Efficient Approximation Algorithms for Scheduling Unrelated Parallel Machines

Pavlos S. Efraimidis[1] and Paul G. Spirakis[2]

[1] Department of Electrical and Computer Engineering,
Democritus University of Thrace,
Bas. Sophias 12, 67100 Ksanthi, Greece
pefraimi@ee.duth.gr
[2] Research and Academic Computer Technology Institute (RACTI)
and Patras University, Greece
Riga Fereou 61, 26221 Patras, Greece
spirakis@cti.gr

Abstract. Scheduling n independent jobs on m Unrelated Parallel Machines (SUM) is the problem of assigning n jobs $j = 1, .., n$ to m machines $i = 1, .., m$ 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 objective is to minimize the makespan of the schedule. SUM is an NP-hard problem even when the number of machines is defined to be $m = 2$.
In this work, efficient approximation algorithms for SUM, when the number of machines is an arbitrary constant, are presented. The results hold for SUM and several extensions of SUM, like the Generalized Assignment Problem (GAP). The approximation algorithms can achieve any given approximation ratio ϵ (approximation schemes) and admit efficient parallelization. The core of the algorithms is a new rounding procedure, the so-called Combinatorial Randomized Rounding (CRR) technique. Furthermore, an interesting property of combinatorial optimization problems and a new Chernoff-like bound, are introduced.

1 Introduction

Scheduling 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. The restriction of SUM where all processing times p_{ij} are within a constant factor β of each other is the problem SUMb. 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. Scheduling problems like SUM, SUMC and other variations have been widely studied, like for example in [8], [5] and [6]. It is known that the general case of SUM, where both parameters m and n are specified as part of the problem instance is NP-hard, and interestingly, it remains NP-hard even when m is defined to be $m = 2$.

For the general problem, the best known result is a 2-approximation algorithm given in [8], where it is also shown that, unless P=NP, no approximation ratio better than 3/2 is possible.

Henceforth the number of machines is always assumed to be a constant. A polynomial time approximation scheme (PTAS) is an algorithm that for each $\epsilon > 0$ finds a $(1+\epsilon)$–approximate solution in time polynomial in the problem size N. If the running time is also polynomial on ϵ then the scheme is called Fully PTAS(FPTAS). A FPTAS for SUM was given in [5] and later an interesting PTAS for SUM was given in [8]. A linear time FPTAS for SUMb was given in [7]. The best known results are given by Jansen and Porkolab ([6]) who showed a linear time FPTAS for SUM and and a linear time ϵ–relaxed decision procedure (RDP) for SUMC. The RDP for SUMC 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 failure at most a given value $\rho: \ 0 < \rho < \frac{1}{2}$. Similarly we define randomized PTAS (RPTAS) and randomized FPTAS (RFPTAS). As a bicriteria problem SUMC admits several optimization versions. Given an instance of SUMC and a cost value C a natural problem is to find a schedule of minimum makespan and cost at most C (problem SUMCoptT). Similarly SUMCoptC is the problem of optimizing the cost when the makespan is bounded. A third option is to optimize a linear function of both objectives (problem SUMCoptTC). The best known parallel algorithms, a randomized $(2+\epsilon)$–approximation algorithm for SUMb and SUM and a randomized $(2+\epsilon)$–makespan 2–cost approximation algorithm for SUMCoptC, are claimed in [14]. Both algorithms run for problems of size N in $polylog(N)$ time on $O(N)$ processors. Other parallel algorithms for related but not directly comparable scheduling problems are given in [2],[12], and [13].

2 Our results

In this work, we present sequential and parallel approximation schemes for SUMb, SUM, SUMC and optimization versions of SUMC.We start with a linear time RFPTAS for the restricted problem SUMb. While this result serves as an introduction to the more involved techniques of this work, it has its own interest. It is simple, it matches the best specific sequential algorithm for SUMb of [7] and it clearly improves upon both the complexity and approximation ratio of the parallel algorithm of [14].

We then show a linear time RRDP for the standard SUM problem and use it to build a linear time RFPTAS for SUM. Similarly, we show a linear time RRDP for the bicriteria SUMC problem. Interestingly the two algorithms are almost identical, a fact that proves the generality of the rounding procedure. Furthermore the same approach can handle even more criteria with the same approximation guarantee and the same asymptotic complexity. The results for SUM and SUMC, while matching in performance guarantee and complexity the best known sequential results of [6] (which however are deterministic), 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 to build $O(n \log n \log\log n)$ time RFPTAS for the optimization problems SUMCoptT and SUMCoptC.

The algorithms of this work are based on a new randomized rounding (RR) technique, the combinatorial randomized rounding (CRR) procedure. CRR introduces combinatorial arguments to the RR procedure and achieves in specific problems settings to bound the deviations of the rounded solutions below any fixed ratio. This is a strong, and partially surprising improvement upon the logarithmic bounds achieved with the standard RR technique ([11]).

