Approximation Algorithms for Scheduling Parallel Jobs: Breaking the Approximation Ratio of 2*

Klaus Jansen and Ralf Thöle

Institut für Informatik, Universität zu Kiel, Germany
{kj,rth}@informatik.uni-kiel.de

Abstract. In this paper we study variants of the non-preemptive parallel job scheduling problem where the number of machines is polynomially bounded in the number of jobs. For this problem we show that a schedule with length at most $(1+\varepsilon)\,\mathrm{OPT}$ can be calculated in polynomial time, which is the best possible result (in the sense of approximation ratio), since the problem is strongly NP-hard.
For the case when all jobs must be allotted to a subset of machines with consecutive indices a schedule with length at most $(1.5+\varepsilon)\,\mathrm{OPT}$ can be calculated in polynomial time. The previously best known results are algorithms with absolute approximation ratio 2.

1 Introduction

In classical scheduling theory, each job is executed by only one processor at a time. In the last years however, due to the rapid development of parallel computer systems, new theoretical approaches have emerged to model scheduling on parallel architectures (for an overview about scheduling of multiprocessor jobs see [2,5,13]).

We study variants of the non-preemptive parallel job scheduling problem. An instance of this problem is given by a list $L := \{J_1, \ldots, J_n\}$ of jobs and for each job J_j an execution time p_j and the number of required machines q_j is given. A schedule $S = ((s_1, r_1), \ldots, (s_n, r_n))$ is a sequence of starting times $s_j \geq 0$ together with the set of assigned machines $r_j \subseteq \{1, \ldots, m\}$ ($|r_j| = q_j$) for $j \in \{1, \ldots, n\}$. A schedule is feasible if each machine executes at most one job at the same time. The length of a schedule is defined as its latest job completion time $C_{\max} = \max\{s_j + p_j | j \in \{1, \ldots, n\}\}$. The objective is to find a feasible schedule of minimal length. This problem is denoted $P|\mathrm{size}_j|C_{\max}$ (for more information on this three field notation see [5]).

$P|\mathrm{size}_j|C_{\max}$ is strongly NP-hard, since already the problem $P5|\mathrm{size}_j|C_{\max}$, is strongly NP-hard [6], and, unless $P = NP$, there is no approximation algorithm

* Research supported by a PPP funding "Approximation algorithms for d-dimensional packing problems" 315/ab D/05/50457 granted by the DAAD, and by EU research project AEOLUS, Algorithmic Principles for Building Efficient Overlay Computers, EU contract number 015964.

with a performance ratio better than 1.5 for $P|\text{size}_j|C_{\max}$ [11]. Furthermore, Johannes [11] has shown that no list-scheduling algorithm can perform better than 2.

The best known algorithm with polynomial running time for this problem was implicitly given by Garey & Graham [7]. They proposed a list-based algorithm with approximation ratio 2 for a resource-constrained scheduling problem. In this scheduling problem one or more resources are given and for the duration of its execution time each job requires a certain amount of each resource. As pointed out by Ludwig & Tiwari [14] this resource-constrained scheduling problem can be used to model $P|\text{size}_j|C_{\max}$ by using the available processors as the single resource. The existence of a polynomial time approximation scheme (PTAS) for the case that the number of available processors is a constant, $Pm|\text{size}_j|C_{\max}$, was presented in [1,9].

A problem closely related to $P|\text{size}_j|C_{\max}$ is the *StripPacking*-problem (ie. packing of rectangles in a strip of width 1 while minimizing the packing heigth). The main difference is that machines assigned to a job need to be contiguous in a solution to the *StripPacking*-problem. We denote the corresponding *Scheduling*-problem by $P|line_j|C_{\max}$. Turek et al. [18] pointed out, that using contiguous machine assignments is desirable in some settings (for example to maintain a physical proximity of processors allotted to a job). One of the first results for the *StripPacking*-problem was given by Coffman et al. [3]. They proved that the level-based algorithm NFDH (Next Fit Decreasing Height) has approximation ratio 3. The currently best known algorithms with absolute approximation ratio were given independently by Schiermeyer [16] and Steinberg [17], who presented algorithms with ratio 2. Coffman et al. [3] also analyzed the asymptotic performance of NFDH, which is $2\,\text{OPT} + h_{\max}$, where $\text{OPT}, h_{\max}$ denote the height of an optimal solution and the height of the tallest rectangle, respectively. An AFPTAS for this problem was presented by Kenyon & Rémila [12]. Recently, Jansen & Solis-Oba [10] presented an asymptotic polynomial time approximation scheme (APTAS) with additive term 1 at the cost of a higher running time.

New results. In this paper we focus on the natural case when the number of machines is polynomial in the number of jobs (in most scenarios the number of machines will be even lower than the number of jobs). We will denote this problem by $P\text{poly}|\text{size}_j|C_{\max}$ or by $P\text{poly}|\text{line}_j|C_{\max}$ in the contiguous case. Using a reduction from 3-Partition [8], it is easy to see that these problems are strongly NP-hard.

