Randomized Approximation Schemes for Scheduling Unrelated Parallel Machines

Pavlos S. Efraimidis* and 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. Randomized fully linear time approximation schemes are shown for the standard problem of minimizing the makespan (SUM) and the bicriteria problem of scheduling with bounded makespan and cost ($SUMC$). The core of the algorithms that also draw techniques from the related works [9], [8], and [1], is an interesting randomized rounding (RR) technique, the Combinatorial Randomized Rounding (CRR). While matching the approximation guarantee and the complexity of the best known sequential results of Jansen and Porkolab ([9]), 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.
Finally, the *poly–bottleneck* property for combinatorial optimization problems is defined and used to build approximation schemes for two natural optimization versions of $SUMC$.

Keywords: scheduling, approximation schemes, randomized rounding, combinatorial optimization, parallel algorithms
Track: A

---

* Financial support from the Bodosaki Foundation to perform doctoral studies is gratefully announced. Bodosaki Foundation, Leoforos Amalias 20, 10557 Athina, Greece
** Address: Research Unit 1, Computer Technology Institute, Riga Feraiou 61, 26221 Patras, Greece, fax: +30 61 993973.
[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 is the problem of assigning n jobs $j = \{1, \ldots, n\}$ to m machines $i = \{1, \ldots, m\}$ so 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 makespan is the maximum load on any machine. 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 makespan and cost.

Due to its theoretical and practical importance, the problem of scheduling n independent jobs on m unrelated parallel machines has been widely studied (for example [12], [8], [9]). The general problem, where both parameters m and n are specified as part of the problem instance is known to be NP-hard, and interestingly, it remains NP-hard even when m is defined to be $m = 2$. For the general problem, the best known results are due to Lenstra, Shmoys and Tardos ([12]), who showed a polynomial time 2-approximation algorithm for it and proved that, unless P=NP, no approximation ratio better than $3/2$ is possible. The inapproximability result raises the natural question whether better approximation results can be obtained when the number m of machines is fixed.

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 ([8]) 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 *SUM* for any fixed $\epsilon > 0$ where given in [12]. 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 [12] is inferior to the one of [8], it achieves a significantly smaller space complexity than the former algorithm. Kopidakis, Fayard and Zissimopoulos presented in [11] a linear time *PTAS* for *SUMβ*, a restricted version of *SUM* where all processing times are within a constant factor of each other.

Recently, Jansen and Porkolab ([9]) showed linear time approximation algorithms for *SUM* and *SUMC*. They presented a linear time *PTAS* 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 [9] accepts as input an instance of *SUMC* and values T and C, and either finds a schedule of makespan and cost at most $(1+\epsilon)T$ and $(1+\epsilon)C$ respectively, 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 behaves as a *RDP* with probability of success at least a given value $\rho : \; 0 < \rho < 1$. Similarly we define *randomized PTAS* (*RPTAS*) and *randomized FPTAS* (*RFPTAS*). Since *SUMC* has two objectives, i.e. makespan and cost, it admits several optimization versions. A natural optimization version is *SUMCoptT*, the problem of finding for given instance of *SUMC* and cost value C, the schedule of minimum makespan and cost at most C. Similarly let *SUMCoptC* be the problem of optimizing the cost when the makespan is bounded. A third option is the problem *SUMCoptTC* that optimizes a linear function of both objectives.

In the field of parallel algorithms Serna and Xhafa use in [20] an elegant technique of Trevisan ([22]) to claim a randomized $(2+\epsilon)$–approximation algorithm for *SUM*β and *SUM* and a randomized $(2+\epsilon)$–makespan 2–cost approximation algorithm for *SUMCoptC*. Both algorithms run for problems of size N in $polylog(N)$ time on $O(N)$ processors. In [3] the authors propose a framework that achieves parallel polylog(n) time approximation algorithms for *SUM* and *SUMC*. LP relaxations and *RR* techniques are used by Schulz and Skutella in [18] and [19] for sequential approximations to scheduling problems. The results of [3], [18], and [19] treat different problems or different problem settings and are not comparable to this work.

We first consider problem *SUM*β, an interesting restriction of *SUM*, and show algorithm *A-SUM*β, a simple linear time *RFPTAS* for it. Algorithm *A-SUM*β is based on linear programming (*LP*) and standard randomized rounding (*RR*) and it is meant to be a smooth introduction to the techniques used in the more involved algorithms of this work. However, algorithm *A-SUM*β is itself an interesting result, due to its simplicity and since it matches the best specific sequential algorithm for *SUM*β of [11] and clearly improves the complexity and approximation ratio of the parallel algorithm claimed in [20].

We then address the standard *SUM* and the bicriteria *SUMC* problems and present efficient randomized approximation algorithms for them. The core of our algorithms, which also draw techniques from the related works ([9], [8], [1]), is an interesting new *RR* procedure, the Combinatorial Randomized Rounding (*CRR*). The striking feature of *CRR* 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. Since the seminal work or Raghavan and Thompson in [16] several variations of *RR* have been shown. However applying simple combinatorial arguments within *RR* procedures to boost the Chernoff-type bounds and hence the performance guarantee of the overall algorithm, is a new approach (to the authors knowledge) that appears to be general enough to find more applications in rounding procedures for other integer linear programs. The algorithms based on *CRR* are very simple, once the general *CRR* 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 results for *SUM* and *SUMC*, while matching in performance guarantee and complexity the best known sequential results of Jansen and Porkolab ([9]), exhibit, due to the *CRR* technique, a significantly simpler and more general rounding scheme. This becomes especially evident in the approximation algorithm for the bicriteria *SUMC* problem. Finally, we define the *poly–bottleneck* property for combinatorial optimization problems and use it with algorithm *A-SUMC* to build $O(n \log n \log\log n)$ time *RFPTAS* for the optimization problems *SUMCoptT* and *SUMCoptC*.

