Tight approximation for scheduling parallel jobs on identical clusters

Marin Bougeret, Pierre-Francois Dutot, Klaus Jansen, Christina Robenek, Denis Trystram

Tight approximation for scheduling parallel jobs on identical clusters

Marin Bougeret[*], Pierre-François Dutot[†], Klaus Jansen[‡],
Christina Robenek,[‡] Denis Trystram[†¶]

Abstract

We consider the Multiple Cluster Scheduling Problem ($MCSP$), where the objective is to schedule n parallel rigid jobs on N identical clusters, minimizing the maximum completion time (makespan). $MCSP$ is 2-inapproximable (unless $\mathcal{P} = \mathcal{NP}$), and several approximation algorithms have already been proposed. However, ratio 2 has only been reached by algorithms that use extremely costly and complex subroutines as "black boxes" which are polynomial and yet impractical due to prohibitive constants.

Our objective within this work is to determine a reasonable restriction of $MCSP$ where the inapproximability lower bound could be tightened in almost linear time. Thus, we consider a restriction of $MCSP$ where jobs do not require strictly more than half of the processors of a cluster, and we provide a 2-approximation running in $O(\log(nh_{max})n(N+\log(n)))$, where h_{max} is the processing time of the longest job. This approximation is the best possible, as this restriction (and even simpler ones) remains 2-inapproximable.

1 Introduction

1.1 Problem statement

In new parallel computing platforms, several clusters share their computing resources in order to distribute the workload. Each cluster is composed of a set of identical processors connected by a local interconnection network. Jobs are submitted in successive packets called batches. The objective is to minimize the time when all the jobs of a batch are completed, to start the following batch as soon as possible. Many such computational systems are available all over the world, and the efficient management of the resources is a crucial problem. Let us start by defining formally the Multiple Cluster Scheduling Problem ($MCSP$).

[*]LIRMM, Montpellier
[‡]Department of Computer Science, University Kiel, Germany
[†]Grenoble University
[¶]Institut Universitaire de France

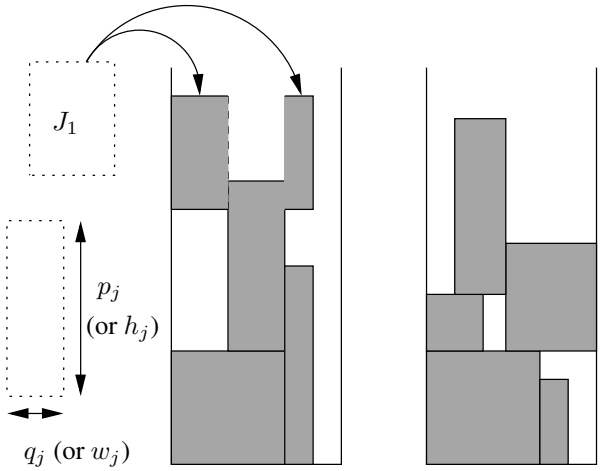

Figure 1. Example for 9 jobs and 2 clusters of a feasible solution for $MCSP$ and not feasible for $MSPP$.

Definition 1 ($MCSP$) *We are given n parallel rigid jobs J_j, $1 \leq j \leq n$, and N clusters. A job J_j requires q_j processors during p_j units of time, and each cluster owns m identical processors. The objective is to schedule all the jobs in the clusters, minimizing the maximum completion time (makespan) under the following constraints:*

1. *the q_j processors allocated to job J_j must belong to the same cluster (as proposed in [5])*
2. *at any time, the total number of processors used in any cluster must be lower or equal to m*

$MCSP$ is closely related to the Multiple Strip Packing Problem ($MSPP$) where the objective is to pack n rectangles in N strips, minimizing the maximum reached height. The only difference between these two problems is the "contiguous" constraint. Indeed, in $MSPP$ rectangles must be allocated contiguously, which in terms of job scheduling amounts to force jobs to use consecutive indexes of processors (see Figure 1).

Of course, the results for $MCSP$ generally do not apply to $MSPP$, as algorithms may schedule jobs in a non con-

tiguous way. The converse is also not clear, as ratio of approximation algorithms for $MSPP$ may not be preserved when considering $MCSP$ (optimal value of an $MCSP$ instance may be strictly better than the corresponding one for $MSPP$). However, as we can notice in Figure 2, many results for $MSPP$ directly apply to $MCSP$, as the proposed algorithms build contiguous schedules that are compared to non-contiguous optimal solutions.

We refer to the packing context in this paper, so that the solutions can be described using the classical vocabulary of packing problems. Thus, the problem treated in this paper ($MCSP$) consists in packing n rectangles in N strips of width 1, minimizing the maximum reached height, and without contiguous constraints[1].

1.2 Related Work

As shown in [11] using a gap reduction from 2 partition, $MCSP$ (and $MSPP$) are 2-inapproximable in polynomial time unless $\mathcal{P} = \mathcal{NP}$, even for $N = 2$. The main positive results for $MCSP$ are summarized below, in Figure 2.

Problem	Ratio	Remarks	Source
$MCSP$, $MSPP$	2ρ	Need solving $P\|\|C_{max}$ with a ratio ρ	[10]
$MCSP$	$5/2$	Fast algorithm	[3]
$MCSP$, $MSPP$	2	Need applying the PTAS of [1] for a constant ($\approx 10^4$) number of clusters	[2]
$MCSP$, $MSPP$	AFPTAS	Additive constant in $\mathcal{O}(\frac{1}{\epsilon^2})$, and in $\mathcal{O}(1)$ for large values of N	[2]
$MCSP$	3	Fast (and decentralized) algorithm handling clusters of different size	[9]
$MCSP$	2	Requires $\max_j w_j \leq \frac{1}{2}$ Fast algorithm	this paper

Figure 2. Main results

We distinguish the 3-approximation [9] and the $\frac{5}{2}$-approximation [3] that have a low computational complexity (these algorithms are usable on real size instances) from the 2-approximation [2] and the $2+\epsilon$-approximation [10].

