# Approximation algorithms for Job Scheduling with reconfigurable resources

Pierre Bergé, Mari Chaikovskaia, Jean-Philippe Lucien Gayon, Alain Quilliot

# Approximation Algorithms for Job Scheduling with Reconfigurable Resources

Pierre Bergé[1], Mari Chaikovskaia[2], Jean-Philippe Gayon[1], and Alain Quilliot[1]

[1]Université Clermont-Auvergne, CNRS, Mines de Saint-Etienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France
[2]IMT Atlantique, LS2N, UMR CNRS 6004, F-44307 Nantes, France

**Abstract**

We consider a scheduling problem with reconfigurable resources. Several types of jobs (e.g. transportation tasks, production operations) have to be processed by a set of identical resources (e.g. robots, workers, processors) over a discrete time horizon. In each time period, teams of resources must be formed to process jobs. During a given time period, a team handles one type of job and the number of jobs that can be processed depends on the team size. A resource which is used to perform some job type in a given period may be employed for another job type in the next period. The objective is to determine the minimum number of resources needed to meet a given demand for each job type. Although this problem stems from a logistics application with reconfigurable robots, we focus here on fundamental issues. We show that the resulting strongly NP-hard **Multi_Bot** problem may be handled in a greedy way with an approximation ratio of 4/3.

**Keywords :** Scheduling, Approximation, Complexity

## 1 Introduction

In many industrial contexts, automatized production must adapt itself to a fast evolving demand of a large variety of customized products. One achieves such a flexibility requirement thanks to *reconfiguration*. Once an operation has been performed, related either to the production of some good or to its transportation, one may redesign the infrastructure that supported this operation by adding, removing or replacing some atomic components, or by modifying the links that connect those components together. Those components may be hardware (robots, instruments), software or human resources. They behave as

renewable resources [2, 3, 11] and move inside the production area in order to fit, during a given production cycle, with current production/transportation needs. Depending on the way one assigns those resources to a given operation, one may not only achieve this operation but also speed it, increase its throughput or lessen its cost, as in the **multi-modal Resource Constrained Project Scheduling Problem** [2].

We consider the following scheduling problem with reconfigurable resources. Several types of jobs (e.g. production operations, transportation tasks) have to be processed by a set of identical resources (e.g. workers, robots, processors) over a discrete time horizon in order to achieve a certain demand. Assigning a number $p$ of resources to some job of type $k$ gives a certain production $c_{pk}$: all production values, called *capacities*, are given as inputs. In each time period, teams of resources must be formed to process jobs. A resource which is used to perform some type of job $k$ at period $t$ may be employed for another type of job $k' \neq k$ in the next period $t+1$. The objective is to determine the minimum number of resources needed to obtain a certain production for each type of job. This problem is called **Multi_Bot** and is strongly NP-hard [6].

It was first introduced in a warehouse logistics context [7] in collaboration with the MecaBotiX company [15] which designs reconfigurable mobile robots. In this application, resources are mobile robots and jobs consist in moving loads of various types, such as pallets or boxes. Those robots are capable of aggregating into a cluster to form poly-robots that can adapt to the type of product (size/mass) and navigate independently in environments such as warehouses, production sites or construction sites. Other applications can be found in the automotive industry where the resources are workers and the processing time for a task in the assembly line depends on the number of workers assigned to it [1].

From a theoretical point of view, **Multi_Bot** is strongly related to some classical scheduling problems of the literature. A very well-known one is **Identical-machines scheduling** (**IMS**), where the objective is to pack items into a set of identical boxes while minimizing the size of the most filled box [10]. The standard scheduling notation of **IMS** is $P||C_{\max}$. Its high multiplicity variant, denoted by $P|HM(n)|C_{\max}$, encodes as binary inputs the number of items with the same size [4]. These two scheduling problems have been widely studied in terms of approximation and parameterized complexity [5, 8, 12, 13, 14, 16, 17]. Problem $P|HM(n)|C_{\max}$, when the item sizes are polynomially bounded, is a special case of **Multi_Bot**: consider that any type of job $k$ correspond to some item size and can be performed only by a specific number of resource $p_k$, the demands of jobs of type $k$ correspond to the high-multiplicity coefficients of the related items. In this article, we will use a heuristic of **IMS** as a sub-routine of our algorithm.

The main result of this paper is the presentation of a polynomial-time $\frac{4}{3}$-approximation algorithm for **Multi_Bot**. The impact of this contribution is, in our opinion, twofold. On one hand, we provide, for industrial applications such as MecaBotiX robots, a fast and efficient heuristic with the strict guarantee that it will not fail on pathological instances more than 33% over the optimum.

On the other hand, we extend the theoretical knowledge on approximability of scheduling problems, showing that a generalization of $P|HM(n)|C_{\max}$ admits a constant approximation ratio.

The paper is organized as follows. In Section 2, we detail the **Multi_Bot** problem, remind its ILP formulation and introduce the notions of *schedule* and *packing*. Next we provide in Section 3 the definition of **Identical-machines scheduling** (**IMS**) problem as well as some preliminary observations on the optimum solutions of **Multi_Bot**. Section 4 is devoted to the description of our approximation algorithm. This algorithm relies on a polynomial time dynamic programming algorithm that solves **Multi_Bot** when the periods are merged into a single macro-period: this is described in Section 5.

# 2 Problem description and notations

We consider a set of identical resources (for instance robots, workers, etc) which cooperate in order to process different types of jobs. A $p$-resource is a *configuration* which makes $p$ resources cooperate on the same job. For example, in robotics, it models the fact that $p$ elementary robots can assemble together in order to transport heavy loads. A maximum of $P$ resources may cooperate. The set of configurations is thus $\mathcal{P} = \{1,\ldots,P\}$. There are $K$ job types and let $\mathcal{K} = \{1,\ldots,K\}$. The demand for type $k$ of jobs is denoted by $d_k$ for every type $k \in \mathcal{K}$, and $d_{\max}$ is the maximum demand. In robotics, it models the fact that there exists $K$ types of loads (pallets, boxes...) and that at least $d_k$ loads of type $k$ have to be transported.

All tasks must be executed within a discrete time horizon $\mathcal{T} = \{1,\ldots,T\}$. At the beginning of each period $t \in \mathcal{T}$, the resources may be reconfigured in order to provide us with numerous configurations which perform jobs for period $t$. During a given period $t$, a $p$-resource can deal with only a single type $k \in \mathcal{K}$, and its production is given by the capacity $c_{pk}$. For each type $k \in \mathcal{K}$, there is at least one value $p$ such that $c_{pk}$ is non-zero, *i.e.* at least one $p$-resource is able to process a job of type $k$. Our purpose is to minimize the number of resources used into the whole process. If $H_t$ denotes the number of active resources during period $t$, then the number of resources necessary to achieve the whole process is $H = \max_{t\in\mathcal{T}} H_t$. Let **Multi_Bot** refer to this optimization problem.

## 2.1 ILP formulation for Multi_Bot

More formally, we provide in Problem 1 the integer linear programming (ILP) formulation of **Multi_Bot**. We denote by $x_{pkt}$ the decision variable representing the number of jobs of type $k$ performed in configuration $p$ at period $t$.

Constraint (2) means that we must have a sufficient total capacity to satisfy demand $d_k$. Quantity $\sum_{t\in\mathcal{T}}\sum_{p\in\mathcal{P}} c_{pk}\cdot x_{pkt}$ represents the maximum number of jobs of type $k$ that can be processed over the horizon, given the $x_{pkt}$. Constraint (3) means that the number of required resources in period $t$ can be written as

$H_t = \sum_{p,k} p \cdot x_{pkt}$. Constraint (4) recalls that all decision variables are non negative integers.

Figure 1 illustrates an optimal solution (*i.e.* which minimizes $H$) for the following instance of **Multi_Bot**. There are two types of jobs ($K = 2$) and three periods ($T = 3$). The maximum size of a performed job is $P = 5$. The demands are $d_1 = 13$ and $d_2 = 10$. The capacities for jobs of type 1 are $c_{11} = 1$ and $c_{21} = 4$. The demand $d_1 = 13$ is achieved as the schedule contains three 2-resources (total production 12) and one 1-resource (production 1) handling jobs of type $k = 1$. The capacities for jobs of type 2 are linear in $p$: for any $1 \leq p \leq P$, $c_{p2} = p$. One can check the demand for $k = 2$ is also reached. Furthermore, it is not possible to find a solution which achieves $H = 5$.

**Problem 1** (**Multi_Bot**)**.**

$$
\begin{aligned}
\textbf{Input:}\quad & \mathcal{K} = \{1, \ldots, K\}, \mathcal{P} = \{1, \ldots, P\} \\
& \mathcal{T} = \{1, \ldots, T\} \\
& \text{Capacities } (c_{pk})_{\substack{p \in \mathcal{P} \\ k \in \mathcal{K}}} \\
& \text{Demands } (d_k)_{k \in \mathcal{K}} \\
\textbf{Objective:}\quad & \text{minimize } H = \max_{t \in \mathcal{T}} H_t && (1) \\
\textbf{subject to:}\quad & \sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}} c_{pk} \cdot x_{pkt} \geq d_k && \forall k \in \mathcal{K} \quad (2) \\
& H_t = \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}} p \cdot x_{pkt} && \forall t \in \mathcal{T} \quad (3) \\
& x_{pkt}, H \in \mathbb{N} && \forall k \in \mathcal{K}, p \in \mathcal{P}, t \in \mathcal{T} \quad (4)
\end{aligned}
$$

## 2.2 Schedules and packings

A solution of the **Multi_Bot** problem is thus a vector of size $PKT$:

$$\mathbf{x} = (x_{pkt})_{\substack{p \in \mathcal{P} \\ k \in \mathcal{K} \\ t \in \mathcal{T}}}$$

In the remainder, we abuse notation when the context is clear: the same vector could be denoted implicitly by $(x_{pkt})_{p,k,t}$ to gain some space. Also, notations $x_{pkt}$ and $x_{p,k,t}$ refer to the same value.