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* ([20]), 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 complexities are given for the EREW PRAM, the most realistic of the PRAM models[2]. The algorithms of this work follow the common paradigm of calculating a fractional schedule with *LP* techniques and then rounding it to a near-optimal integer schedule. The *LP*'s that occur in the algorithms have a block-angular structure and are approximated very efficiently, sequentially or in parallel with the logarithmic-potential based price-directive decomposition method (*LogPDD*) of Grigoriadis and Khachiyan ([5]).

$E[X_{ij}]$ 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 $[n]$. The rest of this work is organized as follows. In Sec. 2 a simple *RFPTAS* for *SUMβ* is shown. The *CRR* 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.4. Due to limited space some proofs and some auxiliary figures have been placed in the Appendix together with a summary of the notation and the definitions used in this work.

2 The *SUMβ* Problem

SUMβ is the restriction of the standard *SUM* problem where in every problem instance, $\frac{p_{min}}{p_{max}} \geq \frac{1}{\beta}$, for $p_{min} = \min_{i,j} p_{ij}$, $p_{max} = \max_{i,j} p_{ij}$, and β is a given constant. We show algorithm *A-SUMβ*, a simple *RFPTAS* for *SUMβ*. A fractional schedule is found with a *LP* technique, and then it is rounded to an approximate schedule with a standard *RR* technique. This approach can satisfy any given constant approximate ratio, if the number of jobs n is larger than an appropriate constant n_0, which depends on m, ρ, β, and ϵ. For instances with number of jobs n less than n_0 an appropriate schedule can be found in constant time with a brute force method. Algorithm *A-SUMβ* and its analysis follow.

Input: An instance of *SUMβ* 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 $\epsilon_2 = \epsilon_4 = \frac{\epsilon}{3}$, $\mu = \frac{3\ln(m/\rho)}{(\epsilon_4)^2}$ 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.

[2] 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, they should imply efficient implementations on more practical parallel computation models, like the *BSP* ([23]) or the *LogP* ([2].

Step 1: *Constant number of jobs. IF* $(n < n_0)$ *THEN* optimal schedule can be found in $O(1)$–time with a brute force method.
Step 2: *Integer program formulation. IF* $n \geq n_0$ *THEN* formulate *SUM*β as an integer linear program and relax it to the linear program *LP-SUM*β.
Step 3: *Fractional Schedule.* Approximate *LP-SUM*β within $(1+\epsilon_2)$.
Step 4: *Rounding.* The approximate fractional solution is rounded randomly to an approximate integer schedule with *XRR*, a standard *RR* technique.

Constant number of jobs. If $n < n_0$ then there are less than m^{n_0} possible assignments of the jobs to the machines, and hence an appropriate schedule can be found in fully polynomial constant time with an enumeration technique of Horowitz and Sahni [8] (see technique $\mathcal{T}_1$ of Sec. B.1 of the Appendix.)
Fractional Schedule. For $n > n_0$ it will be shown that the makespan of the optimal schedule is large enough to guarantee with positive probability very tight *RR*. To simplify the analysis the problem is scaled with $1/p_{max}$, so that $\forall\, i,j\ :\ \frac{1}{\beta}\ \leq\ p_{ij}\ \leq\ 1$. The integer program formulation of *SUM*β is:

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

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 $x_{ij} \geq 0$ gives the linear program *LP-SUM*β that can be solved with any of the known polynomial time algorithms for *LP*. However *LP-SUM*β has specific properties that can be used to achieve very efficient approximate solutions. The problem variables of *LP-SUM*β are grouped into n independent m-dimensional simplices (blocks) and there is a constant number of positive packing constraints (coupling constraints). These properties are exploited by the logarithmic-potential based PDD algorithm *LogPDD* of Grigoriadis and Khachiyan ([5]), to approximate *LP-SUM*β within $(1+\epsilon_1)$. This is formally stated in the following Theorem which is used throughout this work. The proof is placed in Sec. B.2 of the Appendix.

Theorem 1. *The linear program* LP-SUMβ*, and the linear programs* LP-SUM *and* LP-SUMC *of the following Sections, 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-SUM*β and let τ_1 be the approximate objective value produced by the *LogPDD* algorithm for a given error ratio ϵ_2. Then:

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

Let *OPT* be the optimal makespan of *SUM*β and let τ^* be the optimal objective value of the relaxed problem *LP-SUM*β. The optimum of the relaxed problem cannot be worse the optimum of the original problem, hence $\tau^*\ \leq\ OPT$.

Furthermore the approximation ratio of algorithm *LogPDD* guarantees that $\tau_1 \leq \tau^* \cdot (1+\epsilon_2)$. Hence:

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

Let $d_j = \min_i p_{ij}$ be the minimum processing time of job j, and let $D = \sum_j d_j$ be the sum of all d_j. The optimal amount of work to process all jobs is exactly D. Simply assigning every job j to the machine i that achieves its minimum processing time d_j, gives a feasible schedule of makespan at most D, even if in the worst case all jobs j end up 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 is still at least D/m. Hence for *SUMβ* (and more generally for *SUM*):

$$\frac{D}{m} \leq \tau^* \leq OPT \leq D \ . \tag{2}$$

Equation 2 and the bound on β gives $\frac{D}{m} \geq \frac{n}{\beta m}$. From this we get $\tau_1 \geq \frac{D}{m} \geq \frac{n}{\beta m} \geq \frac{m\beta\mu}{\beta m} \geq \mu = \frac{3\log(\frac{m}{\rho})}{(\epsilon_4)^2}$. Combining this with Equ. 1 gives

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

Rounding. The fractional solution is rounded to an approximate integer schedule with *XRR*, a simple extension of Raghavan and Thompson's *RR* technique of [16, Sec.2] to 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, that is each job j is independently assigned one of the machines i. The probability that job j is assigned by the *RR* procedure to machine i is equal to the 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 $E[X_{ij}] = x_{ij}$. The processing load of each machine i in the rounded schedule 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 expected rounded load is equal to its load S_i in the fractional schedule:

$$\forall\, i \ : \ E[\Psi_i] = E[\sum_i p_{ij} X_{ij}] = \sum_i p_{ij} E[X_{ij}] = \sum_i p_{ij} x_{ij} = S_i \ . \tag{4}$$

Equation 4 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 simple extensions of Raghavan and Spencer's Chernoff-like bounds on the tail of the distribution of the Weighted Sum of Bernulli Trials ([15], [14]). The proof is placed in Sec. B.3 of the Appendix.

