INSTITUT FÜR INFORMATIK

A $(3/2+\epsilon)$ approximation algorithm for scheduling malleable and non-malleable parallel tasks

Klaus Jansen

Bericht Nr. 1203
Februar 2012
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

A $(3/2+\epsilon)$ approximation algorithm for scheduling malleable and non-malleable parallel tasks

Klaus Jansen

Bericht Nr. 1203
Februar 2012
ISSN 2192-6247

e-mail: kj@informatik.uni-kiel.de

A $(3/2+\epsilon)$ approximation algorithm for scheduling malleable and non-malleable parallel tasks

Klaus Jansen, Institut für Informatik, Universität zu Kiel
E-mail: kj@informatik.uni-kiel.de

February 16, 2012

Abstract

In this paper we study a scheduling problem with malleable and non-malleable parallel tasks on m processors. A non-malleable parallel task is one that runs in parallel on a specific given number of processors. The goal is to find a non-preemptive schedule on the m processors which minimizes the makespan, or the latest task completion time. The previous best result is the list scheduling algorithm with an absolute approximation ratio of 2. On the other hand, there does not exist an approximation algorithm for scheduling non-malleable parallel tasks with ratio smaller than 1.5, unless $P = NP$. In this paper we show that a schedule with length $(1.5+\epsilon)OPT$ can be computed for the scheduling problem in time $O(n \log n) + f(1/\epsilon)$. Furthermore we present an $(1.5 + \epsilon)$ approximation algorithm for scheduling malleable parallel tasks. Finally, we show how to extend our algorithms to the variant with additional release dates.

1 Introduction

In this paper we study the following scheduling problem with non-malleable parallel tasks. Suppose a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n jobs and a set $M = \{1, \ldots, m\}$ of m identical processors are given. Each job J_j has a processing time $p_j \in \mathbb{Z}^+$ and simultaneously requires $q_j \leq m$ processors during its execution. A schedule $S = ((S_1, M_1), \ldots, (S_n, M_n))$ is a sequence of starting times $S_j \geq 0$ together with a set $M_j \subset M$ of assigned processors of cardinality $|M_j| = q_j$ for $j = 1, \ldots, n$. A schedule is feasible, if each processor executes at most one job at each time. The objective of the problem denoted by $P|size_j|C_{max}$ is to find a feasible schedule with minimum length $OPT = \max_{j=1,\ldots,n}(S_j + p_j)$.

The problem $P|size_j|C_{max}$ is strongly NP-hard even for a constant number $m \geq 5$ of processors [6]. Therefore, we are interested in approximation algorithms. An α-approximation algorithm for a minimization problem is a polynomial-time algorithm A that constructs for each instance I a solution of value at most α times the optimum value $OPT(I)$ (i.e. algorithm A generates a schedule of length $A(I) \leq \alpha OPT(I)$); α is called the performance guarantee or absolute approximation ratio of the algorithm. Using a reduction from the partition problem, there is no approximation algorithm for $P|size_j|C_{max}$ with ratio better than 1.5, unless $P = NP$. Furthermore, there is no asymptotic approximation algorithm with ratio $\alpha OPT(I) + \beta$, where $\alpha < 1.5$ and β is a polynomial in n [17]. The best known approximation algorithm (the list scheduling algorithm), which has absolute ratio 2, was given implicity by Garey and Graham [9]; pointed out by Turek et al. [26] and Ludwig and Tiwari [23]. Feldmann, Sgall and Teng [7] observed that the length of a non-preemptive list schedule is actually at most $(2 - 1/m)$ times the optimum preemptive makespan.

A polynomial time approximation scheme (PTAS) for the case that the number m of processors is constant, denoted by $Pm|size_j|C_{max}$, was presented in [1, 13]. A polynomial time approximation scheme is a family of algorithms, that compute for any fixed $\epsilon > 0$ and instance I a schedule with length at most $(1+\epsilon)OPT(I)$. Recently, Jansen and Thöle [16] found a PTAS for the case where the number of processors is polynomially bounded in the number of jobs. For the general problem without any restrictions on the instance, the best known approximation algorithm is the list scheduling algorithm with ratio 2.

The following property is important for our approach. Let S_j be the starting time of the job $J_j \in \mathcal{J}$. Suppose that for each time step t the set $\mathcal{J}_t$ of jobs executed at time t uses at most m processors (i.e. for $\mathcal{J}_t = \{J_j | t \in [S_j, S_j + p_j)\}$ we have $\sum_{J_j \in \mathcal{J}_t} q_j \leq m$ for all time steps $t \in [0, max_j S_j + p_j))$. This property to hold for all starting times of jobs is a sufficient condition to obtain a feasible schedule. If this property is fulfilled for all starting times, we can generate a feasible (canonical) schedule by stepwise assigning jobs to processors [17, 16]; starting with time $t = 0$. Furthermore, the number of processor intervals used by jobs and idle intervals at each time step can be bounded by $n + 1$ [17], where n is the number of jobs. Therefore, we can obtain a compact way of encoding the output.

In the case with additional release dates $r_j \geq 0$, the starting times S_j of the jobs must be larger than or equal to r_j. For this problem, denoted by $P|size_j, r_j|C_{max}$, the list scheduling algorithm produces a schedule with makespan at most twice the makespan of an optimum preemptive schedule [17]. This leads to an 2-approximation algorithm for $P|size_j, r_j|C_{max}$. Naroska and Schwiegelshohn [25] independently showed that list scheduling gives a 2-approximation for the problem. Furthermore, no list scheduling [17] can achieve a better performance guarantee than 2 for the problem $P|size_j, r_j|C_{max}$. For an overview about other multiprocessor scheduling problems and results we refer to [3, 5].

Finding an improved approximation algorithm to reduce the gap between the best known ratio 2 and the lower bound of 3/2 (without and with release dates) is known as an open problem in the research area (see for example Johannes [17] and Ludwig [22]). We present the following new result in this paper:

Theorem 1.1 *For every fixed $\epsilon > 0$, there is an algorithm A such that*

$$A(I) \leq (1.5 + \epsilon)OPT(I)$$

for every instance I of $P|size_j|C_{max}$, where $A(I)$ is the length of the schedule generated by algorithm A and $OPT(I)$ is the length of the optimum schedule. The running time (number of elementary arithmetic operations) of the algorithm A is $O(n \log n) + f(1/\epsilon)$. All arithmetic operations are performed on numbers whose encoding length is bounded by a polynomial in $\log(n)$, $\log(m)$, and $\log(p_{max})$.

This result narrows the gap between the best known approximation algorithm with ratio 2 and the lower bound of 1.5. It can also be generalized to the case with additional release dates. Note that the input length $|I| \leq O(\log(n) + \log(m) + \sum_{j=1}^{n}(\log(q_j) + \log(p_j))) \leq O(n(\log(m) + \log(p_{max})))$ and $|I| \geq n + \log(m) + \log(p_{max})$.

A related problem is scheduling of malleable parallel jobs, denoted by $P|fctn_j|C_{max}$, where the numbers of processors per job are not known a priori. Here the execution time of a job depends on the number of allotted processors. Instead of one pair (p_j, q_j) for each job J_j we have a function $p_j : D_j \to \mathbb{Z}^+$ that gives the execution time $p_j(a)$ of each job J_j in terms of the number $a \in D_j \subset M$ of processors that are assigned to J_j. The goal is to find a feasible schedule S with minimum length $C^*_{max} = \max_{j=1,\ldots,n}(S_j + p_j(m_j))$, where $m_j \in D_j$ is the chosen number of processors for job J_j.

Belkhale and Banerjee [2] gave an algorithm with approximation ratio $2/(1 + 1/m)$ for monotone malleable jobs. Malleable jobs are called monotone, if their processing times and their work are monotone; i.e. allocating more machines to a job decreases the running time and increases the work:

if $\ell \leq \ell'$, then $p_j(\ell) \geq p_j(\ell')$ and $\ell p_j(\ell) \leq \ell' p_j(\ell')$. Turek et al. [26] improved this result, using no assumptions, and showed an approximation ratio of 2. Ludwig and Tiwari [23] also presented an approximation algorithm for malleable tasks with ratio 2, but with an improved running time. For monotone malleable jobs, Mounie, Rapine and Trystram [24] presented an approximation algorithm for the scheduling problem with ratio $1.5 + \epsilon$. Decker, Lücker, and Monien [4] presented an 1.25 approximation algorithm for scheduling n identical malleable jobs on m processors. Jobs are called identical, if the execution time on any number of processors is the same for all jobs (i.e. $p_j(\ell) = p_{j'}(\ell)$ for all pairs $J_j, J_{j'} \in \mathcal{J}$ and all numbers $\ell \in \{1, \ldots, m\}$).

Jansen and Porkolab [13] also presented a PTAS for $Pm|fctn_j|C_{max}$ with running time $O(n) + f(1/\epsilon)$ for the case that the number m of processors is constant. Recently, Jansen and Thöle [16] gave a PTAS for scheduling malleable jobs for the case that the number of processors is polynomially bounded by the number of jobs. The best previous result without additional assumptions (on the processing times and work) are algorithms with approximation ratio 2. Since the scheduling problem with parallel tasks is a special case of scheduling malleable tasks (with $D_j = \{q_j\}$ for each job J_j), there is also no approximation algorithm for $P|fctn_j|C_{max}$ with ratio < 1.5, unless $P = NP$. In this paper we prove the following new result.

Theorem 1.2 *For every $\epsilon > 0$, there is an algorithm A such that*

$$A(I) \leq (1.5 + \epsilon)OPT(I)$$

for every instance I of $P|fctn_j|C_{max}$, where $A(I)$ is the length of the schedule generated by algorithm A and $OPT(I)$ is the length of the optimum schedule. The running time of the algorithm A is polynomial in n, $\max_j |D_j|$ and $\log(m + p_{max})$; i.e. polynomial in the input length for any $\epsilon > 0$.

The algorithm can also be generalized to the variant with additional release dates.

The main ideas to schedule parallel tasks are as follows. We use a rounding and elimination technique for the jobs (with large or medium processing times) by Jansen and Solis-Oba [15] and a delay technique for the huge jobs (with very large execution times) by Jansen and Thöle [16] to obtain structural results for an approximate schedule. A variant of the AFPTAS for 2D strip packing by Kenyon and Rémila [19] is used to schedule the small jobs into horizontal layers. The main difficulty is to avoid that the running time of our algorithm depends on the number m of processors (since $|I| \leq O(n(\log(m) + \log(p_{max})))$. To obtain a polynomial running time, we use two interesting new techniques. First we create a gap in the schedule of height $1/2 + \delta$ and width $(1/8)\delta^4 m$ (where δ depends on ϵ) by analyzing the structure of the approximate schedule. Second we round the width of each large narrow job down to a multiple of αm (where α also depends on ϵ) and use a dynamic program to calculate the starting times of these jobs. To get the desired faster parameterized running time, we use another approach for the large narrow jobs. Here we guess an approximate load vector for these jobs and solve a linear program (LP) approximately to compute a schedule for almost all large narrow jobs. The remaining non-scheduled large jobs are placed into the constructed gap.

For malleable tasks the scheduling problem gets more complicated, because we do not know the processor numbers in advance. Here we set up a linear program (LP) to select the number of allotted processors for the jobs. One problem is to combine the small jobs with other jobs in the linear program. To do this we guess at the beginning the approximate structure of the small jobs in an optimum schedule and include this information into the LP. Interestingly, a basic LP solution only has a constant number of fractional variables. Fractional small and medium jobs can be executed at the end of the schedule, and fractional large jobs can be placed as above into a gap. The main difficulty here are fractional huge jobs. To handle these jobs, we pre-assign a constant number of huge jobs (depending on the number of fractional jobs in the LP) with largest widths for each rounded

starting and execution time. This helps us to select an integral processor number for each fractional huge job and to compute a feasible approximate schedule for all jobs.