For the case that all machines assigned to a job have contiguous addresses we show the existence of an algorithm with approximation ratio 1.5.

Theorem 1. *For every $\varepsilon > 0$ and every instance I of $P\text{poly}|line_j|C_{\max}$ there exists an algorithm A with $A(I) \leq (1.5 + \varepsilon)\,\text{OPT}(I)$ and running time polynomial in n, where $A(I)$ is the length of the schedule for instance I generated by algorithm A and $\text{OPT}(I)$ is the length of an optimal schedule for instance I.*

The previous best known result for this problem is the above mentioned 2-approximation algorithm for the resource-constrained problem by Ludwig & Tiwari [14].

In the general case (non-contiguous addresses) we show the existence of a polynomial time approximation scheme (PTAS).

Theorem 2. *For every $\varepsilon > 0$ and every instance I of $Ppoly|size_j|C_{\max}$ there exists an algorithm A with $A(I) \leq (1+\varepsilon)\,\mathrm{OPT}(I)$ and running time polynomial in n, where $A(I)$ is the length of the schedule for instance I generated by algorithm A and $\mathrm{OPT}(I)$ is the length of an optimal schedule for instance I.*

The previous best known result for this problem is a 2-approximation algorithm based on list scheduling by Garey & Graham [7]. This is the best possible result (in the sense of approximation ratio), since the problem is strongly NP-hard.

A similar problem is the scheduling of so called *malleable* jobs, where the number of required machines for each job is not known a priori; the execution time of each job depends on the number of allotted machines. That is, instead of p_j, q_j each job J_j has an associated function $p_j : \{1, \ldots, m\} \to \mathbb{Q}^+$ that gives the execution time $p_j(\ell)$ of J_j in terms of the number ℓ of processors that are assigned to J_j. For this *Scheduling*-problem Turek et al. [18] and Ludwig & Tiwari [14] presented algorithms with approximation ratio 2 for both cases (contiguous and non-contiguous) without additional properties. Mounié et al. [15] gave an algorithm for scheduling monotonic malleable tasks with approximation ratio $(1.5 + \varepsilon)$. For the special case of identical malleable tasks Decker et al. [4] presented an approximation algorithm with ratio 1.25. In the full version of this paper we will show how the algorithms for both non-malleable cases can be extended to solve the corresponding malleable versions with the same approximation ratios (without monotonic properties of the processing times).

Structure. In Sect. 2 we present the algorithm for scheduling jobs on machines with contiguous addresses. In Sect. 2.1 we show how an optimal solution can be transformed into a nearly optimal solution with simpler structure. After these preliminary steps we explain in Sect. 2.2 the pre-positioning of a subset of the rectangles and introduce a linear program (LP) formulation for the assignment of the remaining rectangles. In Sect. 2.3 we outline how to pack the rectangles according to the fractional solution of the LP. In Sect. 3 we present the scheduling algorithm for the case that the machines allotted to each job are not required to have contiguous addresses.

2 Contiguous Parallel Job Scheduling

Since each job is required to be executed on contiguous machines, an instance of the *Scheduling*-problem can be translated directly into a *StripPacking*-instance; create for each job J_i a rectangle $R_i = (w_i, h_i)$ with width $w_i = q_i/m$ and height $h_i = p_i$ (where q_i is the number of required machines and p_i is the execution time of J_i). The objective is then to find an orthogonal, axis parallel arrangement of all rectangles into a strip of width 1 and minimal height (without rotations). Thus, the StripPacking- and the Scheduling-notation can be used synonymously in the contiguous case. In this paper we use the StripPacking-notation since this notation is more descriptive.

2.1 Near-Optimal Packing with Simple Structure

The following construction of a nearly optimal solution with simple structure based on a given optimal solution is similar to the solution constructed in [10].

Let $0 < \varepsilon' \leq 1$ and let $L = \{R_1, \ldots, R_n\}$ be a *Scheduling*-instance and for each rectangle (job) R_i let w_i be its width and h_i be its height. A packing (schedule) P for instance L is given as a set of pairs $P = \{(x_1, y_1), \ldots, (x_n, y_m)\}$, where each pair $(x_i, y_i) \in \mathbb{R}^2_{\geq 0}$ denotes the position of the lower left corner of rectangle R_i in the strip. Note that in this case the representation of the schedule by a packing is sufficient since the subset of assigned processors is well-defined by the first assigned processor. We assume that the lower left corner of the strip coincides with the origin of a Cartesian system of coordinates. A packing P is valid if the rectangles do not overlap and $x_i + w_i \leq 1$ for all $i \in \{1, \ldots, n\}$. The height of packing P is given by $\mathrm{h}(P) := \max_{i \in \{1,\ldots,n\}}(y_i + h_i)$.