All algorithms admit simple optimal work parallelizations that outperform the best claimed parallel algorithms for SUM and SUMCoptC ([14]), both in the performance ratio ($1+\epsilon$ 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. The parallel algorithms require $O(1)$ or ($O(\log n \log\log n)$) iterations, and hence should imply efficient implementations on practical parallel computation models, like the *BSP* ([16]) or the *LogP* ([1]. Due to space limitations certain proofs have been omitted from this work. A detailed description of all results is given in [3].

3 The SUMb Problem

SUMb is the restriction of standard SUM where there is a given constant β such that: $\frac{p_{min}}{p_{max}} \geq \frac{1}{\beta}$, for $p_{min} = \min_{i,j} p_{ij}$, and $p_{max} = \max_{i,j} p_{ij}$. We show a simple RFPTAS for SUMb. A fractional schedule is found 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.

Input: An instance of SUMb 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 $\epsilon_2 = \epsilon_4 = \frac{\epsilon}{3}$, $\mu = \frac{3\ln(m/\rho)}{(\epsilon_4)^2}$ an appropriate constant and $n_0 = m \cdot \beta \cdot \mu$ be a constant threshold value for the number n of jobs.
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 SUMb as an integer linear program and relax it to the linear program *LP-SUMβ*.
Step 3: *Fractional Schedule.* Approximate *LP-SUMβ* 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 [5] (see technique $\mathcal{T}_1$ of [3]).
Fractional Schedule. We assume that $n > n_0$. 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 SUMb 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–SUMb. The problem variables of LP–SUMb 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 ([4]), to approximate LP–SUMb within $(1 + \epsilon_1)$. The following Theorem follows directly from [4] and is used throughout this work (Claim 2.1 in [3]).

Theorem 1. *The linear program LP–SUMb (and the linear programs LP–SUM and LP–SUMC) can be approximated within any constant ratio ϵ in $(O(n))$–sequential time and in $O(\log n)$–time on $O(n/\log n)$–processors.*

Let τ^* be the optimal objective value of LP–SUMb and let τ_1 be the approximate objective value produced by the LogPDD algorithm for 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 SUMb and let τ^* be the optimal objective value of the relaxed problem LP–SUMb. Clearly $\tau^* \leq OPT$. Combining this with the approximation ratio of algorithm LogPDD we get:

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

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. 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. 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 SUMb (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 $\tau_1 \geq \frac{D}{m} \geq \frac{n}{\beta m} \geq \frac{m\beta\mu}{\beta m} \geq \mu = \frac{3\log\left(\frac{m}{\rho}\right)}{(\epsilon_4)^2}$. 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 technique of [11, Sec.2] to weighted sums of Bernoulli 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 Bernoulli 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 Bernoulli 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 Bernoulli 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 below of course) and hence the makespan of the rounded schedule might be larger than the fractional makespan. The following bound is an extension of Raghavan and Spencer's Chernoff-like bound on the tail of the distribution of the Weighted Sum of Bernoulli Trials ([10], [9]). The proof can be found in [3] (Theorem 2.1).

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 Bernoulli 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*

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

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:

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

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.

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

Using Theorems 1 and 3 we get the main result:

Theorem 4. *Algorithm A–SUMb is a RFPTAS for SUMb. It runs in $O(n)$ sequential time and in $O(\log n)$–parallel time on a $\frac{n}{\log n}$–processor EREW PRAM.*

4 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 problem of SUM and then use it to build the linear time RFPTAS.

4.1 Algorithm A–SUM

Given an instance of SUM 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 uses 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 $\Phi_f$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 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 $i = argmin_i\{p_{ij}\}$

4.2 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 A–SUM returns with probability at least $(1-\rho)$ a schedule of makespan at most $(1+\epsilon)T$. Let $\mu = 3\ln\frac{2m}{\rho}(\epsilon_4)^{-2}$ be an appropriate constant value. As in Sec. 3 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. This 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.

Large Jobs. Let k be the constant $k = \lceil \xi \cdot \mu \cdot m \cdot m \cdot \epsilon_3^{-1} \rceil$, and let the k jobs with the largest's d_j (ties 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. Section 3.3 of [3] 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 ([5]) in the way it has been used in [6], 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. 3.3 of [3].

Lemma 1. *The cardinality of Φ_f and the running time for generating it are polynomial in ϵ. 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 RFPTAS, 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. 12).