The input size of **Multi_Bot** is $O\left(T + KP(\log c_{\max} + \log d_{\max})\right)$. Concretely, values $K, P, T$ can be seen as combinatorial inputs while demands and capacities are numerical values. Hence, values $c_{pk}$ and $d_k$ might be exponential in the input size, but also $x_{pkt}$, $H_t$ and $H$. However, observe that the size of a solution $\mathbf{x}$ is polynomial in the input size. We call such a solution a *schedule* of the **Multi_Bot** instance.

Given an instance of **Multi_Bot**,

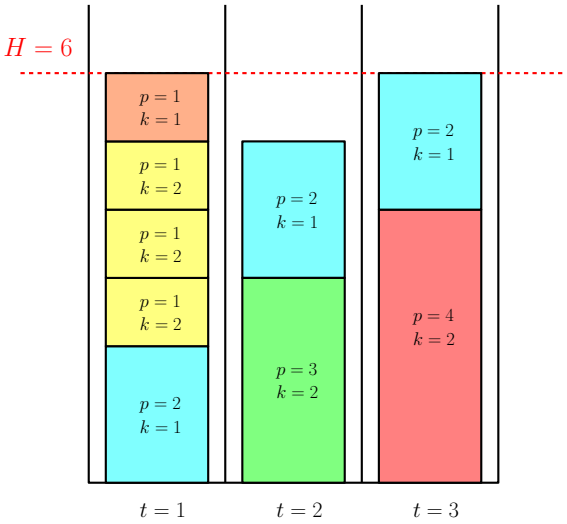


Figure 1: An example of schedule $\mathbf{x} = (x_{pkt})_{p,k,t}$ with $H = \max_t H_t = 6$, meaning that at most 6 resources are used per period.

- let $H^*$ denote the optimum value for the objective function $H$,
- let $\mathbf{x}^* = \left(x^*_{pkt}\right)_{p,k,t}$ be a schedule reaching this optimum.

In comparison with a schedule $\mathbf{x}$ which is a vector with $KPT$ values, a *packing* $\boldsymbol{\pi} = (\pi_{pk})_{\substack{p\in\mathcal{P}\\k\in\mathcal{K}}}$ represents the schedule of jobs of type $k$ by $p$-resources, independently from any time consideration. For example, it can be used to describe the production process during one specific period. This is a vector with $KP$ values. Given some schedule $\mathbf{x} = (x_{pkt})_{p,k,t}$, its *associated packing* is simply given by all values $\pi_{pk} = \sum_{t\in\mathcal{T}} x_{pkt}$, for all $p \in \mathcal{P}, k \in \mathcal{K}$. In particular, the packing associated with the optimal schedule $\mathbf{x}^*$ is denoted by $\boldsymbol{\pi}^*$ :

$$\boldsymbol{\pi}^* = \left(\sum_{t\in\mathcal{T}} x^*_{pkt}\right)_{\substack{p\in\mathcal{P}\\k\in\mathcal{K}}}$$

We pursue with the definition of measures for both schedules and packings.

**Definition 1** (Volume)**.** *The* volume *of a packing is the total number of resources involved in this packing. Formally, for* $\boldsymbol{\pi} = (\pi_{pk})_{p,k}$*,*

$$\mathrm{vol}(\boldsymbol{\pi}) = \sum_{k\in\mathcal{K}} \sum_{p\in\mathcal{P}} p \cdot \pi_{pk}$$

*For the sake of simplicity, we also use this notion for schedules. The volume of a schedule* $\mathbf{x}$ *is the volume of its associated packing,* i.e. $\sum_{p,k,t} p \cdot x_{pkt}$*.*

**Definition 2** (Maximum)**.** *The maximum of a packing* $\boldsymbol{\pi}$ *is the maximum* $p \in \mathcal{P}$ *such that some* $\pi_{pk}$ *is non-zero. Formally,*

$$\max(\boldsymbol{\pi}) = \max \{p \in \mathcal{P} : \pi_{pk} > 0 \text{ for some } k\}$$

.

Eventually, we define another measure on packings that we will use in our approximation algorithm: the *scale*.

**Definition 3** (Scale)**.** *Given some integer parameter* $\lambda$*, let us call the* big configurations *the* $p$*-resources with* $p > \frac{2\lambda}{3}$ *and the* medium configurations *the* $p$*-resources with* $\frac{\lambda}{3} < p \leq \frac{2\lambda}{3}$*. Naturally, the* small configurations *refer to* $p \leq \frac{\lambda}{3}$*. We define the* $\lambda$-scale *as the number of jobs performed with big configurations in packing* $\boldsymbol{\pi}$ *plus half the number of jobs performed with medium configurations in* $\boldsymbol{\pi}$*. It is a half-integer:*

$$\lambda\text{-}scale(\boldsymbol{\pi}) = \sum_{k \in \mathcal{K}} \sum_{p > \frac{2\lambda}{3}} \pi_{pk} + \frac{1}{2} \sum_{k \in \mathcal{K}} \sum_{\frac{\lambda}{3} < p \leq \frac{2\lambda}{3}} \pi_{pk} \tag{5}$$

Given a solution $\mathbf{x}$ of **Multi_Bot** using $H$ resources, its associated packing $\boldsymbol{\pi}$ has a limited $H$-scale. Indeed, looking at how much medium or big configurations a period can contain, we see that it has at most either one big configuration or two medium configurations. For example, a period cannot contain both a big and a medium configuration, by definition. Hence, $\lambda$-scale$(\boldsymbol{\pi}) \leq T$.

As an example, for the packing $\boldsymbol{\pi}$ associated to the schedule proposed in Figure 1, we have $\text{vol}(\boldsymbol{\pi}) = 17$, $\max(\boldsymbol{\pi}) = 4$ and $5\text{-scale}(\boldsymbol{\pi}) = 1 + 4 \cdot \frac{1}{2} = 3$.

# 3 Preliminaries

## 3.1 Approximation algorithms

Unfortunately, **Multi_Bot** is strongly NP-complete for the general case because it can be reduced from **Bin Packing** [6]. Consequently, assuming P≠NP, there is no polynomial-time exact algorithm for **Multi_Bot**, even if the numerical values are supposed to be polynomially-bounded by the input size. A very natural question is thus the approximability of this problem.

An $r$-approximation algorithm, $r \geq 1$, for **Multi_Bot** is a polynomial-time algorithm which outputs a solution $\mathbf{x} = (x_{pkt})_{p,k,t}$ such that:

$$H = \max_{t \in \mathcal{T}} \left( \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}} p \cdot x_{pkt} \right) \leq r \cdot H^*$$

Approximation algorithms offer the guarantee that the number of resources used by the solution returned is at most a linear function of the optimum.

### 3.2 Identical-machines scheduling

We recall the definition and some results related to a well-known problem in operations research: **Identical-machines scheduling** [10]. This problem can be seen as the optimization of **Bin Packing** by the capacities, where the number of boxes is fixed. In the scheduling framework, **IMS** corresponds to $P||C_{\max}$. Its objective is to assign a set of $n$ tasks given with their processing times to $m$ identical machines such that the makespan is minimized. To distinguish **IMS** with **Multi_Bot**, we will use a slightly different syntax. Formally, we are given a set of items $\mathcal{I} = \{1, \ldots, n\}$ and a set of boxes $\mathcal{B} = \{1, \ldots, m\}$. An item $i$ has a certain size $s_i$. The objective is to pack all items into the boxes such that the maximum size packed into a box is minimized. More precisely, we aim at minimizing the size of the most filled box.

**Problem 2** (**Identical-machines scheduling** (**IMS**))**.**

$$
\begin{aligned}
\textbf{Input:}\quad & \text{Items } \mathcal{I} = \{1, \ldots, n\}, \\
& \text{Boxes } \mathcal{B} = \{1, \ldots, m\} \\
& \text{Sizes } \{s_i\}_{i \in \mathcal{I}} \\
\textbf{Objective:}\quad & \text{minimize } H && && (6)\\
\textbf{subject to:}\quad & \sum_{j \in \mathcal{B}} x_{ij} = 1 && \forall i \in \mathcal{I} && (7)\\
& H_j = \sum_{i \in \mathcal{B}} p_i \cdot x_{ij} && j \in \mathcal{B} && (8)\\
& H \geq H_j && j \in \mathcal{B} && (9)\\
& x_{ij} \in \{0, 1\} && i \in \mathcal{I}, j \in \mathcal{B} && (10)
\end{aligned}
$$

Observe that the minimization function of **Multi_Bot** and **IMS** are similar: in both problems, we aim at minimizing a certain "volume" of the jobs/items which have been put into the most filled box/period. Naturally, we will try in the remainder to reduce - in some sense - instances of **Multi_Bot** into instances of the well-known problem **IMS**. There is a natural correspondence between packings and instances of **IMS**, since each $p$-resource perfoming a job $k$ in $\boldsymbol{\pi}$ can be seen as an item of size $p$. Said differently, for any $p \in \mathcal{P}$, the $\sum_k \pi_{pk}$ resources present in packing $\boldsymbol{\pi}$ can be converted into $\sum_k \pi_{pk}$ items of size $p$.

Given an **IMS** instance $\mathcal{J}$, we define:

- its *volume* $\text{vol}(\mathcal{J})$ as the total size of its items, *i.e.* $\text{vol}(\mathcal{J}) = \sum_{i \in \mathcal{I}} s_i$,
- its *maximum* as the maximum item size: $\max(\mathcal{J}) = \max_{i \in \mathcal{I}} s_i$.