At a first sight, the best results seem to be the 2-approximation in [2] and the $2+\epsilon$-approximation in [10]. However, the 2-approximation requires using a high running time algorithm when the number of clusters is lower than a huge constant N_0. Thus, any exponential dependency in N_0 is hidden, and the value of this constant ($N_0 \approx 10^4$) makes this algorithm practically impossible to use. Moreover, the $2+\epsilon$-approximation requires solving the famous $P||C_{max}$ problem (which is makespan minimization when scheduling sequential jobs on identical machines) with a ratio $1+\frac{\epsilon}{2}$. Thus, to give a rough idea, applying this technique with $\epsilon = \frac{1}{3}$ would lead to a $\Omega(n^{36})$ algorithm, using the PTAS of [6]. [2]

1.3 Motivations and contributions

As explained above, the two previous results of [2, 10] require both extremely high running time, and do not provide any insight on $MCSP$ because of these black boxes subroutines. Thus, we followed in [3] another approach by looking for fast and direct algorithms for $MCSP$. Our previous $\frac{5}{2}$-approximation in [3] is based on the discarding technique presented in Section 2.2. What we call discarding technique is a classical framework in scheduling problems. The idea is to define properly a set of *negligeable* items (items are rectangles here), and to prove that it is possible to add these items only at the end of the algorithm without degrading the approximation ratio. Thus, the effort can be focused on the set I' of remaining *large* items, that are generally more structured.

The $\frac{5}{2}$-approximation was obtained through a basic application of this technique (*i.e.* with a set I' containing only really huge rectangles, in this case rectangles whose width is larger than $\frac{1}{2}$). As we believe that the discarding technique of Section 2.2 is well suited for $MCSP$, we apply it again using a more challenging set I'. A natural direction would be to improve the $\frac{5}{2}$ ratio for $MCSP$ by targeting a *fixed* ratio $\rho < \frac{5}{2}$. Typically, one could target $\rho = \frac{7}{3}$ by defining the small jobs as those whose length is lower than $\frac{1}{3}$ (instead of $\frac{1}{2}$). However, as the relative performance improvement is getting smaller, and the difficulty of these "ratio tailored" proofs is likely to increase rapidly, we consider here a different approach.

Our objective is to find a reasonable restriction of $MCSP$ where the inapproximability lower bound could be tightened in almost linear time. In this spirit, we study a restriction of $MCSP$ where all rectangles have a width lower than $\frac{1}{2}$, meaning that jobs submitted to the clusters do not require strictly more than half of the processors. We provide for this problem a very fast 2-approximation running in $O(\log(nh_{max})n(N+\log(n)))$, where h_{max} is the maximum height of any rectangle. It turns out that this result is the best possible approximation, as this restriction of $MCSP$ (and even simpler ones, where the width of rectangles is lower than $\frac{1}{c}$, $c \in \mathbb{N}, c \geq 2$) remains 2-inapproximable unless $\mathcal{P} = \mathcal{NP}$.

[1] Therefore, when trying to pack a rectangle of width w_j at level l of a strip, we only need to check if the occupation at level l is lower than $1 - w_j$.

[2] Even if some recent advances in the PTAS design for $P||C_{max}$ allowed to decrease the asymptotic depedencies in $\frac{1}{\epsilon}$ (like $2^{\mathcal{O}(\frac{1}{\epsilon^2}\log^3(\frac{1}{\epsilon}))}$ in [8], the running time of these algorithms remain high due to constants.

2 General principles

In this section, we generalize the framework used in the $\frac{5}{2}$-approximation of [3]. This framework will be applied in Section 3 to get the 2-approximation.

2.1 Preliminaries

Recall that our objective is to (non contiguously) pack a set I of n rectangles r_j into N strips of width 1. Rectangle r_j has a height h_j and a width w_j. We denote by $s(r_j) = w_j h_j$ the surface of r_j. These notations are extended to $W(X)$, $H(X)$ and $S(X)$ (where X is a set of rectangles), which denote the sum of the widths (resp. heights, surfaces) of rectangles in X.

A *layer* is a set of rectangles packed one on top of the other in the same strip (as depicted Figure 3). The height of layer Lay is $H(Lay)$, the sum of the height of all the rectangles in Lay. A *shelf* is a set of rectangles that are packed in the same strip, such as the bottom level of all the rectangles is the same. Even if it is not relevant for the non-contiguous case, we consider for the sake of simplicity that in a shelf, the right side of any rectangle (except the right most one) is adjacent to the left side of the next rectangle in the shelf. Given a shelf sh (sh denotes the set of rectangles in the shelf), the value $W(sh)$ is called the *width* of sh. Packing a shelf at level l means that all the rectangles of the shelf have their bottom at level l. A *bin* is a rectangular area that can be seen as a reserved space in a particular strip for packing rectangles. As the width of a bin is always 1, we define a bin by giving its height h_b, its bottom level l_b and the index i_b of the strip it belongs to. Packing a shelf sh in a bin b means that sh is packed in strip S_{i_b} at level l_b. Moreover we always guarantee that the height of any rectangle of sh is lower than h_b.

The *utilization* $u_i^\pi(l)$ of a packing π in strip S_i at level l (sometimes simply denoted by $u(l)$ or $u_i(l)$) is the sum of the width of all the rectangles packed in S_i that cut the horizontal line-level l (see Figure 3). Of course we have $0 \leq u_i^\pi(l) \leq 1$ for any l and i.