Bounded height. Since we want to divide the solution into a constant number of *slots*, we need to know the height of an optimal solution, at least up to the required accuracy ε'. By using the *StripPacking*-algorithm of Steinberg [17], we can find a solution for the *StripPacking*-instance of height $v \leq 2\,\mathrm{OPT}$, where OPT is the height of an optimal solution. Let $C := \{(1 + 0\varepsilon')v/2, (1 + 1\varepsilon')v/2, \ldots, (1 + \lceil 1/\varepsilon' \rceil \varepsilon')v/2\}$, then there exists a value $v^* \in C$ such that $\mathrm{OPT} \leq v^* \leq (1 + \varepsilon')\,\mathrm{OPT}$. Obviously we only have to consider $\lceil 1/\varepsilon' \rceil + 1$ different candidates to find the right one. For simplicity we divide the height of each rectangle by v^*, such that the height of an optimal solution for the scaled instance is bounded by 1. In the following we show the existence of an algorithm that packs all rectangles of a scaled instance into a strip of height at most $(1.5 + \varepsilon')$. This height bound is sufficient to prove Theorem 1, since rescaling yields $v^*\,(1.5 + \varepsilon') \leq (1.5 + 4\varepsilon')\,\mathrm{OPT}$.

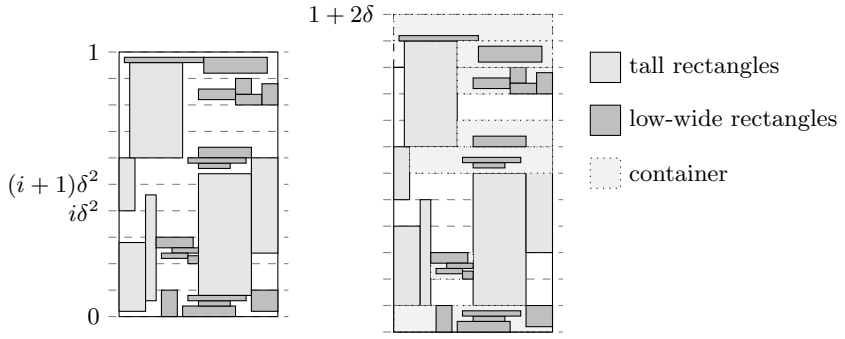
Partitioning the set of rectangles. In the following we assume that the instance is already scaled such that an optimal packing P^* has height $\mathrm{h}(P^*)$, where $(1-\varepsilon') < \mathrm{h}(P^*) \leq 1$. Let ε be the largest value of the form $\varepsilon = 1/(2a)$ for an integer a such that $\varepsilon \leq \varepsilon'/13$. First we create a gap in size between *tall* and *low* rectangles and between *wide* and *narrow* restangles by removing *middle-sized* rectangles with small total area. Let $\sigma_0 := 1, \sigma_1 := \varepsilon$, and $\sigma_k := (\sigma_{k-1})^{8/\sigma_{k-1}^3}$ for all $k \geq 2$. Define $L^{>1/2} := \{R_i \in L | h_i > (1+\varepsilon)/2\}$, and $L_k := \{R_i \in L \setminus L^{>1/2} | w_i \in (\sigma_k, \sigma_{k-1}]$ or $h_i \in (\sigma_k, \sigma_{k-1}]\}$. Define for each subset $L' \subseteq L$ the total area of L' by $A(L') = \sum_{R_i \in L'}(w_i h_i)$.

Lemma 1. *There exists $k \in \{2, \ldots, 4/\varepsilon + 1\}$ such that $A(L_k) \leq \varepsilon/2 A(L)$.*

Choose the smallest value k satisfying the conditions of Lemma 1 and define $\delta := \sigma_{k-1}$ and $s := 8/\delta^3$. For simplicity we define the following sets and call rectangles belonging to each set accordingly: $L_{\mathrm{ta}} := \{R_i \in L | h_i > \delta\}$ *tall* rectangles, $L_{\mathrm{lo}} := \{R_i \in L | h_i \leq \delta^s\}$ *low* rectangles, $L_{\mathrm{wi}} := \{R_i \in L | w_i > \delta\}$ *wide* rectangles, and $L_{\mathrm{na}} := \{R_i \in L | w_i \leq \delta^s\}$ *narrow* rectangles. We will denote the subset of low-wide rectangles in the following with $L_{\text{lo-wi}} := L_{\mathrm{lo}} \cap L_{\mathrm{wi}}$.

Rounding and shifting tall rectangles. A crucial part of the simple structure are the positions and heights of the tall rectangles. Let P be an optimal

packing for all rectangles. First we increase the height of each tall rectangle $R_i \in L_{\text{ta}}$ to the nearest multiple of δ^2. Then, we shift the rectangles up until all rectangles $R_i \in L_{\text{ta}}$ have their corners placed at points (x'_i, y'_i), such that there exists some integer k_i with $x'_i = x_i$ and $y'_i = k_i\delta^2$. Since tall rectangles have height at least δ, these transformations increase the total height of packing P by at most $1/\delta(2\delta^2) = 2\delta$ (see Fig. 1).