The paper is organized as follows. In Section 2 we give the algorithm for scheduling parallel tasks. First in Subsection 2.1 we present several structural results for an approximate solution. Next in Subsection 2.2 we introduce our first approach for the large jobs based on dynamic programming, and in Section 2.3 we show how to generate a gap of height $1/2 + \delta$ and reasonable width. In Subsection 2.4 we show how to place small jobs via a linear program based on 2D strip packing. To speed up our algorithm, in Section 2.5 we give a faster algorithm based on an approximate load vector for large narrow jobs. Furthermore, in Section 3 we present the approximation algorithm for malleable tasks. Finally we discuss the variant with additional release dates in Section 4.

2 Scheduling Parallel Tasks

2.1 Structural Results for an Approximate Solution

First we determine a constant $\delta \in (0, \epsilon/15]$ depending on the accuracy $\epsilon > 0$ such that the total load $\sum_{j:p_j \in (\delta^5, \delta]} p_j q_j \leq \delta \sum_{j=1}^{n} p_j q_j$ and that $1/\delta$ is integral and even. Note that we may suppose that $\epsilon \leq 1$ (otherwise simply reduce ϵ to 1). Such a choice of a constant δ has been used by Jansen and Solis-Oba [15]. Suppose from now on that the number of processors m is larger than a polynomial in $1/\delta$; the exact bound is specified later. The case $m \leq poly(1/\delta)$ is easier and discussed at the end of Subsection 2.5. By computing an 2-approximate solution with makespan $LS(I) \leq 2OPT(I)$, we know that the optimum makespan $OPT(I) \in [LS(I)/2, LS(I)]$. By scaling (i.e. dividing all processing times by $LS(I)$) the optimum makespan $OPT(I) \in [1/2, 1]$. We run the algorithm described in the following for several discrete values $T(i) = 1/2 + i\delta$, where $i = 0, \ldots, 1/(2\delta)$. Clearly, there is one value i^* such that $T(i^*) \leq OPT(I) < T(i^*+1)$. By dividing the processing times by $T(i^*)$, for this choice we obtain $1 \leq OPT(I) < T(i^*+1)/T(i^*) \leq 1 + 2\delta$.

The goal is now to find a packing or schedule of the jobs into a big rectangle of width m and height $1+2\delta$. It is allowed to cut jobs of width q_j into vertical slices with integral widths. The vertical slices corresponding to a job have to be placed into the big rectangle onto the same level in order to start them at the same time. We divide now the schedule (or the big rectangle) into horizontal layers of height δ^2. Notice that there is only a constant number $(1 + 2\delta)/\delta^2$ of horizontal layers. Next we partition the jobs into four classes:

(a) huge jobs with running time $p_j > 1/2 + \delta$,

(b) large jobs with running time $p_j \in (\delta, 1/2 + \delta]$,

(c) medium jobs with running time $p_j \in (\delta^5, \delta]$, and

(d) small jobs with running time $p_j \leq \delta^5$.

By a modification of the schedule we obtain the following result:

Lemma 2.1 *If there is a schedule for the job set $\mathcal{J}$ with length $OPT(I) \leq 1+2\delta$ (where δ satisfies the load constraint above and $\delta \leq 1/10$), then there is also an approximate schedule for a subset $\mathcal{J}' \subset \mathcal{J}$ with length at most $1 + 5\delta$ such that*

(1) *huge and large jobs have processing time $a\delta^2$ with $a \in \mathbb{Z}^+$,*

(2) *huge and large jobs start and finish at multiples of δ^2,*

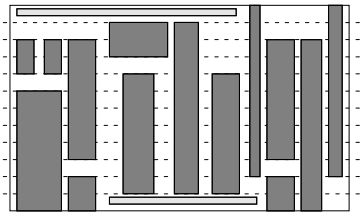

Figure 1: Approximate schedule after rounding the execution times

(3) *there are no medium jobs in* $\mathcal{J}'$,

(4) *small jobs lie completely in horizontal layers of height* δ^2.

The remaining jobs in $\mathcal{J} \setminus \mathcal{J}'$ *are all medium and can be placed at the end of the schedule with length* $\leq 2.5\delta$.

Proof: First we round the processing time of the huge and large jobs up to the next multiple of δ^2 and suppose that these jobs are started and finished at multiples of δ^2. Such a modification of a schedule is possible by increasing the height of each big horizonal layer of height δ by $2\delta^2$. This gives an enlarged schedule of total length $\leq (1+2\delta) + 2\delta^2(1+2\delta)/\delta \leq 1 + 4.4\delta$ for $\delta \leq 1/10$. Since each large or huge job J_j has processing time $> \delta$, J_j is situated in at least two consecutive big layers. This implies that J_j gets an additional height of at least $2\delta^2$. Therefore, a modification of the schedule by increasing the processing time and moving the job vertically by at most δ^2 is possible. Next we remove all medium jobs from the schedule. These jobs can be packed at the end of the schedule. Using the list scheduling algorithm for the medium jobs, the increased schedule length can be bounded by (see also Lemma 4.1 in [26])

$$2\max[\sum_{j:p_j \in (\delta^5,\delta]} p_j q_j/m, \max_{p_j \in (\delta^5,\delta]} p_j] \leq 2\max[\delta \sum_{j=1}^{n} p_j q_j/m, \delta] \leq 2\delta(1+2\delta) \leq 2.5\delta.$$

In addition we may assume that small jobs of height $\leq \delta^5$ lie completely in horizontal layers of height δ^2. Small jobs that lie in two layers also can be eliminated. The total height of such jobs is at most $[(1+2\delta)/\delta^2]\delta^5 \leq \delta^3 + 2\delta^4 \leq \delta^2$. The structure of a rounded solution is illustrated in Figure 1. The additional total height δ^2 of the small jobs that lie in two layers and the additional height $2\delta^2(1+2\delta)/\delta$ caused by rounding and shifting the large and huge jobs can be bounded together by 3δ for $\delta \leq 1/10$. This gives a total length of $1 + 5\delta$ for all non-medium jobs. The schedule length for the medium jobs can be bounded as seen above by 2.5δ. □

We need an additional structure for the approximate schedules. Notice that jobs are in general not executed on consecutive processors. This complicates the structure of the solution. On the other hand, we can repack configurations or sets of huge and large jobs on processors such that most of the consecutive processors have the same pattern of starting times and execution times for these jobs. One important step is to exchange processors such that configurations with the same execution time (and same starting time) of a huge job (with $p_j > 1/2 + \delta$) are executed on consecutive processors. In the same way we can modify the schedule such that vertical slices of the same huge job are packed on consecutive processors. This is possible, since each processor contains at most one huge job. The schedule for our example after this modification is given in Figure 2.

All other large and small jobs have to be placed into the remaining free space. Small jobs can be packed with a variant of the asymptotic fully polynomial time approximation scheme (AFPTAS) for

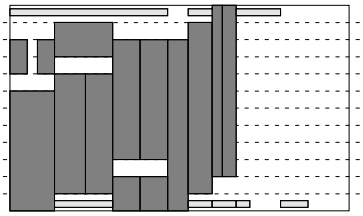

Figure 2: Approximate schedule after vertical exchange steps

2D strip packing by Kenyon and Remila [19] into layers of height δ^2 (see Subsection 2.4). For large jobs we use the following property mentioned in the introduction. If for each time step t, the set $\mathcal{J}_t$ of jobs executed at time t uses at most m processors, then there is a feasible schedule for all large jobs. If this property is fulfilled, we can generate a canonical feasible schedule [17, 16] by stepwise assigning jobs to processors.

2.2 Dynamic Program for Large Jobs

Let us now consider the large jobs. Using the property above, it is sufficient to assign the starting times to the large jobs. By the modification of the schedule for the large jobs, the starting times are multiples $a\delta^2$ where $a \in \{0, \ldots, (1+4\delta)/\delta^2 - 1\}$. Notice that large jobs cannot be started at time $1+4\delta$ or afterwards. Therefore, each schedule can be described as a vector $(m_0, \ldots, m_{(1+4\delta)/\delta^2-1})$, where m_a is the number of processors used at time $a\delta^2$ for $a = 0, 1, \ldots, (1+4\delta)/\delta^2 - 1$. Since the total number of different vectors is too large (i.e. polynomial in m), we first assign starting times to large jobs with width larger than $\lfloor \alpha m \rfloor$ (where α is a constant depending on δ or ϵ specified later). The number of different assignments for all large wide jobs can be bounded by a function $f(\alpha, \delta) = O(1)$, since the area of each large wide job is at least $\delta \lfloor \alpha m \rfloor$ and there are at most $\frac{(1+5\delta)m}{\delta \lfloor \alpha m \rfloor} = \frac{(1+5\delta)}{\delta} \frac{m}{\lfloor \alpha m \rfloor} \leq \frac{2(1+5\delta)}{\alpha\delta}$ of these jobs in the solution. Here we suppose that $m \geq 2/\alpha$ and obtain $\frac{m}{\lfloor \alpha m \rfloor} \leq \frac{2}{\alpha}$. To avoid the dependence on m in the running time, we round down the width of each remaining large narrow job to a multiple of $(\alpha m/n)$ (i.e. a fractional value); this can be done in time polynomial in $\log(n/\alpha)$ and $\log(m)$ for each job. Then, using dynamic programming, we assign a set of jobs with total rounded (fractional) width $b(\alpha m/n)$ (where $b \in \{0, 1, \ldots, n/\alpha\}$) to each possible rounded starting time $a\delta^2$ and each rounded processing time $h\delta^2 \in \{\delta + \delta^2, \ldots, 1/2 + \delta\}$. Notice there are at most $1/(2\delta^2) - 1/\delta \leq 1/(2\delta^2)$ many different rounded large processing times. Each assignment can be described by a vector $v = (v_{a,h})$ where $v_{a,h}$ is the total rounded number of processors used by jobs with starting time $a\delta^2$ and execution time $h\delta^2$. The number of assignments or vectors can be bounded by the polynomial $(n/\alpha + 1)^{O(1/\delta^4)}$ (i.e. independent of m) for each constant δ. All possible vectors can be computed via a dynamic program. Notice that for each approximate schedule S for $\mathcal{J}'$ there exists a vector $v = (v_{a,h})$ where the distance between the value $v_{a,h}$ and the number of processors used in S is at most $n_{a,h}(\alpha m/n)$, where $n_{a,h}$ is the number of large narrow jobs with starting time $a\delta^2$ and rounded execution time $h\delta^2$.