Let us now describe three useful basic procedures. The $CreateLayer(X, h)$ procedure creates a layer Lay (using rectangles of X) of height at most h, using a Best Fit (according to the height) policy (BFH). Thus, $CreateLayer(X, h)$ adds at each step the highest rectangle that fits. Of course, the layer produced by the procedure is such that $H(Lay) \leq h$. Moreover, notice that we will always pack the layers in the strips with the narrowest rectangles on the top. $CreateShelf(X, w)$ creates a shelf sh (using rectangles of X) of width at most w, using the Best Fit (according to the width) policy (BFW). Thus, $CreateShelf(X, w)$ adds at each step the widest rectangle that fits. Of course, the shelf produced by the procedure is such that $W(sh) \leq w$. Throughout the paper, we consider that the sets of jobs used as parameters in the algorithms are modified after the calls.

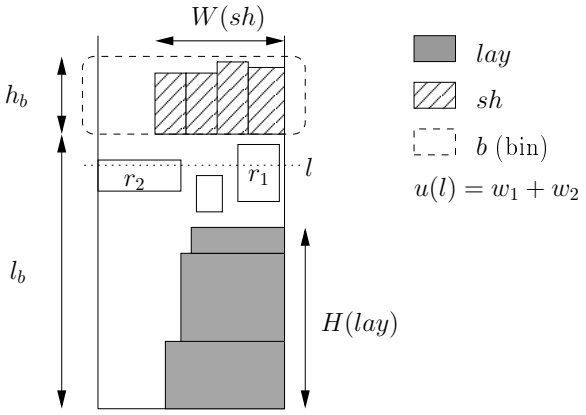

Figure 3. Example of a layer, a shelf, a bin and of the utilization function. sh is packed in b.

Let us now state a standard lemma about the efficiency of the "best fit" policies.

Lemma 2 *Let Sh denote the shelf created by $CreateShelf(X, w)$. If the k widest rectangles of X are added to Sh, then $W(Sh) > \frac{k}{k+1}w$.*

Proof Let x be the cardinality of X. Let us assume that $w_i \geq w_{i+1}$ for $1 \leq i < x$. Let $i_0 \geq k+1$ be the first index such that r_{i_0} is not in Sh. Let $a = \Sigma_{i=1}^{i_0-1} w_i$. We have $W(Sh) \geq a \geq (i_0 - 1)w_{i_0} > (i_0 - 1)(w - a)$ leading to $a > \frac{i_0-1}{i_0}w \geq \frac{k}{k+1}w$. □

2.2 Discarding technique

2.2.1 How to pack all rectangles in three steps

Discarding techniques are common for solving packing and scheduling problem. As mentioned before, the idea is to define properly a set of small items (rectangles here), and to prove that adding these small items only at the end of the algorithm will not degrade the approximation ratio. Thus, the effort can be focused on the remaining large items. In this section we present an adaptation of this general technique to the context of non-contiguous multiple strip packing. Thus, we will define a set of big rectangles $I' \subset I$, and the larger the set I', the better the approximation ratio (as the remaining small rectangles become really negligible).

In order to partition rectangles according to their height, we use the well-known dual approximation technique [7].

We denote by v the guess of the optimal value. Given an instance I, let $L_{WD} = \{w_j > \alpha\}$ be the set of wide rectangles, $L_H = \{h_j > \beta v\}$ be the set of high rectangles, and $I' = L_{WD} \cup L_H$ be the set of the big rectangles, with $0 < \alpha < 1$ and $0 < \beta < 1$. Let $r(\alpha, \beta) = \frac{1}{1-\alpha} + \beta$ be the approximation ratio we target (the origin of this formula will be explained in Section 2.2.2). We also need the following definition.

Definition 3 *A packing is x-compact if and only if for every strip S_i there exists a level l_i such that for all $l \leq l_i$, $u_i(l) > x$ and u_i restricted to $l > l_i$ is non-increasing.*

Figure 4 provides an example showing a $(1-\alpha)$ compact packing, and why step c) is simple. Indeed, adding as soon as possible a small rectangle r_j ($h_j \leq \beta v$ and $w_j \leq \alpha$) to a $(1-\alpha)$ compact packing cannot exceed $v(\frac{1}{1-\alpha} + \beta)$.

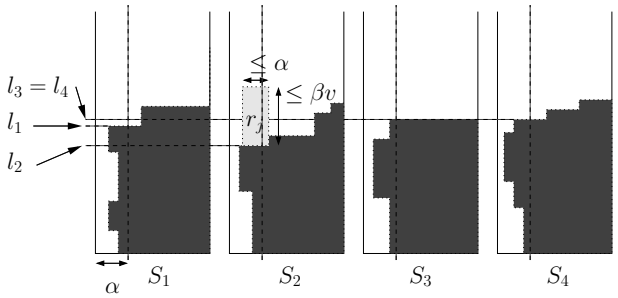

Figure 4. $(1-\alpha)$ **compact packing**

Let us now describe the three main steps of our approach. Notice that what we call a preallocation is a normal packing (*i.e.* that defines the bottom level of each rectangle, which is sufficient) that is based on simple structures like shelves and layers. We will prove that to get a $r(\alpha, \beta) = (\frac{1}{1-\alpha} + \beta)$ ratio, it is sufficient to:

a) construct a preallocation π_0 of I' that fits in $r(\alpha, \beta)v$, and such that rectangles of $L_{WD} \subset I'$ are already packed in a $(1-\alpha)$-compact way

b) turn π_0 into a $(1-\alpha)$-compact packing π_1 by repacking rectangles of $I' \setminus L_{WD}$ using the list algorithm LS_{π_0} of Lemma 4

c) add the small remaining rectangles ($I \setminus I'$) using algorithm LS (see Lemma 5)

Step a) is the most difficult one. Thus, Section 3 is entirely devoted to the construction of π_0 (targeting $\alpha = \frac{1}{3}$ and $\beta = \frac{1}{2}$). Of course, building the preallocation becomes harder when α and β are small, as the number of rectangles of I' increases and $r(\alpha, \beta)$ decreases. Roughly speaking, the simple shapes of rectangles of I' allows us to construct π_0 with a simple structure. We will denote by π_0^i the set of rectangles packed by π_0 in S_i.