(a) Original packing (b) Packing with simpler structure

Fig. 1: Rounding and shifting rectangles

Containers for low rectangles. Since we want to increase only the height but not the width of the packing, we cannot round up the width of the wide rectangles. Instead we introduce *containers*, into which all low-wide ($L_{\text{lo-wi}}$) and a subset of the low-narrow rectangles will be packed. Consider a scaled and shifted packing P. Draw horizontal lines spaced by a distance δ^2 across the strip (due to the rounding and shifting, the lower and upper sides of the tall rectangles lie along two of these lines). These lines split the strip into at most $(1+2\delta)/\delta^2$ horizontal rectangular regions that we call *slots* (see Fig. 1). A container is a rectangular region inside a slot whose left boundary is either the right side of a tall rectangle or the left side of the strip, and whose right boundary is either the left side of a tall rectangle or the right side of the strip (see Fig. 1). In the following we consider only containers that contain at least one low-wide rectangle. As in [10], we can show, that a packing can be transformed such that all non-empty containers have width $w_{\max}(C) + i\delta^s$ for $i \in \mathbb{N}$, where $w_{\max}(C)$ is the sum of the width of at most $1/\delta$ low-wide rectangles. This limits the number of possible widths for the containers; however, some of the rectangles previously packed in C might not fit anymore, but the total area of all non-fitting rectangles is bounded by δ.

2.2 Pre-positioning

The next step is to determine the positions of the containers and of a subset of the tall rectangles (*critical* rectangles). On the one hand we have to make sure that all discarded rectangles have height bounded by $1/2$. On the other hand we have to make sure that the pre-positioning has a polynomial running time. In particular, we can only enumerate the positions of a constant number of tall rectangles and containers. From here on let $\mathcal{C}$ be the set of containers and let $L'_{\text{ta}} \subseteq L_{\text{ta}} \setminus L^{>1/2}$ be the subset of K tall rectangles with largest area for some constant K (see full version), and let $L' = \mathcal{C} \cup L'_{\text{ta}}$. Note that $|\mathcal{C}| \leq {}^{(1+2\delta)}/_{\delta^3} \leq 2\delta^{-3}$ and thus $|L'| \leq K+2\delta^{-3}$. In order to determine the positions of the *critical* rectangles, first we enumerate assignments of the K tall rectangles L'_{ta} and of the containers $\mathcal{C}$ to *slots* and *snapshots*. Then we describe a dynamic program that assigns the tall rectangles from $L^{>1/2}$ to snapshots without enumerating all possibilities. Using these assignments we set up a linear program (LP). If this LP has a solution, we have found a fractional solution for the packing problem. Furthermore, if almost all low-wide rectangles fit into the containers, we show that the fractional solution can be transformed into a feasible integral solution by discarding some rectangles with small total area.

Slot assignment. We split again the strip into horizontal slots of height δ^2. A slot assignment for L' is a mapping $f : L' \to M$ where $M = \{1, \ldots, {}^{(1+2\delta)}/_{\delta^2}\}$ corresponds to the set of slots. For a given slot assignment f the set of slots that will be used for packing a rectangle $R_j \in L'$ is given by $\{f(R_j), \ldots, f(R_j) + \gamma_j - 1\}$, where $\gamma_j \delta^2 = h_j$ is the height of rectangle R_j (in particular $\gamma_j = 1$ if R_j is a container). Since the number of different mappings f is bounded by $O(n^{(1+2\delta)/\delta^2})$, we can consider all mappings f in polynomial time and for each mapping try to find a packing for L that is consistent with f.

Snapshots. In order to handle the x position of the rectangles we introduce snapshots. We use the snapshots to model the relative horizontal positions of all rectangles in L'. Consider a packing for L'. Trace vertical lines extending the sides of the rectangles in L' (see Fig. 2). The region between two of these adjacent lines is called a snapshot. If we index all snapshots from left to right, every rectangle $R_j \in L'$ appears in a sequence of consecutive snapshots $\mathrm{S}_{\alpha_j}, \ldots, \mathrm{S}_{\beta_j}$, where α_j denotes the index of the first snapshot in which rectangles R_j occurs and β_j denotes the index of the last snapshot. In Fig. 2 rectangle R_1 is contained in snapshot S_1, while R_2 is contained in snapshots $\mathrm{S}_2, \mathrm{S}_3, \mathrm{S}_4$, thus $\alpha_1 = 1, \beta_1 = 1, \alpha_2 = 2$ and $\beta_2 = 4$. More formally, an assignment of rectangles in L' to snapshots is given by two functions $\alpha, \beta : L' \to \{1, \ldots, g\}$, where g denotes the number of snapshots. Since $|L'| \leq K + 2\delta^{-3}$ the maximum number of snapshots g in any packing for L' is at most $g \leq 2(K + 2\delta^{-3})$, and thus the number of different assignments of L' to snapshots is polynomial, $O(g^{2|L'|})$.