A heuristic of **IMS** will be used as a sub-routine for our approximation algorithm dedicated to **Multi_Bot**. It is called Longest-processing-time-first (Lpt-first) [10]. Its description is relatively simple: first sort the items

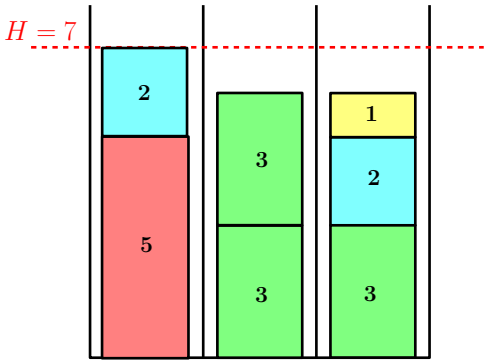


Figure 2: Output of LPT-FIRST with item sizes $(5, 3, 3, 3, 2, 2, 1)$

by decreasing order of their sizes (the largest size comes first), second put the items into the boxes following this order. The filling must satisfy the following rule: we always fill the box with the largest empty space, or said differently the least filled box. Figure 2 shows the packing obtained with LPT-FIRST for item sizes $(5, 3, 3, 3, 2, 2, 1)$ with three boxes. Graham showed that LPT-FIRST is a $\frac{4}{3}$-approximation algorithm [10]. In addition to offering a small approximation ratio, LPT-FIRST is a fast algorithm: it runs only in $O(n(\log m + \log n))$.

In the literature, **IMS** admits other approximation algorithms, such as MULTIFIT [8] which has a ratio $\frac{13}{11}$, proven by Yue [18]. Hochbaum and Schmoys proposed a PTAS [12] and we know that **IMS** does not admit a FPTAS since it is strongly NP-complete [9]. In the remainder, despite the existence of smaller approximation ratios, we focus only on the LPT-FIRST algorithm since it allows us to identify an approximation of **Multi_Bot** with our framework.

### 3.3 Optimal configurations

We provide a crucial observation for **Multi_Bot** which will help us designing efficient algorithms in the remainder. This is based on the notion of *optimal configuration* described below.

**Definition 4** (Optimal configuration for $k \in \mathcal{K}$)**.** *For any $k \in \mathcal{K}$, let $p_0(k)$ be the* optimal configuration *for jobs of type $k$,* i.e. *the configuration $p \in \mathcal{P}$ which is the most efficient in terms of resources. In brief,*

$$p_0(k) = \underset{p \in \mathcal{P}}{\operatorname{argmax}} \frac{c_{pk}}{p}$$

For any optimum solution $\mathbf{x}^*$ of **Multi_Bot**, given some type $k$ of job, the number $x^*_{p_0(k),k,t}$ of $p_0(k)$-resources used at each period $t$ is potentially very large (exponential in the input size). However, we know that the number of $p$-resources used, with $p \neq p_0(k)$, is bounded polynomially by the input size.

**Lemma 1.** *There is an optimum solution* $\mathbf{x}^*$ *for which, for any* $k \in \mathcal{K}$*, and* $p \neq p_0(k)$*,* $x^*_{pkt} \leq p_0(k) \leq P$*.*

*Proof.* Consider an arbitrary optimum schedule $\mathbf{x}^*$ and assume that some $x^*_{pkt}$, with $k \in \mathcal{K}$ and $p \neq p_0(k)$, is larger than $p_0(k)$. Then, we build a schedule $\mathbf{x}'$ with $H' = \max_t \sum_{p,k} p x'_{pkt}$ at most $H^*$ and which satisfy all demands. We fix $q = \lfloor \frac{x^*_{pkt}}{p_0(k)} \rfloor \geq 1$. Let $\mathbf{x}'$ be the same schedule than $\mathbf{x}^*$ but replace, at period $t$, a number $p_0(k)q$ of $p$-resources processing jobs of type $k$ by a number $pq$ of $p_0(k)$-resources processing also jobs of type $k$. Observe that this transformation does not modify the volume taken for period $t$. Moreover, the production is only modified for jobs of type $k$ and, as we used a more efficient configuration in the new schedule, the production of jobs of type $k$ at period $t$ cannot decrease. Formally,

$$\sum_{p'=1}^{P} c_{p'k} x'_{p'k} = \sum_{p'=1}^{P} c_{p'k} x^*_{p'k} + q\left(p c_{p_0(k),k} - p_0(k) c_{pk}\right) = \sum_{p'=1}^{P} c_{p'k} x^*_{p'k} + R$$

By Definition 4, we know that $c_{p_0(k),k} \geq p_0(k)\frac{c_{pk}}{p}$, so $R \geq 0$. In summary, schedule $\mathbf{x}'$ ensures at least the same production than $\mathbf{x}^*$, uses the same volume per period (same $H^*$). Therefore, it is also an optimum schedule and it satisfies $x'_{pkt} \leq p_0(k)$. □

From now on, any optimum solution $\mathbf{x}^*$ of **Multi_Bot** will be supposed to be such as the one described by Lemma 1.

# 4 Structure and analysis of the approximation algorithm

In this section, we present the shape of our approximation algorithm. This will consist in two steps: first solving a slightly different problem from **Multi_Bot** with only one period, second use its solution to propose a global schedule for the $T$ periods. We show how the approximation ratio $\frac{4}{3}$ can be obtained for the general **Multi_Bot** when one-period problems are solved exactly.

## 4.1 Presentation of the algorithm

Our idea to design approximation algorithms for the general **Multi_Bot** follows. We call this general framework Bot-approx:

- **Step 1.** Compute a polynomial-sized collection $\Pi$ of packings with structural properties (see Theorem 1 for details),
- **Step 2.** For each $\boldsymbol{\pi} \in \Pi$, create an **IMS** instance $\mathcal{I}(\boldsymbol{\pi})$ with $T$ boxes and items which are directly obtained from the packing $\boldsymbol{\pi}$ (transformation $\boldsymbol{\pi} \rightarrow I(\boldsymbol{\pi})$ is described in Section 4.2),

- **Step 3.** Find an approximate solution for all **IMS** instances $\mathcal{I}(\boldsymbol{\pi})$ with LPT-FIRST. Return the one which corresponds to the best schedule.

Step 1 is very fuzzy for now, and we will introduce in Section 5 our method for computing this collection of packings. Our objective is to produce a collection $\Pi$ containing at least one packing $\boldsymbol{\pi} \in \Pi$ which is a good candidate for achieving a satisfying schedule $\mathbf{x}$ when we put its $p$-resources into the periods. Indeed, at least one packing $\boldsymbol{\pi} \in \Pi$ will allow us to return after Steps 2 and 3 a schedule $\mathbf{x}$ with $H \leq \frac{4}{3}H^*$. The properties of this collection $\Pi$ are presented in Theorem 1. In particular, all packings of $\Pi$ will satisfy the demands $d_k$ for each type of job $k$, *i.e.* $\sum_{p\in\mathcal{P}} c_{pk}\pi_{pk} \geq d_k$.

**Theorem 1.** *Given some* ***Multi_Bot*** *instance, we can produce in polynomial time* $O(KP^3T^3 + KP^7)$ *a collection* $\Pi$ *of at most* $3P$ *packings such that:*

- *each packing in* $\Pi$ *satisfy all demands,*
- *it contains at least one packing* $\boldsymbol{\pi}$ *which satisfies* $\text{vol}(\boldsymbol{\pi}) \leq \text{vol}(\boldsymbol{\pi}^*)$*,*
- *if* $H^* < 3P$*, it contains at least one packing* $\boldsymbol{\pi'}$ *which satisfies* $\text{vol}(\boldsymbol{\pi'}) \leq \text{vol}(\boldsymbol{\pi}^*)$*,* $\max(\boldsymbol{\pi'}) \leq H^*$*, and* $H^*$-$\text{scale}(\boldsymbol{\pi'}) \leq T$*.*

*Proof.* Section 5 is entirely dedicated to the proof of this result. Its conclusion is given in Section 5.4. □

In the remainder of this algorithm, we apply Steps 2 and 3 for any packing in $\Pi$. Eventually, as the size of the collection is at most $3P$, we will obtain a set of at most $3P$ schedules. We will simply keep the one which provides the minimum $H$.

Step 2 will be detailed in Section 4.2. For some packing $\boldsymbol{\pi} \in \Pi$, a natural idea is, for each value $\pi_{pk}$, $p \in \mathcal{P}$, $k \in \mathcal{K}$, to create $\pi_{pk}$ items of size $p$ and solve **IMS** with $T$ boxes. Said differently, we can construct an instance, with exactly $\sum_k \pi_{pk}$ items of size $p$ for each $p \in \mathcal{P}$, which will be equivalent to the packing $\boldsymbol{\pi}$. In this way, a solution of this **IMS** instance with $T$ boxes correspond to a schedule for the initial **Multi_Bot** instance.

Unfortunately, this transformation might not be achieved in polynomial time as values $\pi_{pk}$, which depend on demands $d_k$, can be exponential in the input size of **Multi_Bot**. In other words, we would create an **IMS** instance of exponential size. Hence, we present a polynomial-time method which allows us to handle this issue and produce a polynomial-sized **IMS** instance for $\boldsymbol{\pi}$, denoted by $\mathcal{I}(\boldsymbol{\pi})$.

Eventually, Step 3 consists in applying LPT-FIRST on each **IMS** instance $\mathcal{I}(\boldsymbol{\pi})$ - created at Step 2 - with $T$ boxes. The solutions obtained thus correspond to schedules, as the $T$ boxes of **IMS** represent the periods of **Multi_Bot** and each of these periods contains a set of performed jobs (represented by the items), characterized by a configuration $p \in \mathcal{P}$ and some type of job $k \in \mathcal{K}$. Consequently, the output of the whole process is a collection of $|\Pi| \leq 3P$ schedules $\mathbf{x}$. Naturally, we keep the schedule with the minimum $H$. We will show that it offers a $\frac{4}{3}$-approximation for **Multi_Bot** (see Theorems 3 and 4).