2.2.2 Proving steps b) and c)

We now prove that applying steps b) and c) leads to a $r(\alpha, \beta)$ ratio. In this section, we suppose that we are given a guess v, and a packing π_0 (called the preallocation) of $I' = L_{WD} \cup L_H$ that fits in $r(\alpha, \beta)v$, and such that rectangles of $L_{WD} \subset I'$ are already packed in a $(1-\alpha)$-compact way. We consider step b): how to turn π_0 into a $(1-\alpha)$-compact packing.

Lemma 4 (Step b)) *Let π_0 be the preallocation of I' constructed in Step a). Let $\widehat{\pi_1} = \pi_0 \cap L_{WD}$ denote π_0 when keeping only rectangles of L_{WD}. Recall that $\widehat{\pi_1}$ is already a $(1-\alpha)$-compact packing of rectangles of L_{WD}.*

Then, we can complete $\widehat{\pi_1}$ into a $(1-\alpha)$-compact packing π_1 of I', such that the height of π_1 is lower or equal to the height of π_0.

Proof Let us define the LS_{π_0} algorithm that adds rectangles of $I' \setminus L_{WD}$. Let us consider a single strip S_i. Let π_0^i denote π_0 restricted to S_i, and $\widehat{\pi_1^i}$ denote $\widehat{\pi_1}$ restricted to S_i. Let $X = \{r_1, \ldots, r_p\}$ be the set of preallocated rectangles of $I' \setminus L_{WD}$ that we have to add to S_i. We assume that $lvl(j) \leq lvl(j+1)$, where $lvl(j)$ is the bottom level of r_j in π_0.

For our considered strip S_i, the LS_{π_0} algorithm executes $AddAsap(r_j, \widehat{\pi_1^i})$, for $1 \leq j \leq p$, where $AddAsap(r, \widehat{\pi_1^i})$ adds rectangle r to $\widehat{\pi_1}$ (in S_i) at the smallest possible level. Notice first that adding with $AddAsap$ a rectangle r_j with $w_j \leq \alpha$ to a $(1-\alpha)$-compact packing creates another $(1-\alpha)$-compact packing. Thus it is clear that π_1 is $(1-\alpha)$-compact.

For any $1 \leq j \leq p$, let $(\widehat{\pi_1^i}, j)$ denote the packing in S_i just before adding r_j with $AddAsap$, and let (π_0^i, j) denote the packing $\pi_0^i \cap (L_{WD} \cup \{r_1, \ldots, r_{j-1}\})$. Let us prove by induction on $j \in \{1, \ldots, p\}$ that $u^{(\widehat{\pi_1^i}, j)}(l) \leq u^{(\pi_0^i, j)}(l)$, for any $l \geq lvl(j)$. The definition of $\widehat{\pi_1}$ implies the property for $j = 1$ (packings are identical). Let us suppose that the property holds for j, and prove it for $j+1$. Let $l \geq lvl(j+1)$. The induction property for rank j implies that r_j is added by $AddAsap$ at a level lower or equal to $lvl(j)$. Thus, if r_j intersects l in $(\widehat{\pi_1^i}, j+1)$, then it also occurs in $(\pi_0^i, j+1)$. Thus in this case we have

$$\begin{aligned} u^{(\widehat{\pi_1^i}, j+1)}(l) &= u^{(\widehat{\pi_1^i}, j)}(l) + w_j \\ &\leq u^{(\pi_0^i, j)}(l) + w_j \\ &= u^{(\pi_0^i, j+1)}(l) \end{aligned}$$

If r_j does not intersect l in $(\widehat{\pi_1^i}, j)$, then clearly $u^{(\widehat{\pi_1^i}, j+1)}(l) = u^{(\widehat{\pi_1^i}, j)}(l) \leq u^{(\pi_0^i, j)}(l) \leq u^{(\pi_0^i, j+1)}(l)$

Thus we proved that for any $1 \leq j \leq p$ we have $u^{(\widehat{\pi_1^i}, j)}(l) \leq u^{(\pi_0^i, j)}(l)$ for any $l \geq lvl(j)$, implying that

every r_j is added by $AddAsap$ at a level lower or equal to $lvl(j)$. Thus, the height of π_1 is lower or equal to the height of π_0

□

We now prove in Lemma 5 that after adding rectangles in step c), the height of the packing does not exceed $r(\alpha,\beta)v = (\frac{1}{1-\alpha}+\beta)v$. This explains why the height of the pre-allocation should also be bounded by $r(\alpha,\beta)v$.

Lemma 5 (Step c)) *Let π_1 be a $(1-\alpha)$-compact packing of I'. Adding to π_1 rectangles of $I \setminus I'$ with a List Scheduling algorithm (LS) leads to a packing π having height lower than $max(height(\pi_1), v(\frac{1}{1-\alpha}+\beta))$.*

Proof The LS algorithm scans all the strips from level 0, and at any level adds any rectangle of $I \setminus I'$ that fits. Notice that the final packing π is $(1-\alpha)$-compact, since we add rectangles r_j with $w_j \leq \alpha$ to an $(1-\alpha)$-compact packing.