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 $\tau^* \leq T$. *ILP-SUM*(φ^*) is relaxed to the linear program *LP-SUM*(φ^*) which has a block-angular structure and is approximated within constant ϵ_2 with algorithm LogPDD (The. 1). The approximate solution x_{ij} of makespan τ_1 satisfies:

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

By the approximation guarantee of PDD and since $\tau^* \leq T$ we get: $\tau_1 \leq \tau^* \cdot (1+\epsilon_2) \leq OPT \cdot (1+\epsilon_2)$. 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:

$$\tau_2 \leq \tau_1 \leq T \cdot (1+\epsilon_2) \ . \tag{8}$$

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.3) 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 Bernoulli trial X_{ij}, such that $E[X_{ij}] = x_{ij}$. As in the rounding step of Algorithm A–SUMb 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 Bernoulli 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:

$$\mathcal{E}_1 = \ \{ \text{ The makespan } \tau_3 \text{ of the rounded solution is } \tau_3 > \xi \cdot \tau_2 \} \ . \tag{9}$$

Applying The. 2 as in The. 3 proves the following Proposition:

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

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

Filtering. *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: $\Psi'_i = \varphi_i + \sum_{j\in J_s \ AND\ p_{ij}\leq 1} p_{ij}X_{ij}$. 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 Bernoulli trials, where each weight is at most 1. Let τ_4 be the makespan of the filtered rounded schedule and let $\mathcal{E}_2$ be the event:

$$\mathcal{E}_2 = \ \{ \text{ In the filtered rounded schedule } \tau_4 > (1+\epsilon_4) \cdot \tau_2 \}. \tag{10}$$

The probability of event $\mathcal{E}_2$ is bounded by the following Proposition which is proved similarly to Lem. 3:

[1] Note that unlucky jobs are dynamically defined and hence separate roundings of the same fractional schedule might give different sets of unlucky jobs.

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. Let $k = \lceil p/\epsilon_3 \rceil$. Any set S of at most $\mid S \mid \leq p$ reals d_i that contains none of the k largest reals $d_i \mid i < k$ satisfies $\sum_{d_i \in S} d_i \leq \epsilon_3 \cdot D$.*

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). The real 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 the large jobs 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 $\epsilon_3 \cdot \tau_2$:

$$\mathcal{E}_3 = \{ \sum_{j \in J_u} d_j > \epsilon_3 \cdot \tau_2 \} \ . \tag{11}$$

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:

$$\tau_5 \leq (\tau_4 + \epsilon_3 \cdot \tau_2) \cdot (1+\epsilon_5) \cdot (1+\epsilon_6) \ \Rightarrow \ \tau_5 \leq (1+\epsilon) \cdot T \ . \tag{12}$$

Combining The. 1 and Prop. 4 gives:

Theorem 5. *Algorithm A–SUM is a $O(n)$–time RRDP for SUM. The parallel running time 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 gives an approximation scheme for the optimization version of problem SUM:

Theorem 6. *We provide 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.

5 Makespan and Cost

Algorithm A–SUM can be extended to handle the bicriteria problem SUMC. The resultant algorithm A–SUMC differs from A–SUM only in that

- 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 $c_{ij} > 1$.

The proof the following theorem is similar to the proof of theorem 5.

Theorem 7. *Algorithm A–SUMC is a $O(n)$–time RRDP for SUMC. The parallel running time is $O(\log n)$ on a $O(\frac{n}{\log n})$ processor* EREW PRAM.

5.1 Optimization versions of SUMC

The randomized relaxed decision procedure A–SUMC can be used within a binary search framework to build efficient approximation schemes for optimization versions of SUMC, like SUMCoptC, SUMCoptT and SUMCoptTC. More precisely, techniques presented in [2] can be used with A–SUMC to build RFPTAS algorithms for SUMCoptC and SUMCoptT. The optimization problem SUMCoptTC is discussed in [15] for an unrestricted number of machines. Using A–SUMC within the technique of [15] gives a RFPTAS for SUMCoptTC. A detailed description of the approximation schemes for the optimization versions of SUMC can be found in [3].

References

1. 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.
2. P. Efraimidis and P. Spirakis. Positive linear programming extensions: Complexity and applications. In *European Conference on Parallel Computing (Euro-PAR 2000)*, Technical Univeristy of Muenich, Germany, 2000.

3. P.S. Efraimidis and P.G. Spirakis. Randomized approximation schemes for scheduling unrelated parallel machines. Technical Report TR 00-007, Electronic Colloquium on Computational Complexity (ECCC), January 2000.
4. M. Grigoriadis and L. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Mathematics of Operations Research*, 21:317–327, 1996.
5. E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23:317–327, 1976.
6. K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *ACM Symposium on Theory of Computing*, pages 408–417, 1999.
7. 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.
8. J.K. Lenstra, D.B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.
9. 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.
10. P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.
11. P. Raghavan and C.D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.
12. 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.
13. A.S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. Submitted.
(http://www.math.tu-berlin.de/∼skutella/publications.html), May 1999.
14. M. Serna and F. Xhafa. Approximating scheduling problems in parallel. In *EuroPAR 97*, 1997.
15. D. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, A62:461–474, 1993.
16. L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.