Based on the Theorems 1, 3 and 4 cited above and proved in the remainder of the article, we present the main result of this paper.

**Theorem 2** (Approximation ratio of BOT-APPROX)**.** BOT-APPROX *is a* $\frac{4}{3}$*-approximation algorithm for* **Multi_Bot**.

*Proof.* Consider some instance of **Multi_Bot**. We distinguish two cases, depending on the value of $H^*$.

If $H^* \geq 3P$, then we know that the collection $\Pi$ computed at Step 1 contains a packing $\boldsymbol{\pi}$ with a smaller volume than $\boldsymbol{\pi}^*$ (Theorem 1). According to Theorem 3, the schedule $\mathbf{x}$ produced with LPT-FIRST by considering this initial packing $\boldsymbol{\pi}$ offers the guarantee that $H \leq \frac{4}{3}H^*$.

If $H^* < 3P$, then, again from Theorem 1, collection $\Pi$ contains a packing $\boldsymbol{\pi}'$ with a smaller volume than $\boldsymbol{\pi}^*$ such that $\max(\boldsymbol{\pi}') \leq H^*$ and $H^*$-scale$(\boldsymbol{\pi}') \leq T$. By Theorem 4, the schedule $\mathbf{x}'$ produced with LPT-FIRST by considering this initial packing $\boldsymbol{\pi}'$ gives also $H' \leq \frac{4}{3}H^*$.

As BOT-APPROX returns the schedule minimizing $H$ among all initial packings $\boldsymbol{\pi} \in \Pi$, we are sure to obtain a final solution which uses at most $\frac{4}{3}H^*$ resources. □

From now on, in the next sections, our objective is to prove Theorem 1 (Section 5), but also Theorems 3 and 4 (Section 4.3) which are the keystones for the proof of Theorem 2.

## 4.2 Transformation of a packing into polynomial IMS

We assume now that we are working with some given packing $\boldsymbol{\pi}$, which belongs to the collection computed with Theorem 1 and is a good candidate to obtain an approximate solution for the **Multi_Bot** instance. The objective is to "schedule" this packing into $T$ periods. Packing $\boldsymbol{\pi}$ satisfies all demands $d_k$ of the **Multi_Bot** instance. In order to assign efficiently each performed job of $\boldsymbol{\pi}$ into the periods, we model it as an **IMS** instance and then use LPT-FIRST to ensure the approximation factor. We focus in this subsection on the transformation from packing $\boldsymbol{\pi}$ to an **IMS** instance $I(\boldsymbol{\pi})$.

A natural way to achieve such equivalent transformation is simply, for each pair $k \in \mathcal{K}, p \in \mathcal{P}$, to create $\pi_{pk}$ items of size $p$. In this way, we obtain an **IMS** instance with a volume (*i.e.* total size of the items) equal to vol$(\boldsymbol{\pi})$. Each item of $I(\boldsymbol{\pi})$ thus represents a performed job and its size gives us the number $p$ of resources it involves. Furthermore, we keep in memory, for each item (equivalently for each $p$-resource of packing $\boldsymbol{\pi}$), which type of jobs this $p$-resource performs, even if it has no impact on the instance $I(\boldsymbol{\pi})$. Hence, assigning these items to a period $t \in \mathcal{T}$ produces a solution of **IMS** which completely corresponds to a schedule, since it is equivalent to assigning $p$-resources performing jobs of type $k$ to period $t$, *i.e.* proposing some vector $(x_{pkt})_{p,k,t}$.

Unfortunately, this simple transformation $\boldsymbol{\pi} \rightarrow I(\boldsymbol{\pi})$ can produce an exponential-sized instance. As mentioned in the previous subsection, the values $\pi_{pk}$ might be exponential in the input size of **Multi_Bot**, as they depend on capacities

and demands which are numerical values. To avoid an **IMS** instance of exponential size, our idea consists in forming "large" items which will represent a set of $p$-resources, instead of a single one. In fact, we will distinguish two cases. When $\text{vol}(\boldsymbol{\pi})$ is upper-bounded by some polynomial function of $P$ and $T$ given below, we use the natural process of transforming each value $\pi_{pk}$ into $\pi_{pk}$ items of size $p$. Otherwise, each item will correspond to a set of $p$-resources and the polynomial size of the constructed **IMS** instance will be guaranteed. The formal definition follows.

**Definition 5.** *We define $I$ as a polynomial-time algorithm which given some packing $\boldsymbol{\pi} = (\pi_{pk})_{p,k}$, produces an **IMS** instance with the following rules:*

- *If $\text{vol}(\boldsymbol{\pi}) \leq 3PT$, for each pair $p \in \mathcal{P}, k \in \mathcal{K}$, add $\pi_{pk}$ items of size $p$ into instance $I(\boldsymbol{\pi})$.*
- *Otherwise, if $\text{vol}(\boldsymbol{\pi}) > 3PT$, items will represent a set of at most $\frac{\text{vol}(\boldsymbol{\pi})}{3T}$ performed jobs with the same configuration $p$ and performing the same type $k$ of jobs. Analytically, for each $p$, let $\alpha_p = \lfloor \frac{\text{vol}(\boldsymbol{\pi})}{3pT} \rfloor$. For each pair $p \in \mathcal{P}, k \in \mathcal{K}$, add $\lfloor \frac{\pi_{pk}}{\alpha_p} \rfloor$ items of size $p\alpha_p$, and 1 item of size $p(\pi_{pk} - \lfloor \frac{\pi_{pk}}{\alpha_p} \rfloor \alpha_p)$*

The first case, $\text{vol}(\boldsymbol{\pi}) \leq 3PT$, corresponds to the natural transformation described above, so we do not give more details on it. However, the second one, $\text{vol}(\boldsymbol{\pi}) > 3PT$, needs extra explanations. Here, an item represents a "block" of several $p$-resources performing jobs of types $k$, in order to ensure that $I(\boldsymbol{\pi})$ has a polynomial size in the encoding of the initial **Multi_Bot** instance. Each item is thus associated with a configuration $p$ and a type of job $k$. Considering some pair $(p, k)$, almost all items associated with it (except one) represent a number $\alpha_p$ of $p$-resources, so their size is $p\alpha_p$. Their number is given by the division of $\pi_{pk}$ (number of $p$-resources performing jobs of type $k$ in packing $\boldsymbol{\pi}$) by the number $\alpha_p$ of $p$-resources represented by each block. But, if $\pi_{pk}$ is not a multiple of $\alpha_p$, an extra item should be added into $I(\boldsymbol{\pi})$ to represent the remaining $p$-resources.

Observe that, in both cases, the total volume of the **IMS** instance $I(\boldsymbol{\pi})$ is equal to the volume of packing $\boldsymbol{\pi}$ as we represented all the performed jobs into $I(\boldsymbol{\pi})$. But the crucial property ensured by transformation $I$ is certainly that the returned **IMS** instance has a size polynomial in $P, K, T$.

**Lemma 2.** *Let $\boldsymbol{\pi}$ be some packing. The **IMS** instance $I(\boldsymbol{\pi})$:*

1. *contains $O(PKT)$ items,*
2. *satisfies $\text{vol}(I(\boldsymbol{\pi})) = \text{vol}(\boldsymbol{\pi})$*
3. *does not contain items of size greater than $\frac{H^*}{3}$ if $\text{vol}(\boldsymbol{\pi}) > 3PT$ and $\text{vol}(\boldsymbol{\pi}) \leq \text{vol}(\boldsymbol{\pi}^*)$.*

*Proof.* 1. If $\text{vol}(\boldsymbol{\pi}) \leq 3PT$, then the number of items is exactly $\sum_{p,k} \pi_{pk}$ which is smaller than $\sum_{p,k} p \cdot \pi_{pk} = \text{vol}(\boldsymbol{\pi}) \leq 3PT$. Else, we know that for each pair $(p,k)$, there are at most $\frac{\pi_{pk}}{\alpha_p} + 1$ items. Let $\mu_p = \frac{\text{vol}(\boldsymbol{\pi})}{3pT}$. As $\mu_p > 1$, $\alpha_p = \lfloor \mu_p \rfloor \geq \frac{\mu_p}{2}$. Hence,

$$\frac{\pi_{pk}}{\alpha_p} = \frac{\pi_{pk}}{\lfloor \mu_p \rfloor} \leq \frac{2\pi_{pk}}{\mu_p} = 6T \frac{p \cdot \pi_{pk}}{\text{vol}(\boldsymbol{\pi})} \leq 6T.$$

Consequently, the total number of items in $I(\boldsymbol{\pi})$ is at most $PK(6T+1)$.

2. The volume of an **IMS** instance is the sum of the sizes over all its items. If $\text{vol}(\boldsymbol{\pi}) \leq 3PT$, then $\text{vol}(I(\boldsymbol{\pi})) = \sum_{p,k} p \cdot \pi_{pk} = \text{vol}(\boldsymbol{\pi})$. Otherwise,

$$\text{vol}(I(\boldsymbol{\pi})) = \sum_{p,k} \left( p\alpha_p \lfloor \frac{\pi_{pk}}{\alpha_p} \rfloor + p(\pi_{pk} - \lfloor \frac{\pi_{pk}}{\alpha_p} \rfloor \alpha_p) \right) = \sum_{p,k} p \cdot \pi_{pk} = \text{vol}(\boldsymbol{\pi}).$$

3. If $\text{vol}(\boldsymbol{\pi}) > 3PT$, then $p\alpha_p = p\lfloor \frac{\text{vol}(\boldsymbol{\pi})}{3pT} \rfloor \leq \frac{\text{vol}(\boldsymbol{\pi})}{3T}$: as the volume of $\boldsymbol{\pi}$ is at most the one of $\boldsymbol{\pi}^*$ and $H^* \geq \frac{\text{vol}(\boldsymbol{\pi}^*)}{T}$, it gives $p\alpha_p \leq \frac{H^*}{3}$. □

## 4.3 Approximation analysis

The objective of this subsection is to show that, given the **IMS** instances $\mathcal{I}(\boldsymbol{\pi})$ we created with Steps 1 and 2, LPT-FIRST algorithm provides a $\frac{4}{3}$-approximation for **Multi_Bot** on at least one of these instances. Indeed, as each $\mathcal{I}(\boldsymbol{\pi})$ contains $T$ boxes, which can be seen as the periods of **Multi_Bot**, any of its solution can be directely converted into a schedule $(x_{pkt})_{p,k,t}$.