Let us assume that the height of π is due to a rectangle $r_j \in I \setminus I'$ that starts at level s. This implies that when packing r_j we had $l_i \geq s$ for any strip i (with l_i defined as in Definition 3). According to this definition we have $u_i(l) > 1-\alpha$ for any $l \leq l_i$. Thus, we have $S(I) > \sum_{i=1}^{N} l_i(1-\alpha) \geq N(1-\alpha)s$, implying that $s < v\frac{1}{1-\alpha}$, and thus that of height of π is lower or equal to $s + max_{j\in I\setminus I'} h_j \leq v(\frac{1}{1-\alpha}+\beta)$. □

Thus, we now apply this framework with $\alpha = \frac{1}{3}$ and $\beta = \frac{1}{2}$ to get a 2-approximation.

3 2-approximation

3.1 Hardness

As explained before, the $2+\epsilon$-approximation in [10] and the 2-approximation we recently proposed in [2] are rather complexity results than practical algorithms. We aim at constructing a low cost algorithm that could be used in a practical context. Thus, we are considering a restriction of $MCSP$ where the inapproximability bound could be tightened with a fast algorithm, and we consider that all the rectangles have a width lower or equal to $1/2$.

Theorem 6 *The $MCSP$ where every rectangle has a width lower (or equal) to $\frac{1}{2}$ has no polynomial algorithm with a ratio strictly better than 2, unless $P = NP$.*

Proof As in [11] for the general version, we construct a gap reduction from 2-partition. Let $\{x_1,\ldots,x_n\} \subset \mathbb{N}^n$ and a such that $\sum_{i=1}^{n} = 2a$. Without loss of generality, let us assume that for any i, $x_i < a$. In order to only have items with size at least two, we define $x'_i = 2x_i$ for any i, and $a' = 2a$. We construct the following instance I_{MSP}: $N = 2$ strips, each of size $2a'-1$. The set of rectangles is $\{r_1,\ldots,r_n,r_{n+1},r_{n+2}\}$, with $w_i = x'_i$ for $1 \leq i \leq n$, $w_{n+1} = w_{n+2} = a'-1$, and $h_i = 1$ for $1 \leq i \leq n+2$. We have $w_i \leq \frac{2a'-1}{2}$ for any i, as all the x_i are strictly lower than a. Notice than any solution of I_{MSP} that packs r_{n+1} and r_{n+2} is the same strip has a height of at least 2, as the available width of size 1 in that strip cannot be used by any rectangle.

Obviously, if there is a 2-partition, then $Opt(I_{MSP}) = 1$. Otherwise, as r_{n+1} and r_{n+2} cannot be packed together, we have $Opt(I_{MSP}) = 2$ □

The previous proof can easily be adapted for any non-trivial restriction on the size of the widest rectangle. Therefore, the fast 2-approximation presented in this section is the best possible result, even for more restricted versions of the MCSP.

3.2 Decomposition

We follow the ideas presented in Section 2, and thus we re-use the notion of **layer**, **shelf**, **bin**, and the procedures named **CreateLayer** and **CreateShelf**.

Again, we use the dual approximation technique [7], and we denote by v the guess of the optimal value. Conforming to the dual approximation technique, we will prove that either we pack I with a resulting height lower than $2v$, or $v < Opt$. Then, we will perform a binary search on v to turn the dual approximation algorithm into a classical approximation algorithm. Notice that for the sake of simplicity we did not add the "reject" instructions in the algorithm. Thus we consider in all the proof that $v \geq Opt$, and it is implicit that if one of the claimed properties is wrong during the execution, the considered v should be rejected.

Recall that all rectangles have $w_j \leq \frac{1}{2}$. Let us define the following sets:

- let $L_{WD} = \{r_j | w_j > 1/3\}$ be the set of wide rectangles
- let $L_{XH} = \{r_j | h_j > 2v/3\}$ be the set of very high rectangles
- let $L_H = \{r_j | 2v/3 \geq h_j > v/2\}$ be the set of high rectangles
- let $L_B = (L_{XH} \cup L_H) \cap L_{WD}$ be the set of huge rectangles, and $b = Card(L_B)$.
- let $I' = L_{WD} \cup L_{XH} \cup L_H$

Notice that we only consider the values v such that $W(L_{XH} \cup L_H) \leq N$ and $H(L_{WD}) \leq 2Nv$.

As expected, the set I' corresponds in our framework to the set of big rectangles for $\alpha = \frac{1}{3}$ and $\beta = \frac{1}{2}$. The

construction of the preallocation π_0 of I' is presented from Section 3.3 to 3.5. The final steps to turn π_0 into a $\frac{2}{3}$-compact packing π_1 and to turn π_1 into the final packing π are quickly described in Section 3.6, as they follow the steps presented in Section 2.2.

We now provide a two phases algorithm that builds the preallocation π_0 of the rectangles of I'. Phase 1 (Section 3.3) preallocates rectangles of L_{WD}, and phase 2 (Section 3.5) preallocates rectangles of $L_H \cup L_{XH}$.

3.3 Phase 1

Phase 1 packs the rectangles of L_{WD} by calling for each strip (until L_{WD} is empty) two times $CreateLayer(L_{WD}, 2v)$. Let us denote by Lay_{2i-1} and Lay_{2i} the layers created in strip S_i. Let us say that Lay_{2i-1} is packed left justified, and Lay_{2i} is packed right justified. Moreover, each layer is repacked in non increasing order of the widths, such that the narrowest rectangles are packed on the top.

Let N_1 denote the number of strips used in phase 1, and let i_1 denote the index of the last created layer (Lay_{i_1} is of course in S_{N_1}). Let L^1_H and L^1_{XH} denote the set of remaining rectangles after phase 1 of L_H and L_{XH}, respectively. Thus, for the moment we have $\pi^i_0 = Lay_{2i} \cup Lay_{2i-1}$ for all $i \leq N_1$.