Dynamic program for $L^{>1/2}$-rectangles. In general we cannot consider all assignments of rectangles in $L^{>1/2}$ to snapshots, because there might be up to n rectangles in $L^{>1/2}$. In the following we introduce an algorithm that allows us to enumerate a subset of all snapshot assignments for $L^{>1/2}$ such that the size of the subset is polynomially bounded in n and there exists one snapshot

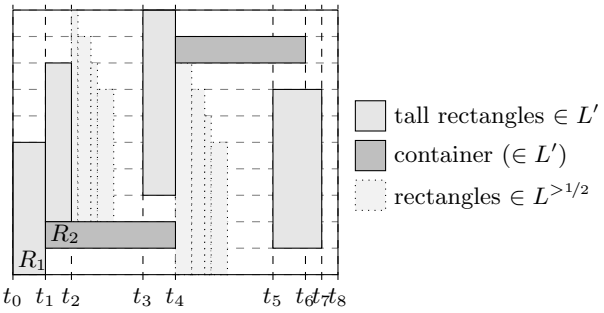


Fig. 2: Packing of rectangles and containers and induced snapshots

assignment in this subset that is *equivalent* (up to renaming) to a snapshot assignment induced by an optimal packing. Rectangles in $L^{>1/2}$ that intersect more than one snapshot are handled separately, since our packing algorithm can only be used for tall rectangles that do not intersect more than one snapshot. We guess the set of tall rectangles intersecting snapshot boundaries, which can be done in polynomial time since there are at most g of these rectangles. In order to pack these rectangles we simply add them to L' (the set of K tall rectangles and containers). This modification increases the size of L' such that $|L'| \leq 3(K + 2\delta^{-3})$, but this does not increase the vector sizes, that we define in the following, since the snapshots introduced by these added rectangles obviously do not allow further $L^{>1/2}$ rectangles to be packed in them (height $> 1/2$). Note that, due to the scaling of all tall rectangles and due to the height bound of 1 for all rectangles, $L^{>1/2}$ can be partitioned into $(1-\varepsilon)/(2\delta^2) < 1/(2\delta^2)$ sets $L_i^{>1/2}$ ($i \in I := \{(1+\varepsilon)/(2\delta^2) + 1, \ldots, 1/\delta^2\}$), where each set $L_i^{>1/2}$ contains all rectangles of height $s_i := i\delta^2$, that is $L_i^{>1/2} := \{R_j \in L^{>1/2} | h_j = s_i\}$. Consider a packing of all rectangles and snapshots as defined above. Then we can define a vector $v^i = (v_1^i, \ldots, v_g^i)$ for each height $s_i, i \in I$, where $v_j^i \in \{0, \ldots, m\}$ is chosen such that $v_j^i \cdot 1/m$ is the sum of widths of all rectangles of height s_i contained in snapshot S_j and height s_i. We assume that a rectangle is contained in a snapshot, if its left side is contained in it. Since the width of the strip is bounded by 1, the value of every component is bounded by m. Thus, a rough upper bound for the number of feasible vectors for each height s_i is given by $(m+1)^g$. The algorithm to calculate all feasible vectors for a given height s_i works as follows.

Assume that $L_i^{>1/2} = \{R_1, \ldots, R_{k_i}\}$. Starting with the set $V := \{(0, \ldots, 0)\}$ we replace in step $l \in \{1, \ldots, k_i\}$ each vector $v \in V$ with all vectors that can be generated by adding $\gamma_l := w_l \cdot m$ to one of its components. After each step remove all duplicate vectors. Removing all duplicates ensures that after each step the number of vectors is bounded by $(m+1)^g$. Using this bound it is easy to see that the overall running time for height s_i is bounded by $O(g^2(m+1)^{g+2})$, since $k_i = |L_i^{>1/2}| \leq |L^{>1/2}| \leq m$, and for each step l at most $g \cdot (m+1)^g$ vectors are generated; furthermore, at most $g^2(m+1)^g \log(m+1)$ comparisons are needed

for the removal of duplicate vectors. Obviously the vector v^i induced by the given packing can be found among the generated vectors. Let V^i denote the set of vectors generated for this height class $L_i^{>1/2}$.

Repeating this computation for every height class $L_i^{>1/2}$ $(i \in I)$ leads to $|I| \leq 1/(2\delta^2)$ sets of at most $(m+1)^g$ vectors. Since each set contains at least the null vector and thus each set is not empty, we can build the direct product $V := \prod_{i \in I} V^i$ of these sets. V contains at most $((m+1)^g)^{1/(2\delta^2)}$ elements and each of these elements consists of one vector for each height class. One element $v = (v^1, \ldots, v^{|I|}) \in V$ corresponds to the vectors induced by the given packing.