We begin with the proof of Theorem 3. Given some packing $\boldsymbol{\pi}$ with a smaller volume than $\boldsymbol{\pi}^*$ and which satisfy all demands, LPT-FIRST achieves a $\frac{4}{3}$-approximation ratio under the condition: $H^* \geq 3P$.

**Theorem 3.** *Consider some* **Multi_Bot** *instance and a packing* $\boldsymbol{\pi}$ *which satisfies all demands. Moreover,* $\text{vol}(\boldsymbol{\pi}) \leq \text{vol}(\boldsymbol{\pi}^*)$. *If* $H^* \geq 3P$, *then the schedule* $\mathbf{x}$ *returned by solving* $\mathcal{I}(\boldsymbol{\pi})$ *with* LPT-FIRST *satisfies* $H \leq \frac{4}{3}H^*$.

*Proof.* Let $N$ be the number of items in $I(\boldsymbol{\pi})$. We know that $N = O(PKT)$ from Lemma 2. We proceed by induction on the number $z$ of items packed by LPT-FIRST. We prove that, for any $0 \leq z \leq N$, the number of resources present in each period (or box) after packing the $z$ most large items is at most $\frac{4}{3}H^*$.

The base case is trivial: when $z = 0$, no item was packed, so the current $H$ is zero. Now, assume we already packed the $z$ first items and we want to pack item $z+1$. According to Lemma 2, $\text{vol}(I(\boldsymbol{\pi})) = \text{vol}(\boldsymbol{\pi}) \leq \text{vol}(\boldsymbol{\pi}^*)$, therefore there is necessarily a period which contains less than $H^*$ resources, otherwise the total volume would overpass $TH^*$, a contradiction. So, the least filled period contains at most $H^*$ resources. Refering to the transformation $I$, see Definition 5, if $\text{vol}(\boldsymbol{\pi}) \leq 3PT$, each item represents a $p$-resource, so we pack item $z+1$ of size at most $P \leq \frac{H^*}{3}$ into it. The volume of this period after adding item $z+1$ is at most $\frac{4}{3}H^*$. If $\text{vol}(\boldsymbol{\pi}) > 3PT$, we also pack some item

of size at most $p\alpha_p \leq \frac{H^*}{3}$ according to Lemma 2. Together with the induction hypothesis, this observation shows us that all periods use at most $\frac{4}{3}H^*$ resources after packing $z+1$ items: $H \leq \frac{4}{3}H^*$. □

Now we fix the other side of the tradeoff, *i.e.* when $H^* < 3P$.

**Theorem 4.** *Consider some* ***Multi_Bot*** *instance with $\lambda = H^*$ and a packing $\boldsymbol{\pi}$ which satisfy all demands. Moreover, $\mathrm{vol}(\boldsymbol{\pi}) \leq \mathrm{vol}(\boldsymbol{\pi}^*)$, $\max(\boldsymbol{\pi}) \leq \lambda$ and $\lambda$-scale$(\boldsymbol{\pi}) \leq T$. If $H^* < 3P$, the schedule $\mathbf{x}$ returned by solving $\mathcal{I}(\boldsymbol{\pi})$ with* LPT-FIRST *satisfies $H \leq \frac{4}{3}H^*$.*

*Proof.* As $H^* < 3P$, $\mathrm{vol}(\boldsymbol{\pi}) \leq \mathrm{vol}(\boldsymbol{\pi}^*) \leq TH^* < 3PT$, hence the transformation $I$ simply consists, for each pair $(p,k)$, in adding $\pi_{pk}$ items of size $p$. As a consequence, each item represents exactly one $p$-resource of packing $\boldsymbol{\pi}$.

LPT-FIRST packs first the items of sizes $p > \frac{2H^*}{3}$. We look at the packing after adding only these big items. As $\lambda = H^*$ and $\lambda$-scale$(\boldsymbol{\pi}) \leq T$, then there are at most $T$ big items. Moreover, all big items ($p > \frac{2H^*}{3}$) do not overpass a size $H^*$, as $\max(\boldsymbol{\pi}) \leq \lambda$. So, at this moment, at most 1 item is present in each period, their size is at most $H^*$ resources, and the periods which are not filled with one big item are empty.

Second, LPT-FIRST packs the medium items of sizes $p > \frac{H^*}{3}$ which are not big. The empty boxes are filled and, when none of them is empty anymore, the remaining medium items are packed upon another medium item, as their boxes are less filled than the ones with big items. There cannot be three medium items into one period $t$ and only one medium item into another period $t'$ since the volume of two medium items is at least $\frac{2H^*}{3}$ and is necessarily larger than for exactly one medium item. Consequently, if there is a period with at least three medium items, then all other periods containing medium items are made up of two of them at least. However, having such a period with three medium items contradicts the scale criterion as we would have $\lambda$-scale$(\boldsymbol{\pi}) > T$. Consequently, there cannot be more than two medium items per period, and the number of resources present in the periods filled with medium items is at most $\frac{4H^*}{3}$.

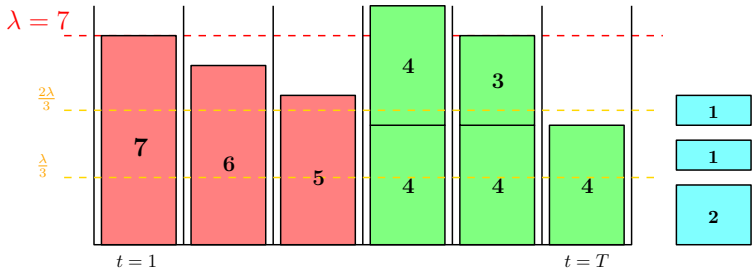


Figure 3: The result of LPT-FIRST after filling periods with big and medium items first. Items of sizes 1 and 2 ($p \leq \frac{\lambda}{3}$) have still to be treated.

Figure 3 illustrates an example of item assignment with LPT-FIRST after considering big (drawn in red) and medium (in green) items only. We fix $\lambda = 7$,

$T = 6$, and items sizes $(7, 6, 5, 4, 4, 4, 4, 3, 2, 1, 1)$. The small items, in blue, are not put inside the boxes yet.

The end of the proof consists in applying again the argument used in the proof of Theorem 3. Now, all periods contain at most $\frac{4H^*}{3}$ resources: the periods with one big item do not exceed $H^*$ and the periods with at most two medium items do not exceed $\frac{4H^*}{3}$. The remaining items have size at most $\frac{H^*}{3}$, and $\text{vol}(\boldsymbol{\pi}) \leq \text{vol}(\boldsymbol{\pi}^*)$. So, at any step, the least filled period will contain at most $H^*$ resources in order to satisfy the volume condition: we can pack a small item into it without exceeding the bound $\frac{4H^*}{3}$. □

# 5 Generation of packings with small volume

The objective of this section is to generate suitable packings for approximating **Multi_Bot**. More precisely, we aim at fixing the process of Step 1 of our algorithm Bot-approx. As stated in Theorem 1, we want to produce in polynomial-time a collection $\Pi$ of packing with certain requirements. This section is dedicated to the proof of Theorem 1. The conclusion of this proof is given in Section 5.4.

We focus on the specific formulation of **Multi_Bot** where $T = 1$. The objective of this problem is to determine the minimum number of resources needed to process all jobs - at least $d_k$ jobs of type $k$ for all $k \in \mathcal{K}$ - in one single period, without reconfiguration. We denote it by **Multi_Bot_1P**. Its mathematical formulation can be obtained by simply replacing $T = 1$ into the one given in Problem 1. A solution is thus a packing $(\pi_{pk})_{p,k}$ as the parameter $t$ does not intervene anymore here. The goal is to minimize the total number of resources, *i.e.* $\sum_{k\in\mathcal{K}} \sum_{p\in\mathcal{P}} p{\cdot}\pi_{pk}$, while all demands are satisfied, *i.e.* $\sum_{p\in\mathcal{P}} c_{pk}{\cdot}\pi_{pk} \geq d_k$ for any $k \in \mathcal{K}$.

Concretely, solving **Multi_Bot_1P** provides us with the optimal way to process all jobs during only one session. From Section 2.1, we know that the input size of **Multi_Bot_1P** is $O(KP(\log c_{\max} + \log d_{\max}))$. We will see in the remainder that an optimal packing for **Multi_Bot_1P** can be found in polynomial time.

## 5.1 Optimal packings with limited volume and scale

To deal with the requirements of Theorem 1, we define a problem which is more general than **Multi_Bot_1P**. We call it **Multi_Bot_1P**$[\lambda, \tau]$: it makes two extra parameters $\lambda, \tau \in \mathbb{N}$ intervene, see Problem 3.

We describe the non-trivial constraints presented in Problem 3. Constraint (11) means that we must have a sufficient capacity to satisfy all demands $d_k$. Constraint (12) means that the volume of the packing must be at most $\lambda T$. Constraint (13) means that a performed job must contain no more than $\lambda$ resources. Finally, constraint (14) implies that the $\lambda$-scale of the resulting packing must be less than $\tau$. Without constraint (12), the problem would always admit

a solution, but its volume $H$ could be as large as possible. Here, we are only interested in solutions with a volume bounded by some polynomial function.