Lemma 7 *If $\exists i_0 < i_1$ such that $H(Lay_{i_0}) \leq \frac{3v}{2}$ then it is straightforward to preallocate I'.*

Proof Let $i_0 < i_1$ such that $H(Lay_{i_0}) \leq \frac{3v}{2}$. This implies that we ran out of rectangles of $L_{WD} \setminus (L_H \cup L_{XH})$ while creating layer i_0. Thus, because of the BFH order there are at least two rectangles of L_B in every layer Lay_i, for $1 \leq i < i_1$, implying that the width of high and very high rectangles packed in each of these layers is strictly larger than $2/3$. Thus, $W(\pi^i_0 \cap (L_H \cup L_{XH})) > 4/3 > 1$ for $1 \leq i < N_1$. Thus, the total width of remaining high and very high rectangles is lower than $N - (N_1 - 1)$.

Let us prove that we can pack all the remaining rectangles of I' (which are included in $(L_H \cup L_{XH})$) in the remaining strips. For each $i \in [|N_1 + 1, N|]$ we create two shelves in S_i (one at level 0 and one at level v). If there are still some unpacked rectangles, then all the shelves are "full", that is the width of each shelf is larger than $2/3$ (as all the width of any rectangle of $L_H \cup L_{XH}$ is lower than $1/3$). Thus, we have $W(\pi^i_0 \cap (L_H \cup L_{XH})) > 4/3 > 1$ (for $N_1 + 1 \leq i \leq N$). This implies that the total width of remaining rectangles of $L_H \cup L_{XH}$ (including those in strip S_{N_1}) is now lower than 1. Thus, we can pack all of them in one shelf in S_{N_1}. □

From now we assume that $H(Lay_i) > \frac{3v}{2}$ for all $i < i_1$. This implies that $S(\pi^i_0) > v$ for $i < N_1$. Moreover, we have $2Nv \geq H(L_{WD}) \geq \sum_{i=1}^{N_1-1} H(\pi^i_0 \cap L_{WD}) > (N_1 - 1)2(3v/2)$, implying $N_1 < \frac{2}{3}N + 1$.

It remains now to pack $L^1_H \cup L^1_{XH}$. Notice that $(L^1_H \cup L^1_{XH}) \cap L_{WD} = \emptyset$ (we say that $(L^1_H \cup L^1_{XH})$ contains purely high and very high rectangles).

3.4 Packing high and very high rectangles

3.4.1 Preliminaries

Let $N_2 = N - N_1$ denote the number of free strips after phase 1. Roughly speaking, phase 2 packs shelves of high or very high rectangles in each of the N_2 last strips and merges some high or very high rectangles with the ones packed in strip N_1 (using the $Merge$ procedure).

In this section we present a technique to fill γ empty strips with high or very high rectangles. In the Section 3.5, we use this technique for $\gamma = N_2$ (using strips $S_{N_1+1} \ldots S_N$) and an additional merging algorithm (that fills efficiently strip S_{N_1}) to pack $L^1_H \cup L^1_{XH}$.

Let us now introduce the procedure **GreedyPack(X, seq)**. Given an ordered sequence of bins seq, $GreedyPack$ creates for each *empty* bin $b \in seq$ a shelf of rectangles of X using $CreateShelf(X, 1)$ and packs it into b (an example of a shelf packed in a bin is depicted Figure 3, Page 3). This procedure returns the last bin in which a shelf has been created, or *null* if no shelf is created. Notice that we will always use sequence of bins that have always width 1, and the same height h_b such that $max_{r_x \in X} h_x \leq h_b$.

We now define the two sequences of bins seq_{XH} and seq_H that will be used by $GreedyPack$. Every bin of seq_{XH} (resp. seq_H) will (possibly) contain one shelf of rectangles of L_{XH} (resp. L_H). Notice that in a free strip it is possible to pack two bins of height v (width of bins is always 1), three bins of height $2v/3$, or one bin of size v and one bin of size $2v/3$. Thus, seq_{XH} is composed of 2γ bins $(b_1, \ldots, b_{2\gamma})$ of height v, considering that we created two bins of height one in each of the strips $S_1, \ldots S_\gamma$. More precisely, for all i we locate b_{2i-1} and b_{2i} in S_i, with b_{2i-x} at level $v(1-x)$ for $x \in \{0, 1\}$. The sequence seq_H is composed of 3γ bins $(b'_1, \ldots, b'_{3\gamma})$ of height $2v/3$, considering that we created three bins in each of the strips $S_\gamma, \ldots S_1$. It means that for all $i \geq 1$, bins b'_{3i-2}, b'_{3i-1} and b'_{3i} are located in $S_{\gamma-i+1}$, with b'_{3i-x} at level $\frac{2xv}{3}$ for $x \in [|0, 2|]$. This sequences of bins will be used in Lemma 9, and later in phase 2.

Finally, let us define the $Add(X, S_{i_{last}})$ procedure that packs the set of rectangles $X \subset L_H \setminus L_{WD}$ in $S_{i_{last}}$. As one can see in Lemma 9, $S_{i_{last}}$ is the last strip where $Greedypack$ created a shelf. Thus, we assume for the moment that $S_{i_{last}}$ may only contain two different shapes of packing, and define the Add procedure accordingly.

In the first case $S_{i_{last}}$ contains a first "full" shelf (full means that the surface of the shelf is at least $v/2$) of rectangles of L_{XH} at level 0, and a shelf sh of rectangles of L_{XH} packed at level v, right justified. In this case, Add creates a shelf sh_1 using $CreateShelf(X, 1 - W(sh))$ and preallocate sh_1 at level v, left justified.