Theorem 2. *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*

$$Prob[\Psi > (1+\epsilon_4)T] < e^{\left(-\frac{(\epsilon_4)^2 T}{2(1+\frac{\epsilon_4}{3})}\right)} \text{ which is } \leq e^{\left(-\frac{(\epsilon_4)^2 T}{3}\right)} \text{ if } \epsilon_4 < 1 \ . \qquad (5)$$

Theorem 3. *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. For each machine i, the probability that its load Ψ_i in the rounded schedule is larger than $\tau_1(1+\epsilon_4)$ can be bounded with Equ. 5:

$$\forall\, i \ : \ P_i = \mathrm{Prob}\{\Psi_i > \tau_1(1+\epsilon_4)\mu\} \leq e^{-\frac{(\epsilon_4)^2 \mu}{3}} \leq \frac{\rho}{m} \ . \qquad (6)$$

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.

$$\mathrm{Prob}\{\tau_2 > (1+\epsilon_4)\cdot\tau_1) = \mathrm{Prob}\{\exists i : \Psi_i > \tau_1 \cdot (1+\epsilon_4)\} \leq \sum_i P_i \leq \rho \ . \qquad (7)$$

Using Theorems 1 and 3 we get the main result (proof in Sec. B of the Appendix):

Theorem 4. *Algorithm* A-SUMβ *is a* RFPTAS *for* SUMβ. *It 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*. We first show algorithm A_{SUM}, a *RRDP* for the decision version of *SUM*. Given an instance of the *SUM* problem and a makespan T, with probability of success at least $(1-\rho)$ algorithm *A-SUM* either produces a schedule of makespan at most $(1+\epsilon)T$ or decides that there is no schedule of makespan at most T. Algorithm *A-SUM* is then used 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 common paradigm of finding a fractional schedule and then rounding it to an approximate integer schedule. However the rounding is done with the new *CRR* technique, since standard *RR* cannot satisfy the tight approximation guarantee needed for the approximation scheme. The algorithm first selects a constant number of large jobs (set J_ℓ) and tries every possible assignment of them to the machines. For each possible assignment φ of the large jobs, a corresponding fractional schedule of all the jobs is found with the *LogPDD* algorithm. Among all fractional schedules the one of minimum makespan is selected and rounded to an integer schedule with *XRR*. *Every job $j \notin J_\ell$ in the rounded schedule, that has been randomly assigned to a machine i such that $p_{ij} > 1$ is called "unlucky" 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 rescheduled 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 with all jobs is with probability at least $(1-\rho)$, a $(1+\epsilon)$–approximate schedule.

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

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

Step 0: *Initializations.*

Let $\epsilon_1 = \epsilon_2 = \epsilon_3 = \epsilon_4 = \epsilon_5 = \epsilon_6 = \frac{\epsilon}{9} = \Theta(\epsilon)$. $\forall j,\; d_j = \min_i\{p_{ij}\}$, $D = \sum_j d_j$, $\mu = \frac{3 \ln \frac{2m}{\rho}}{(\epsilon_4)^2}$, $\xi = e^2 + \ln\left(\frac{2m}{\rho}\right)$, 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 Φ_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 *LogPDD*.

Among all fractional schedules select the one of minimum makespan.

Step 4: *Combinatorial Randomized Rounding.*

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

3.2 Analysis of algorithm *A-SUM*

To show that *A-SUM* is a *RRDP* for SUM it is sufficient to show that if the given T is a feasible makespan value then algorithm *A-SUM* returns with probability at least $(1-\rho)$ a schedule of makespan at most $(1+\epsilon)T$. 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 $d_j = \min_i p_{ij}$ and $D = \sum_j d_j$. To simplify the analysis, the problem is scaled by the factor $\frac{\mu}{T}$ so that the given makespan becomes $T = \mu$. The value $T = \mu$

has been chosen so that if all coefficients p_{ij} would be $p_{ij} \leq 1$, rounding a fractional schedule with *XRR* would give, with probability at least $1-\rho$, an integer schedule with makespan within the required approximation guarantee. The next steps deal with the existence of large coefficients $p_{ij} > 1$.

Simple Filtering. The first step deactivates all x_{ij} for which the corresponding p_{ij} is larger than T. This action is called "simple filtering" and does not influence the feasibility of any integer schedule of makespan at most T. Now all active p_{ij} are not larger than T, but some of the p_{ij} can still be larger than 1. Before finding an approximate fractional schedule, we 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 $k = \left\lceil \xi \cdot \mu \cdot m \cdot \frac{m}{\epsilon_3} \right\rceil$, 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: $J_\ell = \{j \mid d_j$ belongs to the k largest $d_j\}$.

Enumeration. Let Φ denote the set of all possible assignments φ of the large jobs $j \in J_\ell$ to the m machines. Let φ^* be the assignment of the large jobs to the machines as they are assigned in an optimal schedule of *SUM*. Since φ^* is not known, the algorithm is executed for each of the possible assignments $\varphi \in \Phi$. The cardinality of Φ is at most m^k, a constant. However $k = O(m^2 \ln^2(2m/\rho)\epsilon^{-3})$ and hence $1/\epsilon$ appears as an exponent of m. Lemma 1 shows that it is sufficient to examine only a substantially smaller subset $\Phi_f \subseteq \Phi$, with cardinality polynomial on ϵ. The set Φ_f is obtained from Φ by applying $\mathcal{T}_1$, a grouping technique of Horowitz and Sahni ([8]) in the way it has been used in [9], and $\mathcal{T}_2$, a geometric grouping technique. Detailed description of $\mathcal{T}_1$ and $\mathcal{T}_2$ and the proof of Lem. 1 are given in Sec. B.1 of the Appendix.