In our packing algorithm we try each vector $v \in V$ consisting of components $v^i, i \in I$ and use these vectors v^i to pack the tall rectangles into the snapshots. Using the described version of the dynamic program results only in (many) vectors, but for our packing algorithm we also need to know what combination of rectangles leads to the given width per snapshot. This can be achieved by extending the dynamic program such that for each vector a component consists not only of the current width, but also of a set of rectangles. During the vector generation step, a rectangle is added to this set, if its width is added to the corresponding width component. Due to this modification the space needed to store the vectors increases but is still polynomial in n; the runtime of the dynamic program is not affected significantly.

Linear Program. In this subsection we present a linear program (LP), which allows us to calculate the width of all snapshots, and thus determine the positions of all rectangles in L'. We now assume that we have chosen a slot assignment f, functions α, β, and $v \in V$ consisting of vectors v^i of widths for each height class. Since all low-wide rectangles and a subset of the low-narrow rectangles get packed into the containers, we do not need to consider them in the LP. For convenience we call the subset of the low-narrow rectangles that are packed into the containers $L^C_{\text{lo-na}}$, and the remaining low-narrow rectangles $L_{\text{lo-na}}$. We construct $L^C_{\text{lo-na}}$ by greedily taking low-narrow rectangles as long as their total area plus the total area of all low-wide rectangles does not exceed the total area of the containers. We discard the first low-narrow rectangle that exceeds the total area in order to assure that enough space can be reserved for the remaining rectangles in the following LP. This discarded rectangle has area at most δ^{2s}, in fact we will show, that this discarded rectangle can be packed along with the rectangles from $L_{\text{lo-na}}$ (see Sect. 2.3). We will denote the total area of the remaining rectangles from $L_{\text{lo-na}}$ as $\mathrm{A}(L_{\text{lo-na}})$. Since f, α, β are fixed, we can calculate the set of *free* slots (i.e. the slots not occupied by L' rectangles) for each snapshot. These free slots will be used for packing the remaining tall rectangles and for the small rectangles from $L_{\text{lo-na}}$. In order to formulate constraints to ensure that enough space is reserved for these rectangles we introduce configurations. We define a configuration as a pair $(\mathrm{SN}, \mathit{\Pi})$, where SN is a subset of the free slots reserved for rectangles from $L_{\text{lo-na}}$ and $\mathit{\Pi}$ is a partition of the remaining free slots into sets of consecutive slots reserved for rectangles from $L_{\text{ta}} \setminus L'$. Every subset $F \in \mathit{\Pi}$ of cardinality $l = |F|$ is reserved to pack rectangles from L_{ta} of height $l\delta^2$. Let n_j denote the number of different

configurations for each snapshot S_j and let $c_i^j := (SN_i^j, \Pi_i^j)$ denote the different configurations for snapshot S_j, $i \in \{1, \ldots, n_j\}$. Let $n_i^j(l) := |\{F \in \Pi_i^j : |F| = l\}|$ denote the number of sets of cardinality l in Π_i^j. The total width of all rectangles in $L_{\text{ta}} \setminus L'$ of height l is denoted as W_l. The variables $x_i^j, j \in \{1, \ldots, g\}, i \in \{1, \ldots, n_j\}$ are used to determine the width of each configuration c_i^j. Additional variables $t_j, j \in \{1, \ldots, g\}$, are used to determine the width of each snapshot S_j.

$$
\begin{aligned}
\text{LP}(f, \alpha, \beta, v): \quad & t_0 = 0, t_g \leq 1 & \\
& t_j \geq t_{j-1} & \forall j \in \{1, \ldots, g\} \\
& t_{\beta_j} - t_{\alpha_j} = w_j & \forall R_j \in L' \\
& \sum_{i=1}^{n_j} n_i^j(l) x_i^j \geq \frac{1}{m} v_j^l & \forall j \in \{1, \ldots, g\}, l \in I \\
& \sum_{j=1}^{g} \sum_{i=1}^{n_j} n_i^j(l) x_i^j \geq W_l & \forall l \in M \\
& \sum_{j=1}^{g} \sum_{i=1}^{n_j} x_i^j |SN_i^j| \delta^2 \geq A(L_{\text{lo-na}}) & \\
& \sum_{i=1}^{n_j} x_i^j \leq t_j - t_{j-1} & \forall j \in \{1, \ldots, g\} \\
& x_1^j, \ldots, x_{n_j}^j \geq 0 & \forall j \in \{1, \ldots, g\}
\end{aligned}
$$

Since $g, n_j, |L'|, |M|, |I|$ are independent of n, this linear program can be solved in polynomial time. If $\text{LP}(f, \alpha, \beta, v)$ has no feasible solution we construct a new LP with a new combination of f, α, β, v.