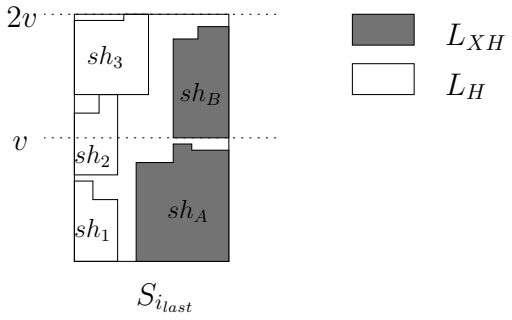

Figure 5. Example of the add procedure.

In the second case (see Figure 5), $S_{i_{last}}$ contains only a shelf of rectangles of L_{XH} packed at level 0, right justified. In this case, Add first moves some rectangles from sh to a new shelf $\widehat{sh}$ until $W(sh) \leq 2/3$. Then, Add packs (right justified) the widest of these two shelves (denoted by sh_A) at level 0, and the other one (denoted by sh_B) at level v. Finally, Add creates two shelves sh_1 and sh_2 using $CreateShelf(X, 1 - W(sh_A))$ and one shelf sh_3 using $CreateShelf(X, 1 - W(sh_B))$. Then, sh_i is packed at level $\frac{2v(i-1)}{3}$, left justified. Notice that stacking shelves sh_1, sh_2, sh_3 does not exceed $2v$.

We end these preliminaries with the following Lemma about the efficiency of Add.

Lemma 8 *Let $X \subset L_H \setminus L_{WD}$ and S_i be a strip packed as expected for $Add(X, S_i)$. Let π_0^i denote the rectangles packed in S_i before the call $Add(X, S_i)$. If $X \neq \emptyset$ after calling the procedure, then $S(\pi_0^i \cup X) > v$.*

Proof Remember that two cases are possible according to what is already packed in S_i before the call. Let us first suppose that there is one full shelf (of area strictly larger than $v/2$) of very high rectangles (at level 0) and another shelf sh of very high rectangles at level v. Then, $X \neq \emptyset$ after the call implies that $W(X) > 1 - W(sh)$, and we have $S(\pi_0^i \cup X) > \frac{v}{2} + W(sh)\frac{2v}{3} + W(X)\frac{v}{2} > v$.

Let us now suppose that S_i contains only one shelf sh of L_{XH} at level 0. Let $sh_A, sh_B, sh_1, sh_2, sh_3$ be defined as described in the Add procedure. As $W(sh_A) \leq \frac{2}{3}$ ($W(sh_b) \leq \frac{2}{3}$ is also true), and $X \cap L_{WD} = \emptyset$, sh_1 and sh_2 contain at least one rectangle, implying that $W(sh_1)$ and $W(sh_2)$ are strictly larger than $\frac{1-W(sh_A)}{2}$ according to Lemma 2. Moreover, $X \neq \emptyset$ after the call implies that after creating sh_1 and sh_2 the total width of remaining rectangles of X was strictly larger than $1 - W(sh_B)$. Putting this together, we get $S(\pi_0^i \cup X) > (1 - W(sh_A))\frac{v}{2} + (1 - W(sh_B))\frac{v}{2} + (W(sh_A) + W(sh_B))\frac{2v}{3} > v$.

□

3.4.2 Filling γ empty strips with high and very high rectangles

The next lemma shows how to fill γ free strips.

Lemma 9 *Let $\widehat{L_{XH}} \subset L_{XH} \setminus L_{WD}$ and $\widehat{L_H} \subset L_H \setminus L_{WD}$ be two sets of rectangles that we have to pack. Suppose that we execute the following calls:*

1. $last = GreedyPack(\widehat{L_{XH}}, seq_{XH})$
2. $GreedyPack(\widehat{L_H}, seq_H)$
3. $Add(\widehat{L_H}, S_{i_{last}})$ *where $S_{i_{last}}$ denotes the strip containing the bin "$last$".*

Then, we get the following properties:

- *If $\widehat{L_{XH}} \neq \emptyset$ after 1, then $S(\widehat{L_{XH}}) > (\gamma + \frac{1}{6})v$*
- *Otherwise, if $\widehat{L_H} \neq \emptyset$ after 3, then $S(\widehat{L_{XH}} \cup \widehat{L_H}) > \gamma v$.*

Remark 10 *Let X such that $X \cap L_{WD} = \emptyset$ and let sh denote a shelf created by $CreateShelf(X, 1)$, supposing that we did not run out of rectangle while creating the shelf. Then, according to Lemma 2, as at least three rectangles fit we have $W(sh) > 3/4$. Moreover, if $X \subset L_{XH}$ then $S(sh) > v/2$, and if $X \subset L_H$ then $S(sh) > 3v/8$.*

Proof [Proof of lemma 9] Let us first suppose that $\widehat{L_{XH}} \neq \emptyset$ after step 1. It implies that after creating the first $2\gamma - 1$ shelves of width at least $3/4$ (according to Remark 10), the total remaining width of rectangles of $\widehat{L_{XH}}$ was strictly larger than 1. Thus, $S(\widehat{L_{XH}}) > \frac{3}{4}(2\gamma - 1)\frac{2v}{3} + \frac{2v}{3} = (\gamma + \frac{1}{6})v$.