Lemma 2.2 *There exists a vector v with rounded numbers of processors used by large jobs corresponding to an approximate schedule for $\mathcal{J}'$ with length $1 + 5\delta$ such that all large jobs with original processor numbers can be finished until time $1 + 5\delta$ with exception of a subset of large jobs with total width $\leq \lceil (1/\delta^4)\alpha m \rceil$.*

Proof: Compared to the approximate schedule we here make an error of at most αm among all starting times and execution times. Notice that the approximate schedule needs an extra space of at

most $n_{a,h}(\alpha m/n)$ for each starting time $a\delta^2$ and execution time $h\delta^2$. Since $\sum_{a,h} n_{a,h} \leq n$, the error can be bounded by αm. Given a vector v computed by the dynamic program that is componentwise close to the approximate schedule, we can place almost all large narrow jobs with original widths into the reserved space. In addition to the error αm caused by the rounding, for each rounded starting $a\delta^2$ and execution time $h\delta^2$ there could be one additional job of width $\leq \lfloor \alpha m \rfloor$ that does not fit completely into the reserved space of width $v_{a,h}$. The total width of all non-placed large narrow jobs can be bounded by $\lfloor \alpha m \rfloor + (1+5\delta)/\delta^2(1/2)(1/\delta^2)\lfloor \alpha m \rfloor \leq (1/\delta^4)\lfloor \alpha m \rfloor \leq \lceil (1/\delta)^4 \alpha m \rceil$ (using $\delta \leq 1/8$). □

The next step is to delay all huge jobs by at most $1/2 + 4\delta$ such that all huge jobs are finished at the same horizontal level (i.e. they finish all at time $1 + 5\delta$). To avoid an overlap, all other jobs that lie completely above the horizontal line $1/2 + \delta$ are delayed by exactly $1/2 + 4\delta$. Notice that each large job J_j with starting time $a_j\delta^2 \geq 1/2 + \delta$ will be moved up by $1/2 + 4\delta$; i.e. the starting of J_j is delayed by $1/2 + 4\delta$. Therefore, these jobs are started now after time $1 + 5\delta$. This increases the total length of the schedule by at most $1/2 + 4\delta$, but now the structure of the enlarged schedule is easier. The huge jobs can be packed on consecutive processors one by one on the same level. The modified solution for our example is given in Figure 3.

Lemma 2.3 *If there is an approximate schedule for $\mathcal{J}'$ of length at most $1 + 5\delta$, then there exists an approximate schedule of length $3/2 + 9\delta$ for $\mathcal{J}'$ such that all huge jobs finish at time $1 + 5\delta$.*

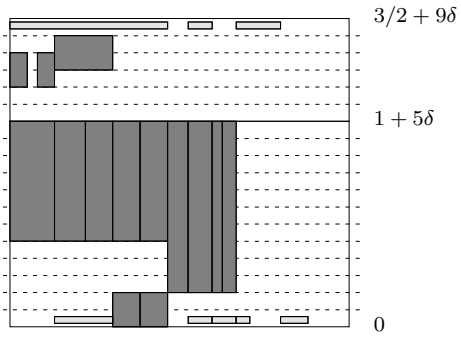

Figure 3: Modified approximate solution after vertical movements

A computed vector v for the large jobs is feasible, if after the transformation of the schedule above for each time step t there are at most m processors used by huge and large jobs.

2.3 Construction of the Gap

In order to reschedule large jobs with total width $\leq \lceil (1/\delta^4)\alpha m \rceil$, we need a gap of height $1/2 + \delta$ and width $\geq \lceil (1/\delta^4)\alpha m \rceil$ in our solution.

Lemma 2.4 *For each optimum schedule of $\mathcal{J}$ with length at most $1 + 2\delta$, there exists an approximate schedule of the non-medium job set $\mathcal{J}'$ with length $3/2 + 9\delta$ such that*

(1) *all huge jobs finish at time $1 + 5\delta$,*

(2) *all huge and large jobs have processing times and starting times that are multiples of δ^2, and*

(3) *there exists a gap of height $1/2 + \delta$ and width $\geq \lfloor (1/8)\delta^4 m \rfloor$ in the schedule.*

Furthermore, we can guess the position of the gap by constant many choices.

Proof: Depending on the structure of the optimum solution, there are four possible places for such a gap.

Case 1: The total width of all jobs with height $> 1/2 + \delta$ is at most $m/2$.

Case 1.1: The total width of all jobs with height $\in (\delta, 1/2+\delta]$ that intersect the horizontal line at height $1/2+(5/2)\delta$ is at most $m/4$. In this case we have a gap of width $\geq m/4$ and height $1/2+(5/2)\delta$ after the vertical movements. We may suppose that the gap lies exactly between the horizontal lines at $1/2 + (5/2)\delta$ and $1 + 5\delta$ (see also Figure 4).

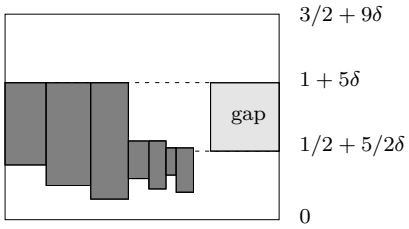

Figure 4: Gap for large jobs in case 1.1

Case 1.2: The total width of all jobs with height $\in (\delta, 1/2 + \delta]$ that intersect the horizontal line at height $1/2 + (5/2)\delta$ is larger than $m/4$. Notice that there are at most $(1 + 5\delta)/(2\delta^2)$ many levels (or different rounded finishing times) of large jobs that cross the horizonal line at $1/2 + (5/2)\delta$. One of these levels at time $\ell\delta^2$ contains large jobs with total width at least $\frac{(m/4)}{(1+5\delta)/(2\delta^2)} \geq (1/4)\delta^2 m$ for $\delta \leq 1/5$. The movement of the jobs above the level generates a gap after time $\ell\delta^2$ of width $\geq (1/4)\delta^2 m$ and height $\geq 1/2 + (5/2)\delta$. Then, the gap can be allocated from time $\ell\delta^2$ to $\ell\delta^2 + 1/2 + (5/2)\delta$ (see also Figure 5). Notice that we can guess the place of the gap in this case (by enumerating at most $(1 + 5\delta)/(2\delta^2) = O(1/\delta^2)$ choices).

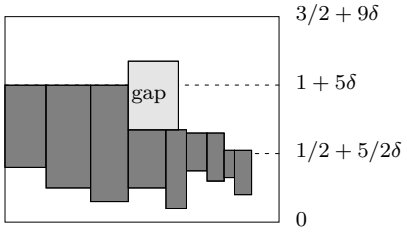

Figure 5: Gap for large jobs in case 1.2

Case 2: The total width of all jobs with height $> 1/2 + \delta$ is larger than $m/2$.

Case 2.1 If there is a huge job J_{j^*} of width $\geq (1/8)\delta^4 m$, then we simply take this job J_{j^*} and guess the starting time of J_{j^*}. In this case we fix the position of J_{j^*} and only move the other huge jobs upwards. This generates a gap of height $\geq 1/2 + (5/2)\delta$ and width $(1/8)\delta^4 m$ above J_{j^*} (for an illustration of this case see Figure 6).

Case 2.2 There is no huge job of width $\geq (1/8)\delta^4 m$. In the optimum solution there are at most $(1 + 5\delta)/\delta^2$ many levels or finishing times of huge jobs with the same height or execution time. Furthermore, the number of different huge rounded execution times is at most $(1/2 + \delta)/\delta^2$. This implies that there are at most $(1 + 5\delta)(1/2 + \delta)/\delta^4$ many blocks of jobs with the same execution and finishing time. Therefore, there is at least one block with width at least $\frac{m/2}{(1+5\delta)(1/2+\delta)/\delta^4} \geq (1/4)\delta^4 m$.

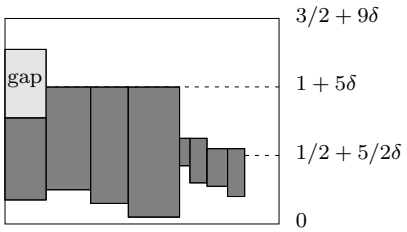

Figure 6: Gap for large jobs in case 2.1

If we do not move this block in the modification of the schedule above, then we have a gap of height $1/2+(5/2)\delta$ and width at least $1/4\delta^4 m$ above the block. Actually we can calculate the position of the gap by guessing the starting time $s\delta^2$ of the block and the execution time $h\delta^2$ of the corresponding jobs.

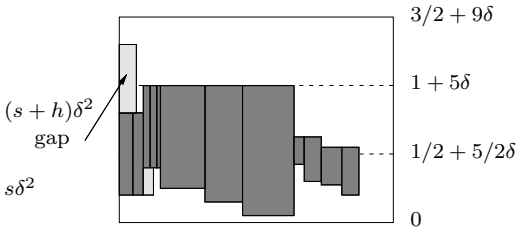

Figure 7: Gaps for large jobs in case 2.2

Now we greedily take a set $\mathcal{J}'$ of jobs each with processing time $h\delta^2$ and width $< (1/8)\delta^4 m$, until the total width $w(\mathcal{J}') \in [(1/8)\delta^4 m, (1/4)\delta^4 m]$. If the total width of all jobs with execution time $h\delta^2$ is smaller than $(1/8)\delta^4 m$, then the guess $h\delta 1^2$ is not correct and can be discarded. Otherwise, all jobs in $\mathcal{J}'$ are started at time s and the remaining huge jobs (with the same execution time) are delayed as described above. In this case we have a gap of height $1/2 + (5/2)\delta$ and width at least $(1/8)\delta^4 m$ just above set $\mathcal{J}'$. To be on the safe side, we only use the width $(1/8)\delta^4 m$ for the gap. Furthermore, for the other calculations, we suppose that a block of width $(1/4)\delta^4 m$ is occupied from time $s\delta^2$ to $1+5\delta-h\delta^2$. This is possible, since the optimum schedule has a block of jobs with total width at least $(1/4)\delta^4 m$ starting at time $s\delta^2$ with execution time $h\delta^2$. For an illustration of this interesting case we refer to Figure 7. □

Now, depending on the guesses including the position and structure of the gap, the positions for the large wide jobs, the vector with the rounded widths of large narrow jobs and the total widths of huge jobs for each rounded execution times, we can compute the total number of free processors for each horizonal layer. Notice that it is sufficient to use the horizontal layers between 0 and $1/2 + \delta$ and between $1 + 5\delta$ and $3/2 + 9\delta$ for the small jobs (see also Figure 8). The transformation of the schedule (i.e. moving jobs upwards and enlarging the schedule to length $3/2 + 9\delta$)) generates more space in the corresponding layers.

Choice of α. We have to reschedule large jobs with total width at most $\lceil(1/\delta^4)\alpha m\rceil$. On the other hand, we have generated a gap of width $\geq \lfloor(1/8)\delta^4 m\rfloor$ above. Notice that $\lceil(1/\delta^4)\alpha m\rceil \leq \lfloor(1/8)\delta^4 m\rfloor$ holds if $(1/\delta^4)\alpha m + 2 \leq (1/8)\delta^4 m$. This inequality holds for $\alpha = (1/16)\delta^8$ and $m \geq 32(1/\delta)^4$. In Subsection 2.2 we suppose that $m \geq 2/\alpha$ (to bound the number of large wide jobs). This implies that m should be larger than $32/\delta^8$.

2.4 How to place the small jobs?

After the placement of the large and huge jobs there are m_ℓ free processors in layer ℓ that can be used for small jobs. Since we do not use the layers between $1/2+\delta$ and $1+5\delta$ and the total number of layers is $(3/2+9\delta)/\delta^2$, the number L of layers for small jobs is equal to $(1+5\delta)/\delta^2$. To simplify the notation, we suppose that m_ℓ is the number of free processors for layer $\ell = 0, \ldots, (1+5\delta)/\delta^2 - 1$ (see also Figure 8 for an illustration). First we check whether the total area of all small jobs is larger than the space left; in this case we discard the corresponding guess for the large jobs and the gap. Otherwise we round the widths of the small wide jobs with widths $> \gamma m$ (where γ depends on δ specified later) up, using a method for 2-dimensional strip packing by Kenyon and Rémila [19]. It generates $G \leq 1/\gamma^2$ different rounded widths $w_1 > \ldots > w_G > \gamma m$. Furthermore, the total area of all jobs after this rounding is at most $(1+5\delta)m(1+\gamma)$.