Observe that if $\lambda \to +\infty$, constraints (12), (13) and (14) disappear and the problem "tends to" **Multi_Bot_1P**, which always admit a solution. In the remainder, we abuse notation and denote by $\lambda = \infty$ this case. Our idea consists in solving **Multi_Bot_1P**$[\lambda, \tau]$ for $\tau = T$ and all values $\lambda \in \{1, 2, 3, \ldots, 3P-1, \infty\}$: if we find a solution, it will be put into collection $\Pi$.

**Problem 3** (**Multi_Bot_1P**$[\lambda, \tau]$)**.**

**Input:** $\mathcal{K} = \{1, \ldots, K\}, \mathcal{P} = \{1, \ldots, P\}$
Capacities $(c_{pk})_{\substack{p\in\mathcal{P}\\k\in\mathcal{K}}}$
Demands $(d_k)_{k\in\mathcal{K}}$
$\lambda, \tau \in \mathbb{N}, 1 \leq \lambda \leq 3P$

**Objective:** find a packing $\boldsymbol{\pi} = (\pi_{pk})_{\substack{p\in\mathcal{P}\\k\in\mathcal{K}}}$ which
minimizes $H$ **or** state "no solution exists"

**subject to:**

$$\sum_{p\in\mathcal{P}} c_{pk} \cdot \pi_{pk} \geq d_k \qquad \forall k \in \mathcal{K} \quad (11)$$

$$\sum_{k\in\mathcal{K}} \sum_{p\in\mathcal{P}} p \cdot \pi_{pk} = H \leq \lambda\tau \qquad (12)$$

$$\pi_{pk} = 0 \qquad \forall k \in \mathcal{K}, p > \lambda \quad (13)$$

$$\sum_{k\in\mathcal{K}} \sum_{p>\frac{2\lambda}{3}} \pi_{pk} + \frac{1}{2} \sum_{k\in\mathcal{K}} \sum_{\frac{\lambda}{3}<p\leq\frac{2\lambda}{3}} \pi_{pk} \leq \tau \qquad (14)$$

$$\pi_{pk}, H \in \mathbb{N}, \qquad \forall k \in \mathcal{K}, p \in \mathcal{P} \quad (15)$$

Concretely, let $\boldsymbol{\pi}(\lambda)$ denote an optimum packing for **Multi_Bot_1P**$[\lambda, T]$ if it exists. We define:

$$\Pi = \{\boldsymbol{\pi}(\lambda) : 1 \leq \lambda \leq 3P-1 \text{ or } \lambda = \infty\} \qquad (16)$$

Collection $\Pi$ will contain at least 1 packing because $\boldsymbol{\pi}(\infty)$ necessarily exists, and at most $3P$ packings. We prove now that the optimum solutions for all these problems produce the expected collection of packings.

**Theorem 5.** *Packings $\boldsymbol{\pi}(\lambda)$ satisfy the following properties:*

1. *for any $\lambda$, $\boldsymbol{\pi}(\lambda)$, if it exists, satisfies all demands $d_k$,*
2. $\text{vol}(\boldsymbol{\pi}(\infty)) \leq \text{vol}(\boldsymbol{\pi}^*)$,
3. *if $H^* < 3P$, then $\boldsymbol{\pi}(H^*)$ exists and we have:* $\text{vol}(\boldsymbol{\pi}(H^*)) \leq \text{vol}(\boldsymbol{\pi}^*)$, $\max(\boldsymbol{\pi}(H^*)) \leq H^*$, $H^*$-$\text{scale}(\boldsymbol{\pi}(H^*)) \leq T$.

*Proof.* 1. The constraint (11) implies that any solution achieves all demands, independently from value $\lambda$.

2. Observe that the solutions (not necessarily optimal) of $\mathbf{Multi\_Bot\_1P}[\infty, T]$ correspond exactly to the set of packings satisfying all demands - constraint (11). Indeed, constraints (13) and (14) disappear when $\lambda = \infty$. In particular, $\boldsymbol{\pi}^*$ is one of these solutions and hence $\text{vol}(\boldsymbol{\pi}(\infty)) \leq \text{vol}(\boldsymbol{\pi}^*)$.

3. The reasoning is similar: if $H^* < 3P$, then $\boldsymbol{\pi}^*$ is a solution of $\mathbf{Multi\_Bot\_1P}[H^*, T]$. Obviously it satisfies all demands and its volume is at most $TH^*$. Furthermore, as each period in schedule $\mathbf{x}^*$ contains at most $H^*$ resources, *i.e.* $H_t^* = \sum_{p,k} x_{pkt}^* \leq H^*$, then necessarily $x_{pkt} = 0$ when $p > H^*$. So, $\pi_{pk} = 0$ for $p > H^*$ and any $k \in \mathcal{K}$. Finally, each period of schedule $\mathbf{x}^*$ cannot contain both a big configuration ($p > \frac{2H^*}{3}$) and a medium one ($\frac{H^*}{3} < p \leq \frac{2H^*}{3}$). It cannot contain either three medium ones as it overpasses capacity $H^*$. As a conclusion, each period contains at most either a big configuration or two medium ones: the $H^*$-scale of $\boldsymbol{\pi}^*$ is at most $T$. Hence, $\mathbf{Multi\_Bot\_1P}[H^*, T]$ admits at least one solution $\boldsymbol{\pi}^*$, so $\boldsymbol{\pi}(H^*)$ exists. As $\boldsymbol{\pi}(H^*)$ is the solution minimizing the volume, we have $\text{vol}(\boldsymbol{\pi}(H^*)) \leq \text{vol}(\boldsymbol{\pi}^*)$. The two other inequalities are the consequences of the definition of $\mathbf{Multi\_Bot\_1P}[H^*, T]$. □

The collection $\Pi$ meets the requirements of Theorem 1. Now, we prove that all these packings can be built in polynomial time.

## 5.2 Dynamic programming for the single-period problems

We begin with the generation of packings $\boldsymbol{\pi}(\lambda)$ for $1 \leq \lambda \leq 3P - 1$. We will deal with the case $\lambda = \infty$ in Section 5.3. We fix some positive integer $\lambda$ with $\lambda \leq 3P - 1$. Our objective is to solve $\mathbf{Multi\_Bot\_1P}[\lambda, T]$ and, hence, to obtain either some packing $\boldsymbol{\pi}(\lambda)$ or a negative answer. We present a dynamic programming (DP) procedure to achieve this task.

**Structure of DP memory**. From now on, variable $\tau$ represents a positive half-integer upper-bounded by $T$ and $W$ a positive integer representing the authorized volume, which is at most $\lambda T$.

We construct a four-dimensional vector `Table`, whose elements are $\texttt{Table}[p, k, W, \tau]$ with three integers $0 \leq p \leq \lambda$, $1 \leq k \leq K$, $0 \leq W \leq \lambda T$ and a half-integer $0 \leq \tau \leq T$. Hence, the total size of the vector is $\lambda^2 T^2 K = O(P^2 T^2 K)$. The role of this vector is to help us producing intermediary packings which allows us to obtain potentially the solution of $\mathbf{Multi\_Bot\_1P}[\lambda, T]$. Intuitively, index $k$ provides us with the types $1, \ldots, k$ of jobs we have to consider. Index $p$ is the maximum configuration which can be used to perform jobs of type $k$. Eventually, $W$ is a volume limit for the packing and $\tau$ is the $\lambda$-scale limit.

We denote by $\boldsymbol{\pi}[p, k, W, \tau]$ a packing which:

- produces only jobs of type $1, 2, \ldots, k$: $\pi_{pk'} = 0$ when $k' > k$,
- uses only configurations $1, 2, \ldots, p$ to process the jobs of type $k$: $\pi_{p'k} = 0$ when $p' > p$,

- satisfies the demands until $d_{k-1}$: $\sum_{p'=1}^{\lambda} c_{p'k'} \cdot \pi_{p'k'} \geq d_{k'}$ for all $1 \leq k' < k$,
- has a volume at most $W$: $\sum_{p'=1}^{\lambda} \sum_{k'=1}^{k} p' \cdot \pi_{p'k'} \leq W$,
- has a $\lambda$-scale at most $\tau$: $\lambda\text{-scale}(\boldsymbol{\pi}) \leq \tau$,
- maximizes the number of jobs of type $k$ processed, *i.e.* $\sum_{p'=1}^{\lambda} c_{p'k} \cdot \pi_{p'k}$.

Packing $\boldsymbol{\pi}[p, k, W, \tau]$ does not necessarily exist since reaching the demands might be avoided by the volume and scale conditions. Observe that, by definition, **Multi_Bot_1P**$[\lambda, T]$ admits a solution if and only if $\boldsymbol{\pi}[\lambda, K, \lambda T, T]$ exists and satisfies the demands for jobs of type $K$. Also, for the specific value $p = 0$, the packing $\boldsymbol{\pi}[0, k, W, \tau]$ cannot use any configuration for jobs of type $k$, so in fact it does not process jobs of type $k$ at all. We can fix: $\boldsymbol{\pi}[0, k, W, \tau] = \boldsymbol{\pi}[\lambda, k-1, W, \tau]$. The particular case $p = 0$ and $k = 1$ gives an empty packing $\boldsymbol{\pi}[0, 1, W, \tau]$: it will be forgotten in the remainder.

The objective of vector `Table` is to contain either the production of jobs of type $k$ by $\boldsymbol{\pi}[p, k, W, \tau]$ if it exists, or $-\infty$. More formally,

$$\texttt{Table}[p, k, W, \tau] = \begin{cases} \sum_{p'=1}^{p} c_{p'k}\pi_{p'k} & \text{if } \boldsymbol{\pi}[p, k, W, \tau] = (\pi_{p'k'})_{p',k'} \text{ exists} \\ -\infty & \text{otherwise} \end{cases} \tag{17}$$

In addition, we also compute two vectors `Bool` and `Pack` with the same dimension sizes than `Table`. Vector `Bool` simply indicates, with a boolean, whether $\boldsymbol{\pi}[p, k, W, \tau]$ satisfies the demand $d_k$ for jobs of type $k$ (`Bool`$[p, k, W, \tau] =$ True) or not (= False). Finally, vector `Pack` provides us with some information which allows us to retrieve exactly the composition of $\boldsymbol{\pi}[p, k, W, \tau]$. More details on vector `Pack` will follow.