Let us now suppose that $\widehat{L_{XH}} = \emptyset$ and $\widehat{L_H} \neq \emptyset$ after step 3. Let sh denote the shelf of rectangles of $\widehat{L_{XH}}$ contained in bin $last$. Remind that i_{last} is the index of the strip containing $last$. For all $i \in [|1, i_{last} - 1|]$, $S(\pi_0^i \cap (\widehat{L_{XH}} \cup \widehat{L_H})) > 2\frac{v}{2} = v$. For all $i \in [|i_{last} + 1, \gamma|]$, $S(\pi_0^i \cap (\widehat{L_{XH}} \cup \widehat{L_H})) > 3\frac{3v}{8} > v$. According to Lemma 8, $\widehat{L_H} \neq \emptyset$ implies $S(\pi_0^{i_{last}} \cap \widehat{L_H}) > v$. Thus, we get that $S(\widehat{L_{XH}} \cup \widehat{L_H}) > \gamma v$. □

3.5 Phase 2

In phase 1 we preallocated L_{WD} in strips $S_1, \ldots, S_{N_1}$. Recall that each layer created in phase 1 is sorted with the narrowest rectangles on the top. It remains now to preallocate $L^1_{XH} \cup L^1_H$ in $S_{N_1}, \ldots, S_N$.

Theorem 11 *It is possible to preallocate $L^1_{XH} \cup L^1_H$ in $S_{N_1}, \ldots, S_N$ with a resulting height lower than $2v$. Thus, our algorithm is a 2-approximation.*

Proof Due to lack of space, we only sketch the proof of the theorem here and refer the reader to [4] for the complete case analysis.

An example of the final packing is depicted Figure 6. Phase 1 optimally filled the $(N_1 - 1)$ first strips using rectangles of L_{WD}. Phase 2 optimally filled strips $N_1 + 1$ to $i_{last} - 1$ using two shelves of very high rectangles in each strip, and optimally filled strips $i_{last} + 1$ to N using three shelves of high rectangles in each strip. Strips N_1 and i_{last} will be carefully filled according to a case by case analysis. According to Section 3.3, we know that the area packed in the $N_1 - 1$ first strips is greater than $(N_1 - 1)v$. The general idea is to pack $L^1_{XH} \cup L^1_H$ is to call $GreedyPack$ (see Lemma 9) on strips $S_{N_1+1}, \ldots, S_N$. If L^1_{XH} and L^1_H are entirely packed by $GreedyPack$, then all the rectangles are packed in $2v$. Otherwise, we can use Lemma 9 to claim that, $S(L^1_{XH} \cup L^1_H) > (N - N_1)v$ (or $(N - N_1 + \frac{1}{6})v$), and the remaining rectangles are packed by carefully studying what was packed in S_{N_1} by phase 1. Thus, we finish the proof using a case distinction according to the shape of the preallocation in S_{N_1}. □

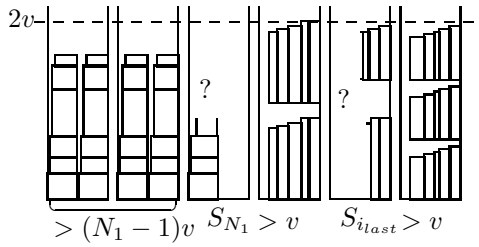

Figure 6. Example of a preallocation.

3.6 Concluding remarks

According to the main steps defined in Section 2.2, the previous 2 phases algorithm that preallocates I' is sufficient to get a 2-approximation. Indeed, we simply add rectangles of $I \setminus I'$ using list algorithms defined in Section 2.2.

Let us now sketch the analysis of the running time of the preallocation algorithm. Phase 1 and Phase 2 run in $\mathcal{O}(n \log(n) + Nn)$ as for each strip creating a layer or a shelf can be done in $\mathcal{O}(n)$ (by sorting wide rectangles according to their heights before phase 1, and sorting high rectangles according to their widths before phase 2). The list scheduling algorithms of Section 2.2.2 that turn π_0 into the final packing can be implemented in $\mathcal{O}(n \log(n))$ by using a global list (*i.e.* that refers to the N strips) of currently "scheduled" rectangles, instead of scanning level by level and strip by strip (which is in $\mathcal{O}(Nn \log(n))$).

Finally, taking into account the repetitions due to the binary search on v (where $0 \leq v \leq nh_{max}$), the overall algorithm runs in $O(\log(nh_{max})n(N + \log(n)))$.

References

[1] N. Bansal, A. Caprara, K. Jansen, L. Prdel, and M. Sviridenko. How to maximize the total area of rectangle packed into a rectangle. In *ISAAC*, 2009.

[2] M. Bougeret, P.-F. Dutot, K. Jansen, C. Otte, and D. Trystram. Approximation algorithm for multiple strip packing. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms (WAOA)*, 2009.

[3] M. Bougeret, P.-F. Dutot, K. Jansen, C. Otte, and D. Trystram. Approximating the non-contiguous multiple organization packing problem. In *Proceedings of the 6th IFIP International Conference on Theoretical Computer Science (TCS)*, 2010.

[4] M. Bougeret, P.-F. Dutot, K. Jansen, C. Robenek, and D. Trystram. Tight approximation for scheduling parallel jobs on identical clusters. LIRMM research report 12001 `http://hal-lirmm.ccsd.cnrs.fr/lirmm-00656780`.

[5] P.-F. Dutot and D. Trystram. Scheduling on hierarchical clusters using malleable tasks. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 199–208. ACM Press, 2001.

[6] D. Hochbaum and D. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

[7] D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[8] K. Jansen. An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. *Automata, Languages and Programming*, pages 562–573, 2009.

[9] U. Schwiegelshohn, A. Tchernykh, and R. Yahyapour. Online scheduling in grids. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–10, 2008.

[10] D. Ye, X. Han, and G. Zhang. On-Line Multiple-Strip Packing. In *Proceedings of the 3rd International Conference on Combinatorial Optimization and Applications (COCOA)*, page 165. Springer, 2009.

[11] S. Zhuk. Approximate algorithms to pack rectangles into several strips. *Discrete Mathematics and Applications*, 16(1):73–85, 2006.