C.8 Problem *SUM*: *There are* n *independent jobs and* m *unrelated parallel machines. Each job* j *is to be assigned to one of the machines and at any time, a machine can process at most one job. The processing time of job* j *on machine* i *is* p_{ij}*. The objective is to find a schedule that minimizes makespan.*

Definition C.9 Problem *SUMβ*: *The restriction of* SUM *where there is a constant* β*, such that for every instance of* SUMC*:*

$$\frac{\max_{i,j} p_{ij}}{\min_{i,j} p_{ij}} \leq \beta \quad . \tag{54}$$

Definition C.10 Problem *SUMC*: *There are n independent jobs and m unrelated parallel machines. Each job j is to be assigned to one of the machines and at any time, a machine can process at most one job. The processing time of job j on machine i is p_{ij}. Assigning job j to machine i incurs a cost c_{ij}.* SUMC *is a decision problem and the objective is to find a schedule with bounded makespan and cost.*

Definition C.11 Problem *SUMCoptT*: *This problem, which is also known as the generalized assignment problem, is an optimization version of* SUMC*. For a given makespan value T, the objective is to find a schedule of minimum cost C and makespan at most T.*

Definition C.12 Problem *SUMCoptC*: *The symmetric problem of* SUMCoptT*. Given an instance of* SUMC *and a value C for the cost, the objective is to find a schedule of minimum makespan and cost at most C.*

Definition C.13 Problem *SUMCoptTC*: *An optimization problem of* SUMC *where the objective is to optimize a linear function of the makespan and the cost of the schedule.*

Latin Letter Notation

A-SUMβ	:	algorithm *A-SUMβ* for *SUMβ*
A-SUM	:	algorithm *A-SUM* for *SUM*
A-SUMC	:	algorithm *A-SUMC* for *SUMC*
BSP	:	The Bulk Synchronous Parallel computing model.
EREWPRAM	:	Exclusive Read Exclusive Write *PRAM*
FRR	:	The filtered randomized rounding procedure
ILP	:	integer linear program or integer linear programming
ILP-SUM	:	integer program formulation of *LP-SUM*
ILP-SUMC	:	integer program formulation of *LP-SUMC*
LP	:	linear program or linear programming
LP-SUM	:	LP relaxation of the integer program formulation of *LP-SUM*
LP-SUMβ	:	LP relaxation of the
	:	integer program formulation of *LP-SUMβ*
LP-SUMC	:	LP relaxation of the integer program formulation of *LP-SUMC*
LogP	:	The *LogP* model for parallel computation
n_0	:	The threshold value for the number of jobs
	:	in algorithm *SUMβ*
OPT	:	The optimal value of the scheduling problem
PDD	:	The logarithmic-potential based algorithm
	:	for linear programs in block-angular form
PRAM	:	The Parallel Random Access Machine
Prob	:	probability of an event
T	:	Makespan of a schedule
T_1	:	Minimum makespan that satisfies relaxed feasibility
T_2	:	Relaxed feasible makespan found by binary search procedure
of Fig. 1.		
T^*	:	Makespan of optimal schedule
$\mathcal{T}_1$	:	Technique to reduce Φ to Φ_1.
$\mathcal{T}_2$	:	Geometric grouping technique to reduce Φ to Φ_2 or Φ_1 to $\Phi_1 \bigcap \Phi_2$.
XRR	:	The standard exclusive randomized rounding technique.

Greek Letter Notation

ϵ	:	Overall approximation ratio
ϵ_1	:	Error Ratio for binary search
ϵ_2	:	Error Ratio for PDD algorithm
ϵ_3	:	Error Ratio for unlucky jobs
ϵ_4	:	Error Ratio for randomized rounding
ϵ_5	:	Error Ratio for technique $\mathcal{T}_1$
ϵ_6	:	Error Ratio for technique $\mathcal{T}_2$
μ	:	Marginal mean value for fractional packing constraints
ξ	:	Deviation of the non-filtered rounded schedule
ρ	:	Upper bound on the probability of failure of the randomized algorithms
τ	:	Represents objective of fractional schedules
τ^*	:	Objective value of optimal fractional schedules
τ_1	:	Fractional schedule for optimal assignment of large jobs
τ_2	:	Objective value of best fractional schedule found
τ_3	:	Objective value of rounded schedule
τ_4	:	Objective value of filtered rounded schedule
τ_5	:	Objective value of final schedule
Φ	:	The set of all possible assignments of the large jobs to the machines
Φ_1	:	A subset of Φ of fully polynomial cardinality
Φ_2	:	A subset of Φ of size polynomial on m
Φ_f	:	The intersection of sets Φ_1 and Φ_2. The set Φ_f contains the assignments for the algorithm is executed. However all assignments of Φ_1 have to be considered.
φ	:	An assignment in Φ.
φ_i	:	The load on machine i due to φ.
$\bar{\varphi}$	:	An assignment in Φ_1.
χ	:	An assignment in Φ.
χ_i	:	The load on machine i due to χ.