Lemma 1. *The cardinality of* Φ_f *and the running time for generating it are* $O(\min\left\{\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}, \frac{2\log(1/\epsilon)}{\epsilon}^{m}\right\})$ *and* $\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}$ *respectively. Using* Φ_f *instead of* Φ *in algorithm* A-SUM *introduces at most an arbitrary small constant error factor of* $(1+\epsilon_5)\cdot(1+\epsilon_6)$ *to the final solution.*

Using Lem. 1 improves *A-SUM* to a fully polynomial time algorithm, at the cost of introducing an arbitrarily small constant error factor to the approximation guarantee of the final solution. The rest of the analysis is done with the assumption that the algorithm examines all assignments of Φ. This assumption is later relaxed by introducing the extra ratio $(1+\epsilon_5)\cdot(1+\epsilon_6)$ to the final approximation guarantee (see Equ. 15).

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.

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

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 $\tau^* \leq T$. Let *LP-SUM*(φ^*) be the *LP* relaxation of *ILP-SUM*(φ^*) obtained by relaxing the integrality constraints on the variables x_{ij} to $\forall i, j : \; x_{ij} \geq 0$. The linear program *ILP-SUM*(φ^*) has a block-angular structure and can be approximated with algorithm *LogPDD* (The. 1). For given constant ϵ_2, the PDD algorithm runs in $O(n)$ sequential time and in $O(\log n)$ parallel time on $n/\log n$ processors, and produces a $(1+\epsilon_2)$–approximate fractional schedule. Let x_{ij} and τ_1 be the output produced by PDD. Then:

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

By the approximation guarantee of PDD and since $\tau^* \leq T$ we get:

$$\tau_1 \leq \tau^* \cdot (1+\epsilon_2) \leq OPT \cdot (1+\epsilon_2) \ . \tag{8}$$

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. Then:

$$\tau_2 \leq \tau_1 \leq T \cdot (1+\epsilon_2) \ . \tag{9}$$

Rounding. 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). The fractional schedule (x_{ij}) is rounded with *XRR* (Sec.2) to an approximate integer schedule for *SUM*. The rounding concerns only the jobs in J_s. 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 $E[X_{ij}] = x_{ij}$. As in the rounding step of Algorithm *A-SUMβ* 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:

$$\mathcal{E}_1 = \ \{ \text{ The makespan } \tau_3 \text{ of the rounded solution is } \tau_3 > \xi \cdot \tau_2 \} \ . \tag{10}$$

Applying The. 2 as in Lem. 3 proves the following Proposition (Detailed proof in Sec. C of the Appendix).

Proposition 1. *The probability of event $\mathcal{E}_1$ is at most $\rho/2$.*

Let J_u be the set of unlucky [3] 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" $p_{ij} > 1$.

[3] Note that unlucky jobs are dynamical, unlike the large jobs (J_ℓ) that are statically defined for every instance of *SUM*. Any job in J_s for which at least one coefficient p_{ij} is larger than 1 can possibly be a unlucky job in a rounded schedule. Hence separate rounded schedules obtained randomly from the same fractional schedules might have different sets of unlucky jobs.

Filtering. To boost the bounds on the deviation of the makespan of the rounded schedule *all unlucky jobs are removed from the rounded schedule.* The remaining schedule is called the filtered rounded schedule. For each machine i, let Ψ'_i be the random variable:

$$\Psi'_i = \varphi_i + \sum_{j \in J_s \ AND \ p_{ij} \leq 1} p_{ij} X_{ij} \ . \tag{11}$$

Ψ'_i corresponds to the load of the machine i due to all the remaining jobs, if unlucky jobs are excluded. 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:

$$\mathcal{E}_2 = \ \{ \text{ In the filtered rounded schedule } \tau_4 > (1+\epsilon_4) \cdot \tau_2 \}. \tag{12}$$

The probability of event $\mathcal{E}_2$ is bounded by the following Proposition which is proved similarly to Lem. 3 (proof in Sec. C of the Appendix).

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

The following combinatorial argument is used to handle the unlucky jobs:

Lemma 2. *Let $d_1 \geq d_2 \geq \ldots \geq d_n > 0$ be a sorted sequence of real numbers and let $D = \sum_{j=1}^{n} d_j$. Let $p \geq 0$ be a non-negative integer and $\epsilon_3 > 0$ a constant. For $k = \lceil p/\epsilon_3 \rceil$, any set S of reals $S = \{ \ d_i \ \mid \ i > k \ \}$ with $\mid S \mid \leq p$ satisfies:*

$$\sum_{d_i \in S} d_i \leq \epsilon_3 \cdot D \ . \tag{13}$$

Proof. The number of jobs n is assumed to be larger than the constant k (else SUM can be solved by a brute force method in constant time). Recall that the reals d_i are sorted in decreasing order. d_k satisfies $d_k \leq \frac{\epsilon_3}{p} D$, because else $\sum_{i=1}^{k} d_i > D$ (contradiction). Since $\forall \ d_i \ \in \ S \ \Rightarrow \ i > k$ this implies $\forall \ d_i \ \in \ S \ :$ $d_i \ \leq \ d_k \ \leq \ \frac{\epsilon_3}{p} D$. Hence: $\sum_{d_i \in S} d_i \leq p \cdot \frac{\epsilon_3}{p} \cdot D \leq \epsilon_3 \cdot D$.

Corollary 1. *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 $\sum_{j \in J_p} d_j \leq \epsilon_3 \cdot \tau_2$.*

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

$$\mathcal{E}_3 = \{ \sum_{j \in J_u} d_j > \epsilon_3 \cdot \tau_2 \} \ . \tag{14}$$

Proposition 3. *The probability of event $\mathcal{E}_3$ is at most $\frac{\rho}{2}$.*

Proof. By Proposition 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 $\frac{\rho}{2}$ and hence from Corollary 1 with the same probability the sum of their minimum processing times d_j is at most $\epsilon_3 \cdot \tau_2$. Hence by assigning each unlucky job to the machine where its processing time is minimized, even if in the worst case all unlucky jobs end up on the same machine the makespan of the schedule does not increase by more than $\epsilon_4 \cdot \tau_2$.