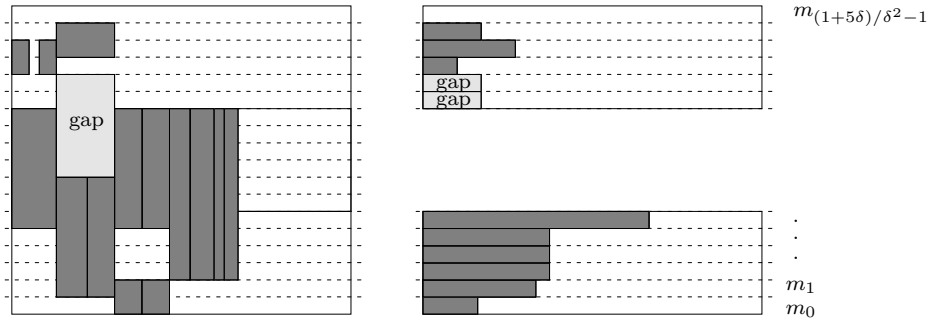

Figure 8: Free Processors for small jobs after locating the gap

Next we set up a linear program for the rounded wide small jobs. First we remove a set $\bar{\mathcal{J}}$ of small wide jobs with the largest numbers of processors required and total execution time within $[\gamma^2 P(\mathcal{J}_{sw}), \gamma^2 P(\mathcal{J}_{sw}) + \delta^5)$ where $\mathcal{J}_{sw}$ is the set of all small wide jobs and $P(\mathcal{J}_{sw}) = \sum_{J_j \in \mathcal{J}_{sw}} p_j$. Then we know that the set $\mathcal{J}_{sw} \setminus \bar{\mathcal{J}}$ of remaining rounded small wide jobs fractionally fits into the horizontal layers (if the guess above corresponds to an approximate schedule). This follows from the rounding technique, since the rounded remaining small wide jobs can be fractionally inserted into the packing for the original small wide jobs [19]. Let n_i be the total execution time of all jobs in $\mathcal{J}_{sw} \setminus \bar{\mathcal{J}}$ with rounded width w_i. Furthermore, let $C_j^{(\ell)}$ be a multiset with rounded wide jobs of total width m_ℓ for layer $\ell = 0, \ldots, L-1$; i.e. $C_j^{(\ell)} = \{a_{j,1}^{(\ell)} : w_1, \ldots, a_{j,G}^{(\ell)} : w_G\}$ such that $\sum_i a_{j,i}^{(\ell)} w_i \leq m_\ell$. Here $a_{j,i}^{(\ell)}$ denotes the number of jobs of width w_i in $C_j^{(\ell)}$. We use a variable $x_j^{(\ell)}$ to indicate the height of the multiset $C_j^{(\ell)}$ in the solution. Suppose that the empty multiset is also allowed. With $\mathcal{C}^{(\ell)}$ we denote the set of all configurations for layer ℓ. Then, the linear program $LP(m_1, \ldots, m_L)$ for a list of n_i jobs of width w_i has the following form:

$$\begin{array}{lcll} \sum_j x_j^{(\ell)} & = & \delta^2 & \ell = 0, \ldots, L-1 \\ \sum_\ell \sum_j x_j^{(\ell)} a_{j,i}^{(\ell)} & \geq & n_i & i = 1, \ldots, G \\ x_j^{(\ell)} & \geq & 0 & \ell = 0, \ldots, L-1, \\ & & & j = 1, \ldots, |\mathcal{C}^{(\ell)}| \end{array}$$

Let ρ be a constant that depends on δ; the exact value is specified later. An ρ-approximate solution of this linear program covering $n_i(1-\rho)$ execution time of width w_i can be computed using the algorithm by Grigoriadis et al. [11], if there is a feasible solution of the LP. The algorithm by Grigoriadis et al. needs $O(G(\log G + 1/\rho^2))$ iterations with $O(1/\delta^2)$ block optimization steps. Each

block optimization consists of an $(1-\rho/6)$ approximation algorithm for a knapsack problem. Since there is an FPTAS for the knapsack problem, the linear program can be solved approximately in time polynomial in $1/\rho$ and G. If the linear program has no feasible approximate solution, then we discard the corresponding guess for the large jobs and the gap. In this case the small wide jobs do not fractionally fit into the horizontal layers.

The number of variables $x_j^{(\ell)} > 0$ in the generated solution is at most $O(1/\delta^2 1/\gamma^2(\log 1/\gamma + \log 1/\rho))$ and can be reduced to $\bar{G} \leq 1/\gamma^2 + (1+5\delta)/\delta^2 \leq 1/\gamma^2 + 2/\delta^2$ by solving a sequence of systems of equalities. Finally we set $\tilde{x}_j^{(\ell)} = x_j^{(\ell)}(1+2\rho)$ for each variable. This implies that $\sum_\ell \sum_j \tilde{x}_j^{(\ell)} a_{j,i}^{(\ell)} \geq (1+2\rho)(1-\rho)n_i \geq n_i$ for $\rho \leq 1/2$. For each layer ℓ we obtain $\sum_j \tilde{x}_j^{(\ell)} = (1+2\rho)\delta^2$. Summing over all layers, the length of the schedule is at most $(3/2+(15/2)\delta)(1+2\rho)$. Similar to the AFPTAS by Kenyon and Rémila [19], the wide small jobs can be placed into the space generated by the solution $\tilde{x}_j^{(\ell)}$ of the LP for the multisets $C_j^{(\ell)}$. The additional height to pack these jobs integrally into a layer ℓ can be bounded by the number of positive variables corresponding to layer ℓ times the maximum height $h_{max} \leq \delta^5$ of a small job. The total increase over all layers is at most $2\bar{G}h_{max} \leq 2\bar{G}\delta^5 \leq 2(1/\gamma^2+2/\delta^2)\delta^5$ where $\bar{G} \leq (1/\gamma^2+2/\delta^2)$ (generating a nice integral packing; see also [14]). The total area $Area(\bar{\mathcal{J}}) = \sum_{J_j \in \bar{\mathcal{J}}} p_j q_j$ can be bounded by $(\gamma^2 P(\mathcal{J}_{sw}) + \delta^5)m \leq (\gamma^2 \frac{Area(\mathcal{J}_{sw})}{\gamma m} + \delta^5)m \leq \gamma Area(\mathcal{J}_{sw}) + \delta^5 m \leq \gamma(1+5\delta)m + \delta^5 m$ (using $Area(\mathcal{J}_{sw}) \geq \gamma m P(\mathcal{J}_{sw})$). These jobs can be executed at the end of the schedule with total length $\leq 2\max[(1+5\delta)\gamma + \delta^5, \delta^5] = 2(1+5\delta)\gamma + 2\delta^5$. Then, the total height of the packing of all jobs (excluding the small narrow ones) is at most

$$\begin{aligned} h' &\leq (3/2+9\delta)(1+2\rho) + 2(1+5\delta)\gamma + (2\bar{G}+2)\delta^5 \\ &\leq (3/2+9\delta)(1+3\gamma) + (2\bar{G}+2)\delta^5, \end{aligned}$$

where $\bar{G} \leq (1/\gamma^2 + 2/\delta^2)$, and $\rho = \gamma/2$. The small narrow jobs with widths $\leq \gamma m$ can be placed greedily with NFDH at the side, leaving only $\leq \gamma m$ processors idle. Suppose that the height h'' after packing the small narrow jobs is larger than h' and that the guess corresponds to an approximate solution. Then we obtain $h'' \leq Area(L_{aux})/(m-\gamma m) + (4\bar{G}+1)h_{max}$, where $h_{max} \leq \delta^5$ and L_{aux} consists of the rounded huge, large and small wide jobs and the original small narrow jobs (see also [14, 19]). Since $Area(L_{aux}) \leq (1+5\delta)m(1+\gamma)$,

$$h'' \leq \tfrac{(1+5\delta)(1+\gamma)m}{(1-\gamma)m} + (4\bar{G}+1)\delta^5 \leq (1+5\delta)(1+3\gamma) + (4\bar{G}+1)\delta^5$$

using $\gamma \leq 1/3$. Therefore, the total height $\max\{h', h''\}$ of the packing for $\mathcal{J}'$ is at most $(3/2+9\delta)(1+3\gamma) + (4\bar{G}+1)\delta^5$. Including the medium jobs, the total height h_{final} can be bounded by $(3/2+9\delta)(1+3\gamma) + (4\bar{G}+1)\delta^5 + 2.5\delta \leq (3/2+11.5\delta) + \delta + 2.5\delta \leq 3/2 + 15\delta$, using $\gamma = \delta/3$ and $\delta \leq 1/10$. Using $\delta \leq \epsilon/15$, the final height $h_{final} \leq 3/2 + \epsilon \leq (3/2+\epsilon)OPT(I)$. This gives an approximation algorithm with ratio $1.5+\epsilon$ for scheduling parallel tasks.

2.5 Faster solution for large jobs

The approach presented above results in an algorithm with running time $n^{f(1/\epsilon)}$ for each $\epsilon > 0$. In this subsection we show how to improve this running time to $f(1/\epsilon) + poly(n)$. All large jobs with width $> \lfloor \alpha m \rfloor$ are placed again via enumeration. Instead of using a dynamic program for the large narrow jobs, we guess the total load of these large jobs assigned to each rounded start time $s\delta^2$ and execution time $h\delta^2$. Let $\Pi^*_{s,h}$ be the total load of large narrow jobs corresponding to the optimum solution. We can guess the vector $\Pi^* = (\Pi^*_{s,h})$ up to an additional error of $\lfloor \alpha m \rfloor$ in each component. To do this we guess all load vectors $\Pi = (\Pi_{s,h})$ where $\Pi_{s,h} = a_{s,h}\lfloor \alpha m \rfloor$ with $a_{s,h} \in \{0, \ldots, 2/\alpha\}$. To ensure that $(2/\alpha)(\alpha m - 1) \geq m$, we use here again the property that $m \geq 2/\alpha$. Notice that the numbers of different

rounded starting times and processing times are bounded by $1/(2\delta^2)$ and $(1+4\delta)/\delta^2$, respectively. The number of all such vectors is bounded by $(2/\alpha+1)^{(1+4\delta)/\delta^2 1/(2\delta^2)} = g(1/\delta)$ for a positive function g using $1/\alpha = p(1/\delta)$, where p is a polynomial. Then, there is a vector Π such that $\Pi_{s,h} \leq \Pi^*_{s,h} \leq \Pi_{s,h} + \lfloor \alpha m \rfloor$ for all s, h. Instead of using Π^*, we use Π for the space reserved for the set $\mathcal{J}_{\ell n}$ of large narrow jobs. Moreover, we set up the following linear program for $\Pi' = \Pi + (\lfloor \alpha m \rfloor, \ldots, \lfloor \alpha m \rfloor)^T$ to determine the positions for most of the large narrow jobs. The linear program uses the variable $y_{j,s}$ for each large narrow job $J_j \in \mathcal{J}_{\ell n}$ and each possible starting time $s\delta^2$.

$$\begin{array}{lcll} \sum_{j:p_j = h\delta^2} y_{j,s} q_j & \leq & \Pi'_{s,h} & h = 1/\delta + 1, \ldots, (1/2+\delta)/\delta^2, s = 0, \ldots, (1+4\delta)/\delta^2 - 1 \\ \sum_{s:s\delta^2 + p_j \leq (1+5\delta)} y_{j,s} & = & 1 & J_j \in \mathcal{J}_{ln} \\ y_{j,s} & \geq & 0 & J_j \in \mathcal{J}_{ln}, s = 0, \ldots, (1+4\delta)/\delta^2 - 1 \end{array}$$

If $\Pi_{s,h} \leq \Pi^*_{s,h} \leq \Pi_{s,h} + \lfloor \alpha m \rfloor$ for all s, h, then there is a fractional (and also integral) solution of the LP corresponding to Π'. The LP for the large narrow jobs can be interpreted as a scheduling problem on a constant number $O(1/\delta^4)$ of unrelated processors [20]. This problem can be solved approximately with ratio $(1+\alpha/2)$ in time $O(n)+g(1/\delta)$, where $1/\alpha = poly(1/\delta)$ and g is exponential in $1/\delta$ [8]. Each approximate solution of the scheduling problem determines positions for large narrow jobs with load bounded by $\Pi_{s,h} + \lfloor \alpha m \rfloor + \alpha m/2 \leq \Pi_{s,h} + 2\lfloor \alpha m \rfloor$ (using $m \geq 2/\alpha$). If we remove for each starting time $s\delta^2$ and execution time $h\delta^2$ a set of large narrow jobs of total load between $2\lfloor \alpha m \rfloor$ and $3\lfloor \alpha m \rfloor$, the remaining jobs fit into the space reserved for the large narrow jobs given by Π. All other jobs have a total load of $(3\lfloor \alpha m \rfloor)[(1+4\delta)/\delta^2][1/(2\delta^2)] \leq 3\lfloor \alpha m \rfloor/\delta^4$, using $\delta \leq 1/4$. If again α is small enough (i.e. if $3\lfloor \alpha m \rfloor/\delta^4 \leq \lfloor (1/8)\delta^4 m \rfloor$ or $3(\alpha m + 1)/\delta^4 \leq (1/8)\delta^4 m - 1$), then all of them can be packed into a gap of height $1/2 + (5/2)\delta$ and width $(1/8)\delta^4 m$. The inequality above holds for $\alpha = (1/48)\delta^8$ and $m \geq 48/\delta^8 + 16/\delta^4$.