2.3 Packing the Rectangles

Due to the page limit we present here only a brief outline of the actual packing of the rectangles. A complete description is given in the full version of this paper. A crucial step is to sort and adopt the configurations. On the one hand, this allows us to pack the tall rectangles $L^{>1/2}$ and on the other hand, this assures that the fragmentation of the space reserved for low-narrow rectangles is *limited*. The pre-positioned rectangles are packed according to the LP-solution. The remaining tall rectangles get fractionally packed in a greedy manner. We pack the containers using a modified version of the algorithm by Kenyon & Rémila [12]. The remaining low-narrow rectangles are packed using a NFDH approach. Overall we can show that almost all rectangles can be packed. In particular, the remaining rectangles have a small total area and maximum height bounded by $(1+\varepsilon)/2$ and thus can be packed in an additional strip with height bounded by $\varepsilon + 9\delta + (1+\varepsilon)/2$. Stacking these strips on top of each other leads to a strip with total height bounded by $1.5 + \varepsilon'$.

3 Non-contiguous Parallel Job Scheduling

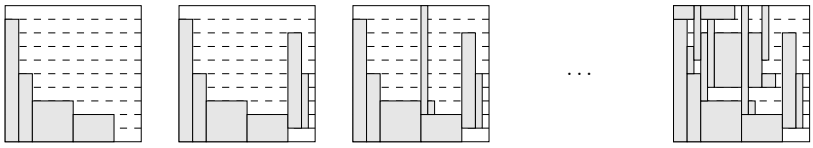
In the following we construct a polynomial time approximation scheme (PTAS) for the non-contiguous machine indices case. The first steps for this algorithm are

basically the same as before. We transform the problem into a packing problem, guess the height of an optimal schedule in order to scale the instance, such that the height of an optimal solution for the scaled instance is bounded by 1. Instead of the 2-approximation algorithm for the *StripPacking*-problem, we use a 2-approximation algorithm for this *Scheduling*-problem by Garey & Graham [7]. We again partition the rectangles, but for this setting it is sufficient to partition the set into tall, low-narrow, and low-wide rectangles, and we reduce the search space by scaling and shifting the tall rectangles in the same manner as before (see Sect. 2.1). After this step the algorithms differ significantly. We use a dynamic program to find a distribution of the tall rectangles among the slots. Then we pack the tall rectangles according to the distribution in a canonical way. The remaining space is merged into one bin per slot. We pack the low rectangles into these bins using again the modified Kenyon & Rémila algorithm. Then, creating a feasible packing can be done by a simple greedy algorithm.

Partitioning the set of rectangles. Let ε' denote the requested accuracy and let $\varepsilon \leq \varepsilon'/9$ be the largest value of the form $\varepsilon = 1/a$ for some integer value a. Let $\sigma_0 := 1, \sigma_1 := \varepsilon$, and $\sigma_k := \sigma_{k-1}^3/2 \cdot 7^2$ for all $k \geq 2$. Define $L_k := \{R_i \in L | h_i \in (\sigma_k, \sigma_{k-1}] \text{ or } w_i \in (\sigma_k, \sigma_{k-1}]\}$. It is easy to see, that there exists $k \in \{2, \ldots, 2/\varepsilon + 1\}$ such that $A(L_k) \leq \varepsilon A(L)$ (see Lemma 1). Choose the smallest value $k \in \{2, \ldots, 2/\varepsilon + 1\}$ such that $A(L_k) \leq \varepsilon A(L)$, and let $\delta := \sigma_{k-1}$, and $\gamma := \sigma_k$. Define $L_{\text{ta}} := \{R_i \in L | h_i > \delta\}$ *tall* rectangles, $L_{\text{lo-wi}} := \{R_i \in L | h_i \leq \gamma, w_i > \delta\}$ *low-wide* rectangles, and $L_{\text{lo-na}} := \{R_i \in L | h_i \leq \gamma, w_i \leq \gamma\}$ *low-narrow* rectangles.

Shifting and Rounding. Shifting and rounding is done the same way as before. Again this leads to the existence of a packing with height bounded by $1 + 2\delta$ (see Sect. 2.1).

Dynamic program for tall rectangles. Draw horizontal lines spaced by a distance δ^2 across the strip starting with the x-axis as first such line. Note that, due to the rounding and shifting the lower side of each tall rectangle corresponds to one of these horizontal lines. We say that rectangle R_i is in slot j if its lower bound corresponds to the jth horizontal line. Since we cannot enumerate all possible assignments of tall rectangles to slots in polynomial time, we use a dynamic programming approach. Due to the height bound of 1 and the rounding, we can partition the set of tall rectangles L_{ta} into height classes $L_{\text{ta}}^i := \{R_j \in L_{\text{ta}} | h_j = i\delta^2\}, i \in I := \{1, \ldots, 1/\delta^2\}$. Note that we partition all tall rectangles in this case, not only $L^{>1/2}$. For each height class $i \in I$ we define a vector $v^i = (v_1^i, \ldots, v_{(1+2\delta)/\delta^2}^i)$, such that each entry v_j^i denotes the total width of all tall rectangles with height $i\delta^2$ in slot $j \in J := \{1, \ldots, (1+2\delta)/\delta^2\}$. The algorithm to calculate all feasible vectors for a given height class i works as follows. Assume that $L_{\text{ta}}^i = \{R_1, \ldots, R_{k_i}\}$. Starting with a set $V^i := \{(0, \ldots, 0)\}$ we replace in step $l \in \{1, \ldots, k_i\}$ each vector $v \in V^i$ with all vectors that can be generated by adding w_l to one of its components. After each step remove all duplicate vectors. Similar arguments as in Sect. 2.2 show that the number of vectors generated and the running time of the algorithm is polynomially bounded in m. After repeating this computation for each height class L_{ta}^i, we can again build the direct product