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 makespan of the final schedule.

Proposition 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. $\text{Prob}\{\mathcal{E}_2 \bigcup \mathcal{E}_3\} \leq \text{Prob}\{\mathcal{E}_2\} + \text{Prob}\{\mathcal{E}_3\} \leq \rho$.
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. 2 and 3 the makespan τ_5 of the final schedule is:

$$\tau_5 \leq (\tau_4 + \epsilon_3 \cdot \tau_2) \cdot (1 + \epsilon_5) \cdot (1 + \epsilon_6) \Rightarrow \tau_5 \leq (1 + \epsilon) \cdot T \ . \tag{15}$$

Combining The. 1 and Prop. 4 gives:

Theorem 5. *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.

From Equation 2 it is known that the optimal makespan *OPT* of *SUM* is always in the interval $[\frac{D}{m}, D]$. Hence using algorithm *A-SUM* within a simple binary search procedure (Fig. 1 in the Appendix) gives an approximation scheme for the optimization version of problem *SUM* (proof in Sec. C of the Appendix).

Theorem 6. *There 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.

4 Makespan and Cost

The methodology used for algorithm *A-SUM* can be used to build algorithm *A-SUMC*, a fully linear time *RRDP* for the bicriteria problem *SUMC*. Algorithm *A-SUMC* is essentially identical to *A-SUM* except:

- The measure d_j is extended to $d_j = \min_i\{p_{ij} + c_{ij}\}$,
- in several expressions m is replaced with $m + 1$, and

– a job $j \notin J_\ell$ is unlucky if it is randomly assigned to a machine i, such that $p_{ij} > 1$ or $c_{ij} > 1$.

Theorem 7. *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.

The detailed description of the algorithms and the proofs of this section are given in Sec. D of the Appendix. Being a bicriteria problem admits *SUMC* to have several optimization versions. Two natural cases are *SUMCoptT* and *SUMCoptC*, which are obtained by specifying an upper bound on one objective and then optimizing the other objective. In this way, given an instance of *SUMC* and value C for the cost (T for the makespan), the problem of finding a schedule of minimum makespan for cost at most C (minimum cost for makespan at most T) is problem *SUMCoptT* (*SUMCoptC*). [4] We show how a *RRDP* for *SUMC* can be used to build a *RFPTAS* for the ϵ-relaxed version of *SUMCoptT*. The results can easily be adopted for the *SUMCoptC* problem. 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* for cost at most C. We identify a simple property of certain combinatorial optimization (CO) problems which is very useful for efficient approximation algorithms. We say that a weighted CO problem has the *poly–bottleneck*[5] property, if its optimal objective value is always within a polynomial factor of one of its input elements (weights).

Proposition 5. *The values of the makespan and the cost of any feasible schedule for* SUMC *are always within a linear factor of one of the problem weights.*

Hence *SUMC* has the *poly–bottleneck* property. Lemma 3 shows that a *RFPTAS* for *SUMCoptT* can be build based on the *poly–bottleneck* property of *SUMC* and the *RRDP* for it. The same can shown for *SUMCoptC* and generally for any problem that has the *poly–bottleneck* property, if a *RRDP* is given for the problem.

Lemma 3. *Let P be an instance of the* SUMCoptT *of size N and given cost C, approximation ratio $\epsilon > 0$ and maximum probability of failure ρ : $0 < \rho < 1$. 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.*

---

[4] A third optimization problem is *SUMCoptTC*, which optimizes a linear combination of both criteria, for example $d \cdot T + C$, for $d > 0$. This problem is discussed in [21] for an unrestricted number of machines. Jansen and Porkolab present in [9, Theorem 3.2] a simple technique to build a *FPTAS* for *SUMCoptTC* from a *RDP* for *SUMC*. Using *A-SUMC* with the same technique gives a *RFPTAS* for *SUMCoptTC*.

[5] The term "bottleneck" has been used by Hochbaum and Schmoys in [7] for a class of graph optimization problems. An important property of the bottleneck problems of [7] 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 Acknowledgements

We wish to thank Prof. Mike Grigoriadis for useful comments on an earlier draft of this work.

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. 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).
4. M. Grigoriadis and L. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM J. Optimization*, 4(1):86–107, 1994.
5. M. Grigoriadis and L. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Mathematics of Operations Research*, pages 317–327, 1996.
6. M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, editors. *Probabilistic Methods for Algorithmic Discrete Mathematics*. Springer, 1998.
7. 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.
8. E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling non-identical processors. *Journal of the ACM*, 23:317–327, 1976.
9. K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *ACM Symposium on Theory of Computing*, pages 408–417, 1999.
10. H. Karloff. *Linear Programming*. Birkhäuser, 1991.
11. 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.
12. J.K. Lenstra, D.B. Shmoys, and È. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.
13. S. Plotkin, D. Shmoys, and È. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.
14. 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.
15. P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.

A

16. P. Raghavan and C.D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.
17. A. Schrijver, editor. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.
18. A.S. Schulz and M. Skutella. Scheduling-lps bear probabilities: Randomized approximations for min-sum criteria. In R. Burkard and G. Woeginger, editors, *Proceedings of the 5th Annual European Symposium on Algorithms (ESA'97), Lecture Notes in Computer Science 1284*, pages 416–429, Berlin, 1997. Springer.
19. A.S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. Submitted. (http://www.math.tu-berlin.de/~skutella/publications.html), May 1999.
20. M. Serna and F. Xhafa. Approximating scheduling problems in parallel. In *EuroPAR 97*, 1997.
21. D. Shmoys and È. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, A62:461–474, 1993.
22. L. Trevisan. Positive linear programming, parallel approximation and pcp's. In *4th European Symposium on Algorithms*, pages 62–75, 1996.
23. L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

APPENDIX

A Definitions and notation

Definition 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 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 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 $p > 0$.*

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

Definition 5. *(*Relaxed Decision Procedure *(*RDP*): A* RDP *is an algorithm that given a minimization problem 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 6. Randomized Relaxed Decision Procedure *(*RRDP*): A (*RRDP*) for a minimization problem, is a randomized algorithm that given a the problem and a value d, with probability of success at least* $\rho > 0$[6]

- *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 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*.*

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

$$\frac{\max_{i,j} p_{ij}}{\min_{i,j} p_{ij}} \leq \beta \ . \tag{16}$$

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

[6] In some cases ρ is by definition $\rho > \frac{1}{2}$. However for the applications of this work this is not important since any fixed $\rho > 0$ can be amplified to any value less than 1, by repeating the experiment a constant number of times.