Before stating our recursive formula to achieve the DP algorithm, we define an extra function $S_{p,\lambda}$ for any $1 \leq p \leq \lambda$. Given some half-integer "budget" $\tau$ and an integer $j$, it returns the updated $\lambda$-scale budget which remains after adding a number $j$ of $p$-resources to some packing. Formally,

$$S_{p,\lambda}(\tau, j) = \begin{cases} \tau & \text{if } p \leq \frac{\lambda}{3} \\ \tau - \frac{j}{2} & \text{if } \frac{\lambda}{3} < p \leq \frac{2\lambda}{3} \\ \tau - j & \text{if } p > \frac{2\lambda}{3} \end{cases}$$

**Recursive formula**. We state now a recursive formula for computing all values of `Table`, `Bool` and `Pack`. We begin with the recursive scheme of `Table` and `Bool`.

The base case is $p = k = 1$. Packing $\boldsymbol{\pi}[1, 1, W, \tau]$ has no demand to satisfy, therefore it necessarily exists. It corresponds to taking the maximum number of 1-resources which satisfy both the volume and scale conditions:

$$\forall W, \forall \tau, \quad \texttt{Table}[1, 1, W, \tau] = \max_{\substack{0 \leq j \leq W \\ S_{1,\lambda}(\tau, j) \geq 0}} j \cdot c_{11} \tag{18}$$

$$\texttt{Bool}[1, 1, W, \tau] = \text{True} \Leftrightarrow \texttt{Table}[1, 1, W, \tau] \geq d_1 \tag{19}$$

For the general statement, we distinguish two cases. First assume that $p = 0$ and $k > 1$. We have $\boldsymbol{\pi}[0, k, W, \tau] = \boldsymbol{\pi}[\lambda, k-1, W, \tau]$.

$$\texttt{Table}[0, k, W, \tau] = \begin{cases} 0 & \text{if } \texttt{Table}[\lambda, k-1, W, \tau] \geq d_{k-1} \\ -\infty & \text{otherwise} \end{cases} \tag{20}$$

$$\texttt{Bool}[0, k, W, \tau] = \text{False} \tag{21}$$

Second, assume that $p \geq 1$, *i.e.* jobs of type $k$ can be processed by $p'$-resources, with $p' \leq p$. We write:

$$\texttt{Table}[p, k, W, \tau] = \max_{\substack{0 \leq j \leq \lfloor \frac{W}{p} \rfloor \\ S_{p,\lambda}(\tau, j) \geq 0 \\ \texttt{Bool}[\lambda, k-1, W-jp, S_{p,\lambda}(\tau, j)]}} \texttt{Table}[p-1, k, W-jp, S_{p,\lambda}(\tau, j)] + jc_{pk} \tag{22}$$

$$\texttt{Bool}[p, k, W, \tau] = \text{True} \Leftrightarrow \texttt{Table}[p, k, W, \tau] \geq d_k \tag{23}$$

Index $j$ represents the number of $p$-resources we might add to our packing. Naturally, this addition should overpass neither the volume constraint ($j \leq \frac{W}{p}$), nor the scale one ($S_{p,\lambda}(\tau, j) \geq 0$). Furthermore, $\texttt{Bool}[\lambda, k-1, W-jp, S_{p,\lambda}(\tau, j)]$ must be True as it guarantees that demands $d_1, \ldots, d_{k-1}$ can be satisfied before the add-on of $j$ $p$-resources. It may happen that no index $j$ (even $j = 0$) satisfies these three conditions. In this case, we fix $\texttt{Table}[p, k, W, \tau] = -\infty$.

Observe that $\texttt{Table}[p, k, W, \tau] = -\infty$ if and only if $\texttt{Bool}[\lambda, k-1, W, \tau]$ is False. This makes sense since it means that with the volume $W$ and the scale $\tau$ we are considering, the demands for jobs of type $1, 2, \ldots, k-1$ could not be reached. Figure 4 provides us with a 2D-view of vector Table, highlighting the area where values are base cases and giving examples of recursive calls, following Equations (20) and (22). We prove that $\texttt{Table}[p, k, W, \tau]$ is equal to the expectations expressed in Equation (17).

**Lemma 3.** *If* $\boldsymbol{\pi}[p, k, W, \tau] = (\pi_{p'k'})_{p',k'}$ *exists, then:*

- $\texttt{Table}[p, k, W, \tau] = \sum_{p'} c_{p'k} \cdot \pi_{p'k}$.
- $\texttt{Bool}[p, k, W, \tau] = \text{True}$ *if and only if* $\boldsymbol{\pi}[p, k, W, \tau]$ *satisfies demand* $d_k$.

*Otherwise,* $\texttt{Table}[p, k, W, \tau] = -\infty$ *and* $\texttt{Bool}[p, k, W, \tau] = \text{False}$.

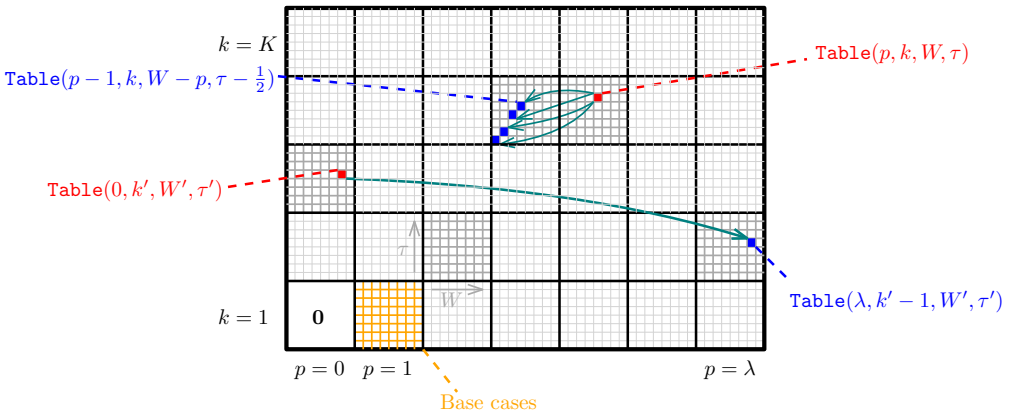


Figure 4: 2D-projection of the 4D-vector `Table` illustrating the different recursive calls

*Proof.* First, we handle the base case. If $p = k = 1$, then $\boldsymbol{\pi}[1, 1, W, \tau]$ exists and is the singleton packing $(\pi_{11})$ which processes the maximum number of jobs of type 1 with 1-resources while satisfying both the volume and scale constraints. Then, $\texttt{Table}[1, 1, W, \tau]$ must be equal to $\pi_{11}c_{11}$, which perfectly fits with the definition given in Equation (18). Moreover, according to Equation (19), $\texttt{Bool}[1, 1, W, \tau]$ is True if and only if $\boldsymbol{\pi}[1, 1, W, \tau]$ reaches demand $d_k = d_1$.

We proceed inductively. Assume that the statement is achieved not only for all pairs $(p', k')$ such that $k' < k$ but also when $k' = k'$ and $p' < p$.

If $\boldsymbol{\pi}[p, k, W, \tau]$ does not exist, then it means that demands $d_1, \ldots, d_{k-1}$ cannot be satisfied with volume $W$ and scale $\tau$ constraints. So, $\texttt{Table}[\lambda, k-1, W, \tau] < d_{k-1}$ and $\texttt{Bool}[\lambda, k-1, W, \tau]$ is False by induction. Consequently, Equations (20) and (22) ensure us that $\texttt{Table}[p, k, W, \tau] = -\infty$ and $\texttt{Bool}[p, k, W, \tau] =$ False, as expected. In the remainder of this proof, we assume $\boldsymbol{\pi}[p, k, W, \tau]$ exists.

If $p = 0$, then $\boldsymbol{\pi}[0, k, W, \tau] = \boldsymbol{\pi}[\lambda, k-1, W, \tau]$. As this packing does not process any job of type $k$, we fixed $\texttt{Table}[p, k, W, \tau] = 0$, see Equation (20). Hence, as stated in Equation (21), $\boldsymbol{\pi}[0, k, W, \tau]$ obviously does not satisfy demand $d_k > 0$.

If $p > 0$, then $\boldsymbol{\pi}[p, k, W, \tau]$ is made up of a certain number $j \geq 0$ of $p$-resources which process jobs of type $k$, and all other components which can be seen together as a slightly smaller packing. The latter packing must satisfy the demands for jobs of type $1, \ldots, k-1$ and also maximize the number of processed jobs of type $k$ with configurations $1, \ldots, p-1$, while fulfilling the volume and scale constraints, even if we know that already a number $j$ of $p$-resources have to be counted. Hence, $\boldsymbol{\pi}[p, k, W, \tau]$ can be obtained from $\boldsymbol{\pi}[p-1, k, W - jp, S_{p,\lambda}(\tau, j)]$ by adding only $\pi_{pk} = j$. This justifies Equation (22). Finally, as stated in Equation (23), $\texttt{Bool}[p, k, W, \tau]$ is True if and only if demand $d_k$ is satisfied by $\boldsymbol{\pi}[p, k, W, \tau]$, *i.e.* $\texttt{Table}[p, k, W, \tau] \geq d_k$. □

Values $\texttt{Table}[p, k, W, \tau]$ provide us with the number of jobs of type $k$ processed by optimum packings $\boldsymbol{\pi}[p, k, W, \tau]$. Nevertheless, our initial objective is to obtain the composition of these packings. To achieve it, we create a third table `Pack` with the same dimensions than `Table` and `Bool`. A first natural idea would be to fill each element $\texttt{Pack}[p, k, W, \tau]$ with the packing $\boldsymbol{\pi}[p, k, W, \tau]$. However, as the encoding size of $\boldsymbol{\pi}[p, k, W, \tau]$ is at least $PK$, it will have some relatively strong impact on the complexity of our algorithm. Hence, we fill $\texttt{Pack}[p, k, W, \tau]$ only with the necessary information needed to retrieve the packings.