Therefore, instead of using the dynamic program we can guess the approximate load vector Π and solve the corresponding scheduling program approximately for Π'. If the linear program has an integral solution, then most of the large narrow jobs can be packed according to Π. The remaining jobs can be packed into the gap as described in Subsection 2.3. This step helps us to reduce the running time. The main algorithm works as follows:

(1) compute a 2-approximate solution for $P|size_j|C_{max}$ and scale the instance such that $OPT(I) \in [1/2, 1]$.

(2) determine the constant δ to partition the jobs into four classes and to discard the medium jobs. Then round the processing times of large and huge jobs to obtain a simplified approximate schedule (see Subsection 2.1).

(3) for each value $T(i) = 1/2 + i\delta$, $i \in \{0, 1, \ldots, 1/(2\delta)\}$ scale the instance again by $T(i)$ and try to find an approximate schedule as follows:

- **(3.1)** guess the starting times for the large jobs with width $> \lfloor \alpha m \rfloor$, an approximate load vector Π for large narrow jobs (see Subsection 2.5) and a position of the gap of height $1/2+(5/2)\delta$ and width $\lfloor (1/8)\delta^4 m \rfloor$ (and in case 2 the position of a huge wide job J^*_j or the position and height of a subset of huge jobs with large width as described in Subsection 2.3).
- **(3.2)** for each possible guess
  - **(3.2.1)** if there is at least one time step with more than m processors used by large and huge jobs (after the transformation of the schedule), then discard the guess above. Otherwise calculate the number of free processors in the layers for the small jobs as

described in Subsection 2.4. If the total area of all small jobs is larger than the free space in the layers, then discard the guess, too.

(3.2.2) solve the linear program to the enlarged load vector Π' for large narrow jobs and solve the linear program for the small wide jobs as described in Subsection 2.4 and 2.5. If both linear programs have a feasible solution, then store the guessed starting times, the approximate load vector, the position of the gap and both feasible solutions. Otherwise discard the guess, too.

(4) for the smallest $T(i)$ where we obtain in step (3.2) a feasible guess, we assign the large jobs according to the load vector and place the remaining non-packed large jobs into the gap. Next we place the huge jobs such that most of them finish at time $1 + 5\delta$; in case 2 either job J_j^* or a subset $\mathcal{J}'$ is placed depending on the guessed starting time.

(5) The small wide jobs are packed into the layers according to the LP solution and the small narrow jobs are packed greedily into the layers (as described in Subsection 2.4). Finally we pack the remaining non-packed jobs at the end of the schedule.

The list scheduling algorithm in step (1) can be implemented in time $O(n \log n)$ by sorting the jobs in non-increasing order of their widths. The running time of our algorithm is dominated by the number of guessing steps (i.e. exponential in $1/\delta$) and the running time to solve the corresponding scheduling problem approximately for each approximate load vector. By merging large narrow jobs together with the same profile, the number of large narrow jobs can be reduced in a pre-processing step to $\min[n, (\log(1/\delta)/\alpha)^{O(1/\delta^4)}]$ [8]. This enables us to solve step (3) of our algorithm in time $O(n) + g'(1/\delta)$. Since the small jobs can be packed into the layers in time $O(n \log n) + poly(1/\delta)$, the total running time of our algorithm for $m \geq 2/\alpha = poly(1/\delta)$ (where δ depends on ϵ [15]) is at most $O(n \log n) + f(1/\epsilon)$.

For $m \leq poly(1/\delta)$, the situation is much easier. First we compute an exact assignment of rounded starting times $s\delta^2$ to all large and huge jobs via dynamic programming in time $m^{O(1/\delta^4)} = 2^{O(1/\delta^4 \log(1/\delta))}$ (as described in [16]). This implies that we can avoid the construction of the gap for the large jobs and the delays for the huge jobs. For each feasible assignment above we calculate the number of free processor in each horizontal layer of height δ^2 and solve the linear program for the small jobs as described in Section 2.4 approximately in $poly(1/\delta)$ time. As above we insert the small jobs into the horizontal layers in $O(n \log n) + poly(1/\delta)$ time. Therefore, in this case we compute a schedule of length at most $(1 + \epsilon)OPT(I)$ in time $O(n \log n) + f(1/\epsilon)$.

3 Scheduling Malleable Tasks

3.1 Guessing steps

First we guess the value of $\delta \in (0, \epsilon/25]$ such that the load of the medium jobs in the optimum solution is small compared to all jobs and $1/\delta$ is integral. This is still possible although we do not know the optimum solution or δ in advance (see also [16]). For simplicity let us suppose that we know δ; in the algorithm similar to [16] we test a constant number of choices for δ. Again we suppose that the number of processors m is larger than a polynomial; the exact bound is specified later. For $m \leq poly(1/\delta)$ we can use the PTAS for $Pm|fctn_j|C_{max}$ [13]. As in the case with parallel tasks, we can scale the instance such that the optimum makespan $OPT(I) \in [1/2, 1]$. To do this we first compute a 2-approximate solution using the algorithm by Turek et al. [26] with value $MAL(I) \leq 2OPT(I)$ and

divide all processing times by $MAL(I)$. Notice that the input length of a problem instance I satisfies $|I| \leq O(\sum_j [|D_j|(\log(m) + \log(p_{max}))])$ and $|I| \geq n + \log(m) + \max_j |D_j| + \log(p_{max})$.

In the first phase we guess the structure of the small wide jobs in an optimum solution. Let γ be a constant that depends polynomially on δ. We guess $1/\gamma^2$ many different small wide jobs with height $\leq \delta^5$ and width $> \lfloor \gamma m \rfloor$. Clearly there is only a polynomial number n^{1/γ^2} of choices. For these chosen jobs J_j we also guess the number of used processors $\ell_j \in D_j \subset M$. Notice that there is only a polynomial number $(\max_j |D_j|)^{1/\gamma^2}$ of possible choices. If there are less than $1/\gamma^2$ small wide jobs in the solution, we simply remove them from the solution and execute them at the end of the schedule. This increases the length of the schedule by at most $1/\gamma^2 \delta^5 \leq \delta$ using $\delta^2 \leq \gamma$.

Otherwise the $1/\gamma^2$ jobs play the role of the rounded values; if we put all small wide jobs ordered by their widths on a stack (see Figure 9), then the $1/\gamma^2$ jobs form so called threshold rectangles. The corresponding rounding idea was used by Kenyon and Rémila [19] to obtain an AFPTAS for 2D strip packing. We sort the widths of the chosen jobs such that $w_1 \geq w_2 \geq \ldots \geq w_{1/\gamma^2}$. Let H be the total height H of all small wide jobs in an optimum solution. We guess the approximate total height H_{app} up to a multiple of γ^2; i.e. $H_{app} = c\gamma^2 \in [H - \gamma^2, H]$. Since the optimum makespan $OPT(I) \in [1/2, 1]$ and each small wide job has width $> \lfloor \gamma m \rfloor$, the total height H of all small wide jobs corresponding to an optimum solution is bounded by $m/\gamma m \leq 1/\gamma$. This implies that the number of choices for c is at most $1 + 1/\gamma^3$. Removing the widest small jobs with total height at most $\gamma^2 + \delta^5 \leq \delta$ from the optimum solution generates a solution where the total height of the small wide jobs is at most H_{app}. By removing also the guessed small wide jobs, we obtain $1/\gamma^2 - 1$ groups of jobs with each group of total height $\leq \gamma^2 H_{app}$. The removed jobs have total height at most $1/\gamma^2 \delta^5 \leq \delta$. Considering only different widths, we obtain a sequence $w'_1 > w'_2 > \ldots > w'_G$ with $G \leq 1/\gamma^2 - 1$, where each rounded width w'_i occurs with total height $c_i \gamma^2 H_{app}$ in our solution where $c_i \in \{1, \ldots, 1/\gamma^2 - 1\}$. For simplicity, let w'_0 be the maximum original guessed width w_1. An example with the stack for the original small wide jobs and the stack for the rounded small wide jobs is given in Figure 9.

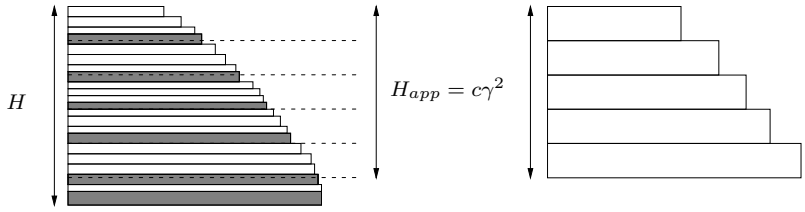

Figure 9: Stack with small wide jobs corresponding to an optimal solution

Now we guess for each horizontal layer ℓ of height δ^2 in our solution the approximate number $\beta^{(\ell)}$ of processors or approximate width used by small jobs. Since small jobs are packed into the layers fractionally, it is sufficient to consider here a relaxation of the solution and to allow that small jobs are divided into horizontal slices and packed into the layers. The underlying idea here is a linear program relaxation for the small jobs. The width of a layer for small jobs is determined by the maximum width $maxwidth^{(\ell)}$ of a configuration with small wide jobs (i.e. given by a subset of small wide jobs) in it plus an integral multiple of γm (i.e. a fractional value) to reserve approximately some additional space for small narrow jobs. For each horizontal layer, the number of guesses is at most $n^{1/\gamma}(1/\gamma + 1)$. Therefore, the total number of guesses here is polynomial in n.

Lemma 3.1 *Suppose that there is an approximate schedule for $\mathcal{J}'$ (with rounded processing and starting times for huge and large jobs, but without medium jobs) with length $\leq 1 + 5\delta$. Furthermore, let $\beta^{(\ell)}_{org}$ be the original number of processors used by small jobs in layer ℓ, for $\ell = 0, \ldots, (1 + 5\delta)/\delta^2 - 1$.*

Then there exists also an approximate schedule for almost all jobs in $\mathcal{J}'$ with length $\leq (1+5\delta)$ that only uses $\beta^{(\ell)}$ processors for small jobs in layer ℓ where $\beta^{(\ell)} = maxwidth^{(\ell)} + \alpha_\ell \gamma m \in [\beta^{(\ell)}_{org} - \gamma m, \beta^{(\ell)}_{org}]$ and $\alpha_\ell \in \mathbb{Z}^+$. The non-scheduled jobs are all small narrow and have total area $\leq 2\gamma m(1+5\delta)$.