(a) First slot (b) Second slot (c) Third slot (d) (e) All slots

Fig. 3: Canonical packing for tall rectangles

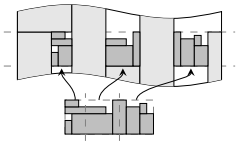


Fig. 4: Split bins to pack slots

of all sets of vectors $V := \prod_{i \in I} V^i$ and again there is an element $v \in V$ that corresponds to the vectors of widths induced by an optimal solution. Analogous to the first case, we extend our dynamic program such that for each component of each vector a set of associated rectangles is stored. Let $L(v_j^i)$ be this set of rectangles associated with v_j^i. We call an element $v \in V$ feasible, if for each slot the total width of all rectangles intersecting this slot and of all rectangles in this slot is not greater than m. In contrast to the previous case not all elements $v \in V$ are feasible.

Canonical packing for the tall rectangles. Given a feasible vector $v \in V$, we can pack all tall rectangles using the following simple algorithm (see Fig. 3). The algorithm starts with the first slot and packs left aligned all tall rectangles $L(v_1^i), i \in I$ into it. This is obviously possible, since the vector is feasible. Now assume that we have packed all slots prior to slot j. The free space in slot j is sufficient to pack all rectangles assigned by $v_j^i, i \in I$, since v is feasible, and furthermore, the free space in this slot is also free in all following slots. This allows us to pack all rectangles given by v_j^i left aligned into slot j.

Bins for the low rectangles. Given a feasible combination of vectors $v \in V$, the total width for each slot that is not occupied by tall rectangles is fixed. Let w_j^f denote the total width of the free space for slot j and define for each slot j a bin C_j of width w_j^f and height δ^2. If we find a packing of all low rectangles into these bins, and know the positions of the tall rectangles, we can simply split the bins into slices of width $1/m$ and add them successively left aligned to the free space of each corresponding slot (see Fig. 4). Since there exists a packing for all rectangles into the strip, the total area of the bins is not smaller than the total area of the remaining low-wide rectangles together with the low-narrow rectangles. We can show, that almost all rectangles are packable using a modified version of the algorithm by Kenyon & Rémila [12].

Analysis. Overall we discarded rectangles with total area bounded by $\varepsilon+2\delta$. Packing these rectangles using NFDH and due to the fact the all these rectangles have height bounded by δ yields an additional height of $2(\varepsilon+2\delta)+\delta$. Thus, the height of the resulting strip is bounded by $(1+2\delta)+(2\varepsilon+5\delta) \leq 1+9\varepsilon \leq 1+\varepsilon'$, if we stack these to strips on top of each other. Thus, we can prove Theorem 2, since rescaling yields $v^*(1+\varepsilon') \leq (1+3\varepsilon')\,\mathrm{OPT}$.

References

1. A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis. Scheduling independent multiprocessor tasks. *Algorithmica*, 32(2):247–261, 2007.
2. J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Handbook on Scheduling: From Theory to Applications*. Springer, 2007.
3. E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.
4. T. Decker, T. Lücking, and B. Monien. A 5/4-approximation algorithm for scheduling identical malleable tasks. *Theor. Comput. Sci.*, 361(2):226–240, 2006.
5. M. Drozdowski. Scheduling multiprocessor tasks – an overview. *European Journal of Operational Research*, 94(2):215–230, 1996.
6. J. Du and J. Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM J. Disc. Math.*, 2(4):473–487, 1989.
7. M. R. Garey and R. L. Graham. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
9. K. Jansen and L. Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmica*, 32(3):507–520, 2002.
10. K. Jansen and R. Solis-Oba. New approximability results for 2-dimensional packing problems. In *Symp. on Math. Foundations of Computer Science (MFCS)*, pages 103–114, 2007.
11. B. Johannes. Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, 9(5):433–452, 2006.
12. C. Kenyon and E. Rémila. A near optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.
13. J. Y-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.
14. W. Ludwig and P. Tiwari. Scheduling malleable and nonmalleable parallel tasks. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 167–176, 1994.
15. 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.
16. I. Schiermeyer. Reverse-fit: A 2-optimal algorithm for packing rectangles. In *Proc. 2nd European Symposium on Algorithms (ESA)*, pages 290–299, 1994.
17. A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal of Computing*, 26(2):401–409, 1997.
18. J. Turek, J. L. Wolf, and P. S. Yu. Approximate algorithms for scheduling parallelizable tasks. In *Proc. 4th ACM Symp. on Parallel Alg. and Architectures (SPAA)*, pages 323–332, 1992.