Notation (Latin) and Abbreviations

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*
CRR	: The combinatorial randomized rounding technique
CO	: Combinatorial Optimization
ILP	: integer linear program or integer linear programming
ILP-SUM	: integer program formulation of *SUM*
ILP-SUMb	: integer program formulation of *SUMβ*
ILP-SUMC	: integer program formulation of *SUMC*
J	: set of all jobs
J_ℓ	: set of large jobs
J_s	: set of non-large jobs $J_s = J \setminus J_\ell$
J_u	: set of "unlucky" jobs
LP	: linear program or linear programming
LP-SUM	: LP relaxation of *ILP-SUM*
LP-SUMβ	: LP relaxation of *ILP-SUMb*
LP-SUMC	: LP relaxation of *ILP-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
LogPDD	: The logarithmic-potential based algorithm
	: for linear programs in block-angular form
PRAM	: The Parallel Random Access Machine
Prob	: probability of an event
Prop.	: Proposition
RR	: Randomized Rounding
Sec.	: Section
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 $\Phi_1 \bigcap \Phi_2$.
The.	: Theorem
XRR	: The standard *RR* technique described in Sec. 2.

Notation (Greek)

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	: Subset of Φ generated by technique $\mathcal{T}_1$
Φ_2	: Subset of Φ generated by technique $\mathcal{T}_2$
Φ_f	: The intersection of sets Φ_1 and Φ_2. The set Φ_f contains the assignments for which the algorithm is executed.
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.

B Proofs for Section 2

B.1 Approximate Enumeration

Φ is the set of all possible assignments φ of the large jobs $j \in J_\ell$ to the m machines and φ^* is the assignment of the large jobs to the machines in an optimal schedule of *SUM*. Since φ^* is not known, the algorithm is executed for each of the possible assignments $\varphi \in \Phi$. The cardinality of Φ is at most m^k, a constant. However $k = O(\frac{m^2 \ln^2(2m/\rho)\epsilon^{-3}}{)}$ and hence $1/\epsilon$ appears as an exponent of m. Hence the running time of the algorithm is not fully polynomial. In this Section it is shown that it is sufficient to examine only a substantially smaller subset $\Phi_f \subseteq \Phi$, with cardinality polynomial on ϵ. The set Φ_f is obtained from

Φ by applying $\mathcal{T}_1$, a grouping technique of Horowitz and Sahni ([8]) in the way it has been used in [9], and $\mathcal{T}_2$, a geometric grouping technique. Using set Φ_f instead of Φ improves *A-SUM* to a fully polynomial time algorithm, at the cost of introducing an arbitrarily small constant error factor to the approximation guarantee of the final solution.

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. This enumeration technique has been used by Horowitz and Sahni in [8] for finding an ϵ–approximate schedule to *SUM*. Jansen and Porkolab used in [9, Algorithm L] a similar technique to generate all substantially different schedules of a number of jobs.

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 $\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}$. ([9, 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 again partitioned, but this time to 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 4. *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 $O(m^L) = O(\frac{\log(1/\epsilon_6)}{\log(1+\epsilon_6)}^m) = O(\frac{2\log(1/\epsilon_6)}{\epsilon_6}^m)$. The last equality is due to Lemma 4. There is a simple algorithm $\mathcal{A}_L$ that implements techniques $\mathcal{T}_1$ and $\mathcal{T}_2$ and generates the set Φ_f from Φ.

Algorithm $\mathcal{A}_L$:

0: $\Phi(0) = \{(0, 0, \ldots, 0)\}$;
1: *FOR* each large job $j = 1, \ldots, k$ *DO BEGIN* :
2: *FOR* each n–tuple t in $\Phi(j-1)$ *DO BEGIN*:
3: *FOR* each of the $i = 1, \ldots, m$ machines *DO BEGIN*:
4: Generate the n–tuple $t' = t + (0, \ldots, p_{ij}, \ldots, 0)$
(Add p_{ij} at position i of tuple t.)
5: *END*
6: *END*
7: Set $\Phi(j)$ to be the set of all substantially different tuples t'.
8: *END*
9: $\Phi = \Phi(k)$

In algorithm $\mathcal{A}_L$ 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. Clearly the cardinality of Φ_2 generated by $\mathcal{T}_2$ is smaller than the cardinality of Φ_1 of $\mathcal{T}_1$. However it is not clear if there is an algorithm more efficient than $\mathcal{A}_L$ that can directly generate Φ_2.

Lemma 5. *(Lemma 1). The cardinality of Φ_f and the running time for generating it are* $O(\min\left\{\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}, \frac{2\log(1/\epsilon)}{\epsilon}^{m}\right\})$ *and* $\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}$ *respectively. Using Φ_f instead of Φ in algorithm* A-SUM *introduces at most an arbitrary small constant error factor of* $(1+\epsilon_5)\cdot(1+\epsilon_6)$ *to the final solution.*

B.2 Approximating the Linear Programs with *LogPDD*

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

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

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 ([10], [17]). 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 $\mathcal{L}$ ([4],[13],[5]). The most efficient for the needs of this work is the logarithmic-potential based PDD algorithm of Grigoriadis and Khachiyan [5], which approximates the LP relaxation of the problem within $(1+\epsilon_1)$ in $O(n)$ time. *LogPDD* is the first method to provide width-free iteration bounds that are optimal up to polylog factors for a sufficiently large ratio of *#blocks/constraints*, which is the case in $\mathcal{L}$. Furthermore *LogPDD* admits a simple optimal work parallelization of $O(\log n)$–running time.