Proof: Consider an approximate schedule as described above. Using the rounding of the starting and processing times, the large and huge jobs use $\alpha^{(\ell)}_{org}$ processors during the entire layer ℓ. The remaining $\beta^{(\ell)}_{org} = m - \alpha^{(\ell)}_{org}$ processors execute small jobs in the layer. Let $maxwidth^{(\ell)}$ be the maximum number of processors that execute small wide jobs at one time step in layer ℓ. Then, we can set $\beta^{(\ell)} = maxwidth^{(\ell)} + \alpha_\ell \gamma m$ with $\alpha_\ell \in \mathbb{Z}_+$ where $\beta^{(\ell)}_{org} - \gamma m \leq \beta^{(\ell)} \leq \beta^{(\ell)}_{org}$. Next we transform the original approximate schedule into one that uses only $\beta^{(\ell)}$ processors for small jobs.

For simplification we suppose that there are no idle processors at any time step in layer ℓ; otherwise we simply insert some dummy small narrow jobs. Starting from the first time step $t = \ell\delta^2$ in layer ℓ, we remove a set of small narrow jobs S which are executed at time t until the total number of processors used by S is within the interval $[\beta^{(\ell)}_{org} - \beta^{(\ell)}, \beta^{(\ell)}_{org} - \beta^{(\ell)} + \gamma m]$. Then, let $t' > t$ be the first time step where less than $\beta^{(\ell)}_{org} - \beta^{(\ell)}$ processors are used by jobs in S. This implies that at least one small narrow job has been finished just before. Since the small wide jobs use at most $maxwidth^{(\ell)}$ processors at time t' and there are no idle processors, there exist a set S' of small narrow jobs starting at t' such that $S \cup S'$ uses at least $\beta^{(\ell)}_{org} - \beta^{(\ell)}$ processors at time t'. Therefore, we can simply add a subset $\bar{S}$ of S' to S such that the number of processors used by $\bar{S} \cup S$ is within $[\beta^{(\ell)}_{org} - \beta^{(\ell)}, \beta^{(\ell)}_{org} - \beta^{(\ell)} + \gamma m]$. By iterating this process we reduce the number of processors used at each time step by $\leq \beta^{(\ell)}_{org} - \beta^{(\ell)} + \gamma m \leq 2\gamma m$. Among all layers we remove in this process a set of small narrow jobs with total area $\leq 2\gamma m(1+5\delta)$. □

If we additionally remove the widest group of rounded small wide jobs with height $\gamma^2 H_{app}$ (i.e. we set $\bar{c}_1 = c_1 - 1$ and $\bar{c}_i = c_i$ for $i \geq 2$), then this generates an approximate solution for the remaining original small wide jobs that fit fractionally into the layers. Let $\beta^{(\ell)}$ be the guessed width for layer ℓ. Via a linear program (similar to Subsection 2.4)

$$\begin{array}{lcll} \sum_j x^{(\ell)}_j & = & \delta^2 & \ell = 0, \ldots, (1+5\delta)/\delta^2 - 1 \\ \sum_\ell \sum_j x^{(\ell)}_j a^{(\ell)}_{j,i} & \geq & \bar{c}_i \gamma^2 H_{app} & i = 1, \ldots, G \\ x^{(\ell)}_j & \geq & 0 & \ell = 0, \ldots, (1+5\delta)/\delta^2 - 1, \\ & & & j = 1, \ldots, |\mathcal{C}^{(\ell)}| \end{array}$$

with variables $x^{(\ell)}_j$ for configurations $C^{(\ell)}_j$ with total width at most $\beta^{(\ell)}$ (i.e. a multiset $\{a_{j,1} : w'_1, \ldots, a_{j,G} : w'_G\}$ with $\sum_i a^{(\ell)}_{j,i} w'_i \leq \beta^{(\ell)}$) we can test whether the rounded small wide jobs fit fractionally into the horizontal layers of widths $\beta^{(0)}, \ldots, \beta^{(\frac{1+5\delta}{\delta^2}-1)}$. The linear program has a constant number of variables and constraints, where the number of variables is exponential in $1/\gamma$ and the number of constraints is polynomial in $1/\delta + 1/\gamma$. Moreover, the coefficients $a^{(\ell)}_{j,i}$ in the LP are bounded by the constant $1/\gamma$ (using $w'_i \geq \gamma m$ and $\beta^{(\ell)} \leq m$). The linear program can be solved exactly in time polynomial in the number of variables and constraints (e.g. using the algorithm by Vaidya [27]). Notice that we do not select here which jobs are small, we only test whether rounded small wide jobs of width w'_i and total height $\bar{c}_i \gamma^2 H_{app}$ fit into the horizontal layers fractionally.

Next we guess all jobs and their rounded starting and execution time in our solution with height $\in (\delta, 1+2\delta]$ and width $> \lfloor \alpha m \rfloor$. For each guessed job J_j with rounded execution time $a\delta^2$ we choose the minimum number of processors $\ell \in D_j$ such that $p_j(\ell) \in ((a-1)\delta^2, a\delta^2]$; if there is no such ℓ then this guess is infeasible and can be discarded. We guess also which jobs are scheduled as medium jobs with large width $> \lfloor \alpha m \rfloor$ (here we choose as width a number of processors $\ell \in D_j$ such that $p_j(\ell) \in (\delta^5, \delta]$ with minimum area $\ell p_j(\ell)$; if there is no such ℓ the guess can be discarded). Since we have at most

$\leq (1+5\delta)/(\delta^5\alpha)$ many such jobs, this can be done in polynomial time. If the total area of all guessed medium wide jobs is larger than $\delta(1+2\delta)m$, then we also discard the corresponding guess. Finally, we guess for each rounded huge execution time $h\delta^2 \in (1/2+\delta, 1+5\delta]$ and each rounded starting time $s\delta^2$ a constant number $K = 3\bar{K} \leq 6(1/\delta^2 + 1/\gamma^2)$ of jobs with the largest width (the width is determined as the minimum number of processors used as above). Here $\bar{K} = (1+5\delta)/\delta^2 + 1/\gamma^2 + 2$ is an upper bound for the number of fractional variables in a linear program described in the next subsection. All other jobs assigned to the same rounded execution time h and starting time s need a width smaller than or equal to the smallest guessed width $\bar{w}_{h,s}$. If the total number of processors in a layer used by large and huge jobs is larger than m, then we also discard the guess.

Similar to the case with parallel jobs, we could guess also the total approximate load of all large jobs for each rounded execution time and rounded starting time. But as we see below, this does not help us to determine the free space for the huge jobs. In fact, it is not possible to determine the exact structure consisting of the total width for each rounded huge processing time and each rounded starting time. This is one of the main difficulties here. There could be different possible scenarios corresponding to the same load values (see also Figure 10).

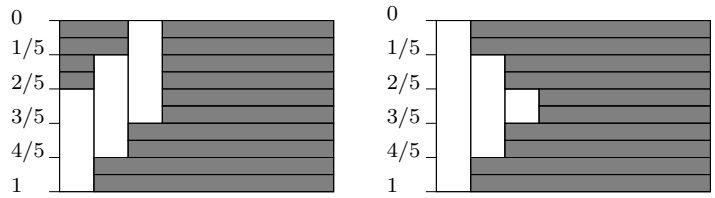

Figure 10: Different scenarios for the same remaining processor numbers

3.2 Linear Program

We solve this difficulty via a linear program that implicitly determines the corresponding best possible scenario. We use variables $x_{j,i}^{(huge)}$, $x_{j,i}^{(large)}$, $x_{j,i}^{(sw)}$, $x_j^{(sn)}$, and $x_j^{(m)}$ to indicate whether a job J_j is executed as huge, large, small wide, small narrow or medium job (here index i indicates either the rounded execution time $i\delta^2$ for huge and large jobs or the rounded width w'_i). Furthermore, we use a variable $y_{j,i,k}$ to indicate the starting time $k\delta^2$ of a large or huge job J_j with running time $i\delta^2$.

For each job J_j executed as a huge or large job with rounded execution time $i\delta^2 \in (\delta, 1+5\delta]$, let $A_{j,i}$ be the minimum number of processors $a \in D_j$ used by J_j with running time $p_j(a) \in ((i-1)\delta^2, i\delta^2]$ (this gives the smallest number of processors with such a rounded processing time). If there is no such $a \in D_j$ with this property, then we set the corresponding variable $x_{j,i}^{(h)} = 0$ or $x_{j,i}^{(\ell)} = 0$. For each job J_j executed as a small wide job with rounded processor number w'_i for $i \in \{1, \ldots, G\}$, let $B_{j,i}$ be the minimum processing time $p_j(a)$ over all processor numbers $a \in (w'_{i+1}, w'_i] \cap D_j$ (this gives the smallest execution time or height in the stack for the 2-dimensional strip packing). Again, if there is no such a then we set $x_{j,i}^{(sw)} = 0$.

In addition we allow that some jobs are small and very wide (these are the jobs in the last group of the original stack). For a job J_j in this group, let $B_{j,0}$ be the minimum processing time $p_j(a)$ over all processor numbers $a \in D_j \cap [w'_1, m]$ (for $D_j \cap [w'_1, m] = \emptyset$ we set $x_{j,0}^{(sw)} = 0$). For J_j executed as a medium (with processing time in $(\delta^5, \delta]$) or small narrow job (with number of processors $\leq \lfloor \gamma m \rfloor$ and processing time $\leq \delta^5$), let $C_{j,m}$ (and $C_{j,sn}$) be the smallest area $ap_j(a)$ over all possible feasible choices of $a \in D_j$ such that $p_j(a)$ is medium and ($p_j(a)$ is small and a is narrow), respectively. Again, if there is no such $a \in D_j$ then we set $x_j^{(m)} = 0$ or $x_j^{(sn)} = 0$.

Let Π_ℓ be the number of processors used by all pre-placed large and huge jobs in the horizontal layer ℓ plus the approximate value $\beta^{(\ell)}$ for the small wide jobs. The remaining $m - \Pi_\ell$ processors can be used for other large and huge jobs. Finally, let $\mathcal{J}''$ be the set of jobs which are not pre-placed in our guessing steps. The guessed small wide jobs are not considered in the LP below; they are packed at the end of the schedule.

The linear program for the malleable jobs has the following form.

$$
\begin{array}{lcll}
\sum_{k:(k+i)\delta^2 \le 1+5\delta} y_{j,i,k} & = & x_{j,i}^{(huge)} & \forall J_j \in \mathcal{J}'' \forall i \\
\sum_{k:(k+i)\delta^2 \le 1+5\delta} y_{j,i,k} & = & x_{j,i}^{(large)} & \forall J_j \in \mathcal{J}'' \forall i \\
\sum_{j,i,k:k\le\ell<k+i} y_{j,i,k} A_{j,i} & \le & m - \Pi_\ell & \forall \ell = 0, \ldots, \frac{1+5\delta}{\delta^2} - 1 \\
\sum_j x_{j,i}^{(sw)} B_{j,i} & \le & c_i \gamma^2 H_{app} & \forall i = 1, \ldots, G \\
\sum_j x_{j,0}^{(sw)} B_{j,0} & \le & \gamma^2 & \\
\sum_j x_j^{(m)} C_{j,m} & \le & \lceil \delta(1+2\delta)m \rceil & \\
\sum_{j,i} x_{j,i}^{(sw)} B_{j,i} w'_i + \sum_j x_j^{(sn)} C_{j,sn} & \le & \sum_\ell \beta^{(\ell)} \delta^2 + 4\gamma m & \\
\sum_i x_{j,i}^{(huge)} + \sum_i x_{j,i}^{(large)} + \sum_i x_{j,i}^{(sw)} + x_j^{(sn)} + x_j^{(m)} & = & 1 & \forall J_j \in \mathcal{J}'' \\
x_{j,i}^{(huge)}, x_{j,i}^{(large)}, x_{j,i}^{(sw)}, x_j^{(sn)}, x_j^{(m)} & \ge & 0 & \forall J_j \in \mathcal{J}'' \forall i \\
y_{j,i,k} & \ge & 0 & \forall J_j \in \mathcal{J}'' \forall i, k
\end{array}
$$

In addition we set a variable $y_{j,i,k} = 0$, if the width $A_{j,i}$ of job J_j with rounded processing time $i\delta^2$ and starting time $k\delta^2$ is larger than the smallest pre-assigned width $\bar{w}_{i\delta^2,k\delta^2}$. Furthermore, we set $y_{j,i,k} = 0$ for $i\delta^2 \le \delta$. If a job J_j is chosen as a medium wide job in the guessing step above, we set $x_j^{(m)} = 1$. The additive term in the last inequality is given by the additional space that we need to place all small jobs.