Theorem 8. *(Theorem 1). The linear program* LP-SUMβ *(Sec. 2), and the linear programs* LP-SUM *(Sec. 3) and* LP-SUMC *(Sec. D), 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 [5], 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 which is clearly enough since an 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-SUM*β.

B.3 Chernoff Bounds (Proof of Theorem 2)

Theorem 2 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 ([15], [14]).

Theorem 9. *([15, 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 $\Psi=\sum_{j=1}^r a_jX_j$. Then $E[\Psi]=\sum_{j=1}^r a_jp_j=S$. Let $\delta>0$, and $S=E[\Psi]>0$. Then

$$Prob[\Psi>(1+\delta)S]<\left[\frac{e^{(\delta)}}{(1+\delta)^{(1+\delta)}}\right]^S \quad . \tag{17}$$

The following technical Lemma will used to replace the right hand side of Equ. 17 with an expression that can be handled easier.

Lemma 6. *([6, Page 200, Lemma 2.4]).* For all $x \geq 0$,

$$(1+x)\ln(1+x) - \geq 3x^2/(6+2x) \ . \tag{18}$$

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 6 and Theorem 9:

Lemma 7. *For $\delta > 0$ and $S \geq 0$*

$$\left[\frac{e^(\delta)}{(1+\delta)^{(1+\delta)}}\right]^S \leq e^{\left(-\frac{(\delta)^2 S}{2(1+\frac{\delta}{3})}\right)} \ , \textit{ and for } \delta < 1 : e^{\left(-\frac{(\delta)^2 S}{2(1+\frac{\delta}{3})}\right)} \leq e^{\left(-\frac{(\delta)^2 S}{3}\right)} \ . \tag{19}$$

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

$$B(S', \frac{T-\lambda-S'}{S'} + \frac{T}{S'}\cdot\delta)\cdot\delta) \leq B(T,\delta) \ , \tag{20}$$

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

This completes the proof of Theorem 2:

Theorem 2: *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*

$$\text{Prob}[\Psi > (1+\delta)T] < e^{\left(-\frac{(\delta)^2 T}{2(1+\frac{\delta}{3})}\right)} \text{ and for } \delta < 1 \ : \ e^{\left(-\frac{(\delta)^2 T}{2(1+\frac{\delta}{3})}\right)} \leq e^{\left(-\frac{(\delta)^2 T}{3}\right)} \ . \tag{21}$$

B.4 Proof of Theorem 4

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 *LogPDD* 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. B.1. Combining all these complexities proves the Theorem:

Theorem 10. *(Theorem 4). 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.*

C Proofs for Section 3

C.1 Proof of Proposition 6

Proposition 6. *(Proposition 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 and it is similar to the proof of Lem. 3 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:

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

C.2 Proof of Proposition 7

Proposition 7. *(Proposition 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.

$$\text{Prob}\{\Psi'_i > \tau_2 \cdot (1+\epsilon_4)\} < B(\tau_2, \epsilon_4) \leq \frac{\rho}{2\cdot m} \ . \tag{24}$$

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

$$\begin{aligned}\text{Prob}\{\tau_4 > \tau_2 \cdot (1+\epsilon_4)\} = \text{Prob}\{\exists\, i \;:\; \Psi_i' > \tau_2 \cdot (1+\epsilon_4)\} \leq \\ \sum_i (\text{Prob}\{\Psi_i' > \tau_2 \cdot (1+\epsilon_4)\}) \leq \frac{\rho}{2} \;.\end{aligned} \tag{25}$$

C.3 Proof of Theorem 5

Theorem 11. *(Theorem 5). 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 set of assignments Φ_f can be calculated in time $\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}$ and has cardinality $\mathcal{F} = \min\left(\left(\frac{m\log(m/\rho)}{\epsilon}\right)^{O(m)}, m^{\frac{2\log(1/\epsilon)}{\epsilon}}\right)$. Algorithm *LogPDD* has to be executed once for each assignment $\bar{\varphi} \in \Phi_f$. The complexity of the *LogPDD* 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.

C.4 Proof of Theorem 6

Theorem 12. *(Theorem 6). There 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. We will show that the binary search procedure of Fig. 1 is *RFPTAS* for *SUM* with sequential running $O(n)$ and parallel running time $O(\log n)$ on a $O(\frac{n}{\log n})$ processor *EREW PRAM*. 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 2 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 $1-\rho$.

<u>Binary Search</u>

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

Fig. 1. Binary Search

Note 1. 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.

D Supplement to Section 4

D.1 The Problem *SUMC*

In this section, we show algorithm A_{SUM}, a linear time *RRDP* for the problem *SUMC*. Given an instance of *SUMC*, values T and C for makespan and cost respectively, and constants $0 < \epsilon < 1$ and $0 < \rho < 1$, with probability of success at least $(1-\rho)$ algorithm *A-SUMC* either produces a schedule of makespan and cost at most $(1+\epsilon)T$ and $(1+\epsilon)C$ respectively or decides that there is no schedule of makespan at most T and cost at most C. Let $\mu = \frac{3 \ln \frac{2m+2}{\rho}}{(\epsilon_4)^2}$, 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. The appropriate constants are $\epsilon_1 = \epsilon_2 = \epsilon_3 = \epsilon_4 = \epsilon_5 = \epsilon_6 = \frac{\epsilon}{9} = \Theta(\epsilon)$. $\forall j,\ d_j = \min_i\{p_{ij} + c_{ij}\}$, $D = \sum_j d_j$,

$\xi = e^2 + \ln\left(\frac{2m+2}{\rho}\right)$, and $k = \left\lceil \xi \cdot \mu \cdot (m+1) \cdot \frac{m+1}{\epsilon_3} \right\rceil$. A job j that is not large, is unlucky if it is randomly assigned to a machine i such that $(p_{ij} > T\ OR\ c_{ij} > C)$.

D.1.1 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 *LogPDD*. 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 with probability at least $1-\rho$ a $(1+\epsilon)$–approximate schedule, if the values T and C are feasible.