Lemma 3.2 *Suppose that there is an approximate schedule $\mathcal{S}$ for $\mathcal{J}'$ (with rounded processing and starting times for huge and large jobs, but without medium jobs) with length $\le 1+5\delta$ where the medium jobs in $(\mathcal{J} \setminus \mathcal{J}')$ have total area at most $\delta(1+2\delta)m$. Furthermore, let $\beta^{(\ell)}$ be the approximate number of processors in $\mathcal{S}$ used by small jobs in layer ℓ, for $\ell = 0, \ldots, (1+5\delta)/\delta^2 - 1$.*

Then there also exists a feasible solution of the linear program for the non-guessed jobs in $\mathcal{J}''$ (including medium jobs).

Proof: We can simply set $x_{j,i}^{(huge)} = 1$, $x_{j,i}^{(large)} = 1$ if a job J_j is executed in $\mathcal{S}$ as a huge or large one with processing time $i\delta^2$, respectively. Furthermore we set $y_{j,i,k} = 1$ if the job is started at time $k\delta^2$. Then, the equalities for the large and huge jobs are satisfied. Furthermore, $\sum_{j,i,k:k\le\ell<k+i} y_{j,i,k} A_{j,i}$ gives a lower bound for the number of processors used by large and huge jobs in layer ℓ. Since $A_{j,i}$ is the minimum number of processors a used by J_j with execution time $p_j(a) \in ((i-1)\delta^2, i\delta^2]$, $A_{j,i}$ is smaller than or equal to the number of processors used by J_j in $\mathcal{S}$. Since there are at most $m - \Pi_\ell$ processors available in layer ℓ (after the pre-placement), the inequalities for the layers are true.

For a job J_j executed as a small narrow or small wide job, we set $x_j^{(sn)} = 1$ or $x_j^{(sw)} = 1$, respectively. If we put all small wide jobs ordered by their width on a stack, then the total height H lies in the interval $[H_{app}, H_{app} + \gamma^2]$. The guessed small wide jobs (which are not in $\mathcal{J}''$) give us the rounded values $w'_0 \ge w'_1 > \ldots > w'_M$. Small jobs J_j which use $a_j \in (w'_{i+1}, w'_i] \cap D_j$ processors for $i \ge 1$ in our schedule can use up to w'_i processors in the linear program (i.e. the rounded number in the stack is equal to w'_i). The total height (or total number of processors) of all these jobs in the stack is exactly $\bar{c}_i \gamma^2 H_{app} = c_i \gamma^2 H_{app}$ for $i \ge 2$. On the other hand $B_{j,i} = \min_{a \in (w'_{i+1}, w'_i] \cap D_j} p_j(a) \le p_j(a_j)$. Therefore, we get $\sum_j x_{j,i}^{(sw)} B_{j,i} \le \sum_j x_{j,i}^{(sw)} p_j(a_j) \le c_i \gamma^2 H_{app}$ for $i \ge 2$. For $i = 1$ we have an additional allowed

height of $\gamma^2 H_{app}$ for the largest rounded width w'_1. Finally for $i = 0$ we have a height of at most γ^2 for the last group in the original stack.

For the inequality considering the total area, the original small wide jobs fit into the layers with the original widths $\beta^{(\ell)}_{org}$. Since each group in the stack has the same height $\gamma^2 H_{app}$, the rounded small wide jobs (excluding the widest group in the stack of height $\gamma^2 H_{app}$) also fit into these layers fractionally. The widest group in the stack has total area at most $m\gamma^2 H_{app} \leq m\gamma$. Therefore, the total area $\sum_{j,i\geq 1} x^{(sw)}_{j,i} B_{j,i} w'_i + \sum_j x^{(sn)}_j C_{j,sn} \leq \sum_\ell \beta^{(\ell)}_{org}\delta^2 + m\gamma$. Using Lemma 3.1 the right hand side is bounded by $\leq \sum_\ell [\beta^{(\ell)} + \gamma m]\delta^2 + m\gamma \leq \sum_\ell \beta^{(\ell)}\delta^2 + m\gamma + (1+5\delta)m\delta^2\gamma/\delta^2 \leq \sum_\ell \beta^{(\ell)}\delta^2 + 3m\gamma$. As seen above the area of the widest groups $\sum_{j,i=0} x^{(sw)}_{j,i} B_{j,i} w'_i$ is at most $\gamma^2 m \leq \gamma m$. Therefore, the inequality corresponding to the total area is satisfied.

Using the choice of δ, the medium jobs have a total area of $\delta(1+2\delta)m$. Therefore, we can simply set $x^{(m)}_j = 1$ if a job is executed as a medium one. Since $C_{j,m}$ is the minimum area among all feasible number of processor numbers $a \in D_j$, using the choice of δ the inequality $\sum_j x^{(m)}_j C_{j,m} \leq \delta(1+2\delta)m$ is satisfied. □

3.3 How to handle the fractional jobs?

In total we here have a linear program with $\leq \bar{K} = (1+5\delta)/\delta^2 + 1/\gamma^2 + 2 \leq 2(1/\delta^2 + 1/\gamma^2) = O(1/\delta^2 + 1/\gamma^2)$ (using $\delta \leq 1/5$ and $\gamma \leq 1/2$) inequalities (not counting the non-negativity constraints) plus one equality for each non-preassigned job. Actually we can replace the variables $x^{(huge)}_{j,i}$, $x^{(large)}_{j,i}$ in the equalities with the sum of the corresponding variables $y_{j,i,k}$. Let n' be the number of non-preassigned jobs. A basic feasible solution of the LP has at most $n' + \bar{K}$ variables with strict positive value. Since we need at least one positive variable for each job, the number of fractional variables can be bounded by $\bar{K} \leq O(1/\delta^2 + 1/\gamma^2)$. The jobs with integral values can be placed according to the LP values. Jobs that have a fractional amount $x^{(sn)}_j \in (0,1)$ or $x^{(sw)}_{j,i} \in (0,1)$ (like small jobs) can be eliminated and scheduled at the end. This gives $\leq 2(1/\delta^2 + 1/\gamma^2)\delta^5 \leq 4(1/\delta^4)\delta^5 \leq 4\delta$ as additional height. Jobs that have a fractional amount $x^{(m)}_j \in (0,1)$ (e.g. a fractional medium job) have a total area $\leq \alpha m \delta 2(1/\delta^2 + 1/\gamma^2) \leq \delta m$ if $\alpha \leq (1/4)\delta^4$ is small enough. Here αm is the largest width and δ is the largest execution time of a medium job, respectively. The list scheduling algorithm generates a schedule for these jobs with height $\leq 2\max[\delta m/m, \delta] = 2\delta$. Here we use the assumption that medium jobs of large width are guessed before.

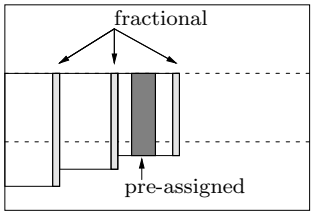

Figure 11: Placement of pre-assigned and fractional huge jobs

Jobs that have a fractional amount $x^{(large)}_{j,i} \in (0,1)$ with $i\delta^2 \leq 1/2 + \delta$ (i.e. fractional large jobs) can be grouped into a block of height $1/2+\delta$ and width $\leq 2(1/\delta^2+1/\gamma^2)\cdot\lfloor\alpha m\rfloor \leq \delta^5 m$ for $\alpha \leq (1/4)\delta^9$ and $\delta^2 \leq \gamma$. Since the total width is integral, the width of the block can be bounded by $\lfloor\delta^5 m\rfloor$. These jobs can be packed into a gap similar to our approach for parallel jobs. The main difficulty now are the remaining fractional huge jobs (i.e. jobs with $x^{(huge)}_{j,i} \in (0,1)$ and $i\delta^2 > 1/2 + \delta$). All other jobs have either integral corresponding variables or are removed in the step above. By vertical exchanges

of processors and corresponding configurations and moving all huge jobs up similar to the variant with parallel jobs, all huge jobs or their fractional parts finish at the same time $1 + 5\delta$ (see Figure 11). In Figure 11 we have three fractional parts of a huge job with different execution times and one indicated pre-assigned job.

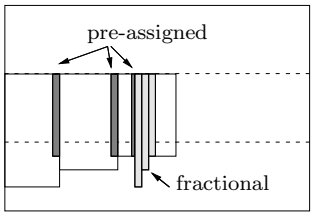

Figure 12: The solution after exchanging fractional and pre-assigned huge jobs

Huge jobs that have fractional amounts for different rounded execution times have to be grouped together. We have to choose one of the corresponding processor numbers and to modify the generated schedule. For each such job J_j, we choose the smallest width m_j (i.e. the number of processors) used among its fractional parts. Let k_j be the corresponding rounded processing time with width m_j and let b_j be the smallest processing time among all fractional parts for J_j. By exchanging with a part of a pre-assigned job J^* of larger width (but with the smallest possible one) in the group corresponding to b_j, all fractional parts of J_j can be placed together into the group with processing time b_j. The exchanged parts of J^* are placed instead of the fractional parts into the other groups all finishing at time $1 + 5\delta$ (see Figure 12 for an illustration of our example above; fractional parts of the huge job and parts of the pre-assigned huge job are exchanged).

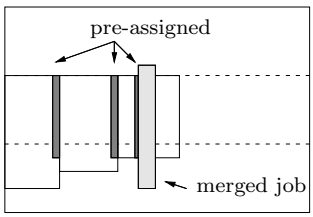

Figure 13: Solution after merging the fractional parts of a huge job

The fractional parts of J_j are merged together as one job with processing time k_j; that results in a width smaller than or equal to the sum of the fractional parts. But now the height of this job could be larger than the processing time b_j. Since J^* was pre-assigned, we can move J_j downwards to the starting time of J^* (see also Figure 13 where the merged job is moved downwards). Since we have a free space of height at least $1 + 5\delta$ just above the starting time of J^*, the merged job J_j fits into this gap. Since K is larger than the number of fractional variables, we can do this exchange step for all fractional jobs one by one. Notice that J^* is now fractionally assigned to different groups. On the other hand, we do not change the starting or finishing time of J^*. Furthermore, we do not increase the number of processors used for J_j before we merge the parts of J_j together. This implies that the starting times still satisfy the sufficient condition to have a feasible schedule. Therefore, we are still able to generate a canonical schedule.

3.4 How to generate the gap ?

Similar to the scheduling problem with parallel tasks we consider different cases to locate the gap of height $1/2 + (5/2)\delta$ and width at most $\delta^5 m$ for the fractional large jobs. We consider again the situation before we delay some jobs.

Case 1: The total width of all huge jobs is at most $m/2$.