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

Step 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 *LogPDD*.

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

Step 5: *Combinatorial 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 $i = \text{argmin}_i\{p_{ij} + c_{ij}\}$

D.1.2 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:

$$\mu = \frac{3 \cdot \ln\left(\frac{2m+2}{\rho}\right)}{(\epsilon_4)^2} \quad . \tag{26}$$

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

$$\xi = e^2 + \ln\left(\frac{2m+2}{\rho}\right) \quad , \tag{27}$$

$$p = \mu \cdot \xi \cdot (m+1) \;\; , \text{ and} \tag{28}$$

$$k = \left\lceil (\mu \cdot \xi \cdot (m+1)) \cdot \left(\frac{m+1}{\epsilon_3}\right) \right\rceil \quad . \tag{29}$$

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.

$$\forall \text{ job } j,\; d_j = \min_i \{p_{ij} + c_{ij}\} \text{ and } D = \sum_j d_j \quad . \tag{30}$$

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

$$J_\ell = \{ j \mid d_j \text{ belongs to the } k \text{ largest } d_j \} \quad . \tag{31}$$

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 where $k = O(\frac{(m+1)^2 \log^2(\frac{2m+2}{\rho})}{\epsilon^3})$. Hence the cardinality of Φ is not polynomial on ϵ.

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 $\mathcal{T}_1$ and $\mathcal{T}_2$. By Lem. 1 using Φ_f instead of Φ introduces at most an arbitrary small constant error factor $(1+\epsilon_5)(1+\epsilon_6)$ to the approximation guarantee for the final solution. The cardinality of Φ_f is constant that depends polynomially on ϵ which is mandatory so that the overall approximation scheme is fully polynomial. 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 40, 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 $ILP-SUMC(\varphi^*)$:

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

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:

$$\tau^* \le \mu = T = C \ . \tag{32}$$

Let *LP-SUMC*(φ^*) be the *LP* relaxation of *ILP-SUMC*(φ^*) obtained by relaxing the integrality constraints on the variables x_{ij} to $\forall i,j:\ x_{ij}\ge 0$. *LP-SUMC*(φ^*) has a block-angular structure and can be approximated with the logarithmic-potential based PDD algorithm of [5] 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:

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

By the approximation guarantee of *LogPDD* and Equ. 32:

$$\tau_1 \le \tau^*\cdot(1+\epsilon_2) \le \mu\cdot(1+\epsilon_2) \ . \tag{33}$$

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.

$$\tau_2 = \min_{\varphi\in\Phi}\{\ \tau_\varphi \ \mid\ \tau_\varphi = \epsilon_2 - \text{approximate solution to } \textit{LP-SUMC}(\varphi)\} \ . \tag{34}$$

Hence:

$$\tau_2 \le \tau_1 \le \mu\cdot(1+\epsilon_2) \ . \tag{35}$$

Rounding. As in Sec. 3 the best fractional solution found is rounded to an approximate schedule, with Combinatorial Randomized Rounding (*CRR*). 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:

$$\mathcal{E}_1 = \ \{ \text{ In the rounded solution } \tau_3 > \xi \cdot \mu\}. \ . \tag{36}$$

Proposition 8. *The probability of event $\mathcal{E}_1$ is at most $\rho/2$.*

Proof. Similar to Proposition 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:

$$\mathcal{E}_2 = \ \{ \text{ In the filtered rounded schedule } \tau_4 > (1+\epsilon_4) \cdot \tau_2\}. \tag{37}$$

Proposition 9. *The probability of event $\mathcal{E}_2$ is at most $\rho/2$.*

Proof. Similar to Proposition 2.

Lemma 8. *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. 2.

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

$$\mathcal{E}_3 = \{\sum_{j \in J_u} d_j > \epsilon_3 \cdot \tau_2\} \ . \tag{38}$$

Proposition 10. *The probability of event $\mathcal{E}_3$ is at most $\frac{\rho}{2}$.*

Proof. Similar to Prop. 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 11. *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:

$$\text{Prob}\{\mathcal{E}_2 \bigcup \mathcal{E}_3\} \leq \text{Prob}\{\mathcal{E}_2\} + \text{Prob}\{\mathcal{E}_3\} \leq \rho \ . \tag{39}$$

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. 9 and 10 the value τ_5 of the final schedule is:

$$(\tau_5 \leq \tau_4 + \epsilon_3 \cdot \tau_2) \cdot (1 + \epsilon_5) \cdot (1 + \epsilon_6) \ \Rightarrow \ \tau_5 \leq (1 + \epsilon) \cdot \mu \ . \tag{40}$$

The set Φ_f of assignments has cardinality

$$\mathcal{F} = \min\left(\left(\frac{(m+1)\log((2m+2)/\rho)}{\epsilon}\right)^{O(m)}, \left\{\frac{2\log(1/\epsilon)}{\epsilon}\right\}^{(m+1)}\right)$$

and it can be calculated in

$$\left(\frac{(m+1)\log((2m+2)/\rho)}{\epsilon}\right)^{O(m)}$$

time with the enumeration techniques $\mathcal{T}_1$ and $\mathcal{T}_2$ of Sec. B.1. Algorithm *LogPDD* has to be executed once for each of the assignments $\bar{\varphi} \in \Phi_f$. The complexity of the *LogPDD* 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 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 13. *(Theorem 7). 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.

D.2 Proof of Proposition 5

Proposition 12. *(Proposition 5). 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 input weights of the problem instance.*

Proof. Let p_{max} be the maximum processing time and c_{max} be the maximum cost that appear 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}. Similarly, the optimal cost can be shown to be always in the interval $[c_{max}, nc_{max}]$, where c_{max} is always one of the given costs c_{ij}.

D.3 Proof of Lemma 3

Lemma 9. *(Lemma 3). Let P be an instance of the* SUMCoptT *of size N and given cost C, approximation ratio $\epsilon > 0$ and maximum probability of failure $\rho :\ 0 < \rho < 1$. 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* $l = x$;
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 [9] instead of algorithm *A-SUMC*.