Case 1.1: The total width of all large jobs intersecting with the horizontal line at $1/2 + (5/2)\delta$ is at most $m/4$. In this case we have a gap of width $m/4$ as before.

Case 1.2: The total width of all large jobs intersecting with the horizontal line at $1/2 + (5/2)\delta$ is larger than $m/4$. Then, there is again at least one level with width $\geq \frac{m/4}{(1+5\delta)/(2\delta^2)} \geq (1/4)\delta^2 m$ for $\delta \leq 1/5$. This generates a gap of width $(1/4)\delta^2 m$ just above one group of large jobs finishing at the same level.

Case 2: The total width of all huge jobs with height $> 1/2 + \delta$ is larger than $m/2$.

Case 2.1 There is a huge job J_{j^*} of height $> 1/2 + \delta$ and width $\geq (1/8)\delta^4 m$ (if there are several such jobs we take the widest one). Such a job is pre-assigned if $(1/8)\delta^4 m > \alpha m$ or equivalently $\alpha < (1/8)\delta^4$. We have chosen α small enough such that this is fulfilled. Since K is large enough, the job J_{j^*} is not touched in the exchange step before (since we use there jobs with the smallest width). Then we simply take this job J_{j^*} and do not modify the starting time of J_{j^*} given by the pre-assignment. After moving the other jobs above J_{j^*} by $1/2 + (5/2)\delta$, we obtain a gap of height $1/2 + (5/2)\delta$ and width $(1/8)\delta^4 m$ just above J_{j^*}.

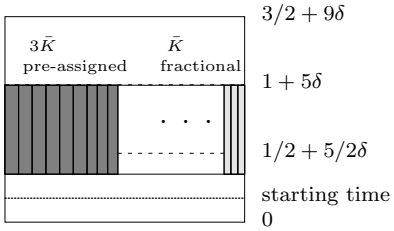

Figure 14: A level with $3\bar{K}$ pre-assigned wide jobs and some fractional jobs

Case 2.2: There is no huge job of width $\geq (1/8)\delta^4 m$. In this case there is at least one finishing time or level with jobs of the same height and total width $\geq \frac{m/2}{(1+5\delta)(1/2+\delta)/\delta^4} \geq (1/4)\delta^4 m$.

Consider this level of jobs with the same finishing time and same height in more detail. For an illustration with $3\bar{K}$ pre-assigned wide jobs and some fractional jobs on such a level we refer to Figure 14. Notice that it contains at least 3 times the number of fractional variables (i.e. $K = 3\bar{K}$ and $\bar{K} = (1 + 5\delta)/\delta^2 + 1/\gamma^2 + 2 \leq O(1/\delta^2 + 1/\gamma^2)$) of pre-assigned jobs with larger width than the other assigned jobs. And suppose that we choose only pre-assigned jobs (at most $\bar{K}$ many) with the smallest width in the exchange step. These jobs or parts of them are replaced by other maybe longer merged jobs. In addition there are $\leq \bar{K}$ fractional jobs in this block. Some of them stay here and are merged later with other fractional ones and some of them are replaced by pieces of pre-assigned jobs of other blocks. The total width of these $\leq 2\bar{K}$ exchanged jobs after the replacement is at most the width of the $2\bar{K}$ wider jobs. Therefore, the largest $2\bar{K}$ pre-assigned jobs plus the jobs with corresponding integral LP values cover at least $1/2$ of the entire width of the block. This gives a block of width $\geq \lfloor (1/8)\delta^4 m \rfloor$. We can move this block and the corresponding jobs to the starting time given by the LP. This gives us a gap of $1/2 + (5/2)\delta$ and width $\geq \lfloor (1/8)\delta^4 m \rfloor$ just above this block (see also Figure 15). If $\delta \leq 1/8$ then $\delta^5 m \leq (1/8)\delta^4 m$. This implies that $\lfloor \delta^5 m \rfloor \leq \lfloor (1/8)\delta^4 m \rfloor$. Therefore, this gap can be used to insert the block with all fractional large jobs.

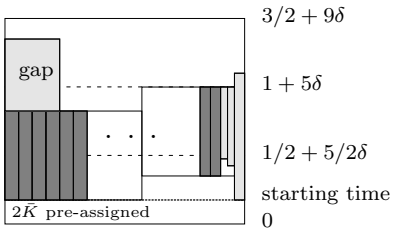

Figure 15: Using a wide level to locate the gap

The guessed small wide jobs, the fractional small narrow and small wide and the fractional and integral medium jobs generate an additional height of at most $\delta + 4\delta + 4.5\delta \leq 10\delta$. Notice that the height for the integral medium jobs $2\max[\frac{\lceil\delta(1+2\delta)m\rceil}{m}, \delta]$ is at most 2.5δ using $m \geq 1/\delta^2$ and $\delta \leq 1/25$. Furthermore, the approximate guess of the stack height, rounding of the widths of the small wide jobs and modifying the total width for small jobs in each layer (including one additional small job) gives an additional area of at most $4\gamma m + \delta^5 m \leq 5\delta^2 m$ using $\gamma = \delta^2$. List scheduling gives us here an additional height of at most $10\delta^2 \leq \delta$ using $\delta \leq 1/10$. Similar to the variant with parallel tasks almost all small wide and narrow jobs can be inserted into the horizontal layers. This step costs an additional height of $(4\bar{G} + 1)\delta^5 \leq 5\delta$ where $\bar{G} \leq 1/\gamma 2 + 2/\delta^2$ (see also Section 2.4).

Therefore, the final length of our schedule can be bounded by $(3/2+9\delta)+16\delta \leq 3/2+25\delta \leq 3/2+\epsilon$ using $\delta \leq (1/25)\epsilon$. The running time of the algorithm is dominated by the number of guesses multiplied with the running time to solve the linear program above. The number of variables and constraints are both bounded by $O(n(1/\delta^4 + 1/\gamma^2)) = O(n^2)$. Furthermore, the coefficients of the (in-)equalities are at most $O(mp_{max}/\gamma^5)$ (in order to get integral coefficients some of the inequalities are multiplied by $1/\gamma^4$, $1/\gamma^2$ or $1/\gamma$). Since a linear program with N variables and M constraints can be solved in time $O(((M+N)N^2 + (M+N)^{1.5}N)L)$ [27], where L is the length of the input, the running time to solve our LP can be bounded by $O(n^5(\log(m + p_{max})))$. Notice that L can be bounded by $O(n^2/\delta^8 \log(mp_{max}/\gamma)) = O(n^2 \log(m + p_{max}))$. In addition, the number of guesses is polynomial in $\max|D_j|$ and n for any fixed $\delta > 0$. Therefore, the total running time is polynomial in $n, \max|D_j|$ and $\log(p_{max} + m)$ (i.e. polynomial in the input length of the instance).

4 Additional Release Dates

Notice that our approach can also handle additional release dates. First, we can round the release dates down to a multiple of δ such that there is only a constant number $\Theta(1/\delta)$ of them as described by Hall and Shmoys [12]. We have to place jobs in the guessing steps and linear programs according to the rounded release dates. In the exchange steps for malleable tasks, we exchange fractional parts only by pre-assigned jobs with the same rounded starting time. Note that large jobs placed into the constructed gap get only larger starting times. The difficulty are the small wide and narrow jobs. Here we round separately the sets of small wide jobs released at each rounded release date and modify the linear program to reserve enough space for small wide and narrow jobs for each interval $[i\delta, (i+1)\delta)$. To do this we guess (according to an approximate schedule) the total execution time of small wide jobs for each rounded width w_i and each interval up to a multiple of δ^5. This gives us also an approximate area for the small narrow jobs executed in each interval. Finally we delay the entire schedule by δ to obtain a feasible schedule satisfying all original release dates.

5 Conclusion

In this paper, we have presented an approximation algorithm with running time $O(n \log n) + f(1/\epsilon)$ which computes a schedule for parallel tasks of length at most $(1.5+\epsilon)OPT(I)$. This narrows the gap between the lower bound of 1.5 and the previous upper bound of 2. Of course, faster approximation algorithms for the scheduling problem are interesting. An interesting open problem is the question whether there is an approximation algorithm for (non-malleable or malleable) parallel tasks with ratio exactly 1.5 (i.e. avoiding the additional $\epsilon > 0$) or not. We believe that the proposed techniques in this paper can also be used for other scheduling problems.

References

[1] A.K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis. Scheduling independent multiprocessor tasks. *Algorithmica*, 32(2):247–261, 2007.

[2] K.P. Belkhale and P. Banerjee. Approximate algorithms for the partitionable independent task scheduling problem. *Proc. of International Conference on Parallel Processing (ICPP)*, pages 72–75, 1990.

[3] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Handbook on Scheduling: From Theory to Applications*. Springer, 2007.

[4] T. Decker, T. Lücking, and B. Monien. A 5/4-approximation algorithm for scheduling identical malleable tasks. *Theoretical Computer Science*, 361(2):226–240, 2006.

[5] M. Drozdowski. *Scheduling for Parallel Processing.* Computer Communications and Networks, Springer, 2009.

[6] J. Du and J.Y.T. Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989.

[7] A. Feldmann, J. Sgall, and S.-H. Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science*, 130:49–72, 1994.

[8] A. Fishkin, K. Jansen, and M. Mastrolilli. Grouping techniques for scheduling problems: faster and simpler. *Algorithmica*, 51:183-199, 2008.

[9] M.R. Garey and R.L. Graham. Bounds for multiprocessor scheduling with resource constraints, *SIAM Journal on Computing*, 4:187-200, 1975.

[10] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[11] M.D. Grigoriadis, L.G. Khachiyan, L. Porkolab, and J. Villavicencio: Approximate max-min resource sharing for structured concave optimization, *SIAM Journal on Optimization*, 41: 1081–1091, 2001.

[12] L. Hall and D. Shmoys. Approximation schemes for constrained scheduling problems. *Proc. of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 134–140, 1989.

[13] K. Jansen and L. Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmica*, 32(3):507–520, 2002.

[14] K. Jansen. Approximation algorithms for min-max and max-min resource sharing problems, and applications. In *Efficient Approximation and Online Algorithms*, LNCS 3484, pages 156–202, 2006.

[15] K. Jansen and R. Solis-Oba. New approximability results for 2-dimensional packing problems. *Proc. of Mathematical Foundations of Computer Science (MFCS)*, pages 103–114, 2007.

[16] K. Jansen and R. Thöle. Approximation algorithms for scheduling parallel jobs. *SIAM Journal on Computing*, 39, 3571–3615, 2010.

[17] B. Johannes. Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, 9(5):433–452, 2006.

[18] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12:415–440, 1987.

[19] C. Kenyon and E. Rémila. A near optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.

[20] J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:269-271, 1990.

[21] J.Y.T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[22] W.T. Ludwig. Algorithms for scheduling malleable and non-malleable parallel tasks. PhD thesis, University if Wisconsin-Madison, 1995.

[23] W.T. Ludwig and P. Tiwari. Scheduling malleable and nonmalleable parallel tasks. *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 167–176, 1994.

[24] G. Mounie, C. Rapine, and D. Trystram. A $\frac{3}{2}$-approximation algorithm for scheduling independent monotonic malleable tasks. *SIAM Journal on Computing*, 37(2):401–412, 2007.

[25] E. Naroska and U. Schwiegelshohn. On an on-line scheduling problem for parallel jobs. *Information Processing Letters*, 81: 297–304, 2002.

[26] J. Turek, J.L. Wolf, and P.S. Yu. Approximate algorithms for scheduling parallelizable tasks. *Proc. of ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 323–332, 1992.

[27] P.M. Vaidya. An algorithm for linear programming which requires $O(((m+n)n^2 + (m+n)^{1.5}n)L)$ arithmetic operations. *Mathematical Programming*, 47: 175–201, 1990.