If $\texttt{Table}[p, k, W, \tau] = -\infty$ or $p = 0$, then $\texttt{Pack}[p, k, W, \tau]$ is kept empty. Else, we define $\texttt{Pack}[p, k, W, \tau]$ as the integer $j$ which is the number of $p$-resources processing jobs of type $k$ in $\boldsymbol{\pi}[p, k, W, \tau]$.

$$\texttt{Pack}[p, k, W, \tau] = \begin{cases} j \text{ if } p = k = 1 \text{ and } \texttt{Table}[1, 1, W, \tau] = jc_{11} \\ j \text{ if } \texttt{Table}[p, k, W, \tau] = \texttt{Table}[p-1, k, W - jp, S_{p,\lambda}(\tau, j)] + jc_{pk} \\ \text{empty otherwise} \end{cases} \tag{24}$$

Observe that the encoding size of each value $\texttt{Pack}[p, k, W, \tau]$ is now $O(\log(\lambda T))$. Some packing $\boldsymbol{\pi}[p, k, W, \tau]$ can now be recovered with the following process, we call RECOVER:

- if $\texttt{Table}[p, k, W, \tau] = -\infty$, then $\boldsymbol{\pi}[p, k, W, \tau]$ does not exist.
- if $p = k = 1$ and $\texttt{Pack}[1, 1, W, \tau] = j$, then $\boldsymbol{\pi}[1, 1, W, \tau]$ is the singleton $(\pi_{11})$ with $\pi_{11} = j$.
- if $p = 0$ and $k > 1$, then $\boldsymbol{\pi}[0, k, W, \tau] = \boldsymbol{\pi}[\lambda, k-1, W, \tau]$.
- if $\texttt{Pack}[p, k, W, \tau] = j$, compute recursively packing $\boldsymbol{\pi}[p-1, k, W-jp, S_{p,\lambda}(\tau, j)]$ with RECOVER and add $\pi_{pk} = j$ to it.

We remind that if $\texttt{Table}[\lambda, K, \lambda T, T] \geq 0$, then $\boldsymbol{\pi}[\lambda, K, \lambda T, T]$ exists and it gives us some $\boldsymbol{\pi}(\lambda)$ if and only if the demand $d_K$ is achieved. Otherwise, if $\texttt{Table}[\lambda, K, \lambda T, T] = -\infty$, then there is no packing $\boldsymbol{\pi}(\lambda)$ for sure. As a consequence, the following algorithm solves **Multi_Bot_1P**$[\lambda, T]$: first compute simultaneously the three tables `Table`, `Bool` and `Pack`, second answer NO if $\texttt{Table}[\lambda, K, \lambda T, T] = -\infty$, otherwise retrieve packing $\boldsymbol{\pi}[\lambda, K, \lambda T, T]$ recursively with sub-routine RECOVER. If $\boldsymbol{\pi}[\lambda, K, \lambda T, T]$ satisfies demand $d_K$, then return it as $\boldsymbol{\pi}(\lambda)$, else answer NO. We call this algorithm BOT-PROGDYN (Algorithm 1).

**Theorem 6.** BOT-PROGDYN *solves* ***Multi_Bot_1P***$[\lambda, T]$ *with a running time* $O(\lambda^3 T^3 K)$.

*Proof.* From Lemma 3, we know that $\texttt{Table}[\lambda, K, \lambda T, T] \geq d_K$ if and only if $\boldsymbol{\pi}(\lambda)$ exists. Therefore, as BOT-PROGDYN achieves the dynamic programming to fills `Table`, it returns the solution of **Multi_Bot_1P**$[\lambda, T]$.

We focus now on the running time of BOT-PROGDYN. The dynamic programming routine uses a memory space of size $O(\lambda^2 T^2 K)$, and each computation

**Algorithm 1:** Exact algorithm BOT-PROGDYN which solves **Multi_Bot_1P**$[\lambda, T]$

**Input:** Instance of **Multi_Bot_1P**$[\lambda, T]$
**Output:** A packing $\boldsymbol{\pi}(\lambda)$ or answer NO
Initialize `Table`, `Bool` and `Pack` as empty vectors;
**for** *all* $1 \leq W \leq \lambda T$ *and* $1 \leq \tau \leq T$ **do**
    fix `Table`$[1, 1, W, \tau]$, `Bool`$[1, 1, W, \tau]$, `Pack`$[1, 1, W, \tau]$ with respectively Equations (18), (19), and (24);
**endfor**
$p = 1$;
**for** *all* $1 \leq k \leq K$ **do**
    **while** $p \leq \lambda$ **do**
        **for** *all* $1 \leq W \leq \lambda T$ *and* $1 \leq \tau \leq T$ **do**
            fix `Table`$[p, k, W, \tau]$, `Bool`$[p, k, W, \tau]$, `Pack`$[p, k, W, \tau]$ with Equations (20-24);
        **endfor**
        $p = p + 1$
    **end**
    $p = 0$;
**endfor**
$D =$ `Table`$[\lambda, K, \lambda T, T]$;
**if** $D \geq d_K$ **then return** RECOVER$(\lambda, K, \lambda T, T)$
**return** NO

uses at most $\lambda T$ recursive calls: the worst case occurs with Equation (22) and the computation of `Table`$[\lambda, K, \lambda T, T]$. Consequently, the number of comparisons/affectations/arithmetic operations is $O(\lambda^3 T^3 K)$.

Furthermore, the time needed to apply RECOVER is negligible compared to the latter. Indeed, the number of recursive calls needed to compute $\boldsymbol{\pi}(\lambda, K, \lambda T, T)$ is at most the size of vector `Pack`, which is $O(\lambda^2 T^2 K)$. □

### 5.3 Optimum packing with unlimited scale

We focus on problem **Multi_Bot_1P**, which is equivalent to **Multi_Bot_1P**$[\infty, T]$ ($T$ has no influence here). In other words, we aim at producing the packing $\boldsymbol{\pi}(\infty)$ which will be denoted $\boldsymbol{\pi}$ in this subsection to simplify notations.

Our reasoning is based on the result stated in Lemma 1. We will assume that the packing $\boldsymbol{\pi} = (\pi_{pk})_{p,k}$ admits low values for non-optimum configurations, *i.e.* $\pi_{pk} \leq p_0(k)$ for any $k \in \mathcal{K}$ and $p \neq p_0(k)$ (consequence of Lemma 1). This packing $\boldsymbol{\pi}$ can be decomposed into two parts: on one hand, the values $\pi_{p_0(k),k}$ for optimal configurations and, on the other hand, a packing $\boldsymbol{\pi'} = (\pi'_{pk})_{p,k}$ taking only values for non-optimum configurations. Formally,

$$\pi'_{pk} = \begin{cases} \pi_{pk} & \text{if } k \in \mathcal{K}, p \in \mathcal{P}, p \neq p_0(k) \\ 0 & \text{if } k \in \mathcal{K}, p = p_0(k) \end{cases}$$

This sub-packing, as expected, has a small volume. Let $W(P) = P\frac{P(P-1)}{2}$. We have $\text{vol}(\boldsymbol{\pi}') = \sum_{p,k} p \cdot \pi'_{pk} \leq \sum_k \sum_{p \neq p_0(k)} p \cdot p_0(k) \leq K \cdot W(P)$. More precisely, the volume of $\boldsymbol{\pi}'$ dedicated to each type of job is at most $W(P)$.

Our construction of packing $\boldsymbol{\pi}$ is inductive. For any $1 \leq k \leq K$, we denote by $\boldsymbol{\pi}_j = (\pi^{(j)}_{pk})_{p,k}$ a packing satisfying the following properties:

- $\boldsymbol{\pi}_j$ satisfies the demands for all types of jobs,
- for any $1 \leq i \leq j$, $\boldsymbol{\pi}_j$ uses the minimum number of resources to satisfy demand $d_i$ with all configurations allowed,
- for any $j < i \leq K$, uses the minimum number of resources to satisfy demand $d_i$ with only configuration $p_0(j)$ allowed.

First, observe that $\boldsymbol{\pi}_K$ is a solution of **Multi_Bot_1P** since it satisfies all demands while minimizing the total volume of the packing. Therefore, we can state $\boldsymbol{\pi} = \boldsymbol{\pi}_K$. Furthermore, $\boldsymbol{\pi}_0$ is a packing satisfying all demands which minimizes the total volume by using only optimal configurations for each type of jobs. It can be determined analytically and this will be the base case of our induction. For any $1 \leq k \leq K$,

$$\pi_{p_0(k),k} = \left\lceil \frac{d_k}{c_{p_0(k),k}} \right\rceil$$

By definition, all other components of packing $\boldsymbol{\pi}_0$ are zero. Below, we proceed the inductive step.

**Lemma 4.** *Assume packing $\boldsymbol{\pi}_j$ is known. One can build packing $\boldsymbol{\pi}_{j+1}$ in time $O(W(P)^2 P)$ using the algorithm* Bot-ProgDyn.

*Proof.* We remind that we are working on some instance $\mathcal{I}$ of **Multi_Bot_1P** = **Multi_Bot_1P**$[\infty, 0]$ (we remind that when $\lambda = \infty$, then the second parameter has no influence on the definition of the problem, so we put 0 arbitrarily). We define another instance $\mathcal{J}$ of **Multi_Bot_1P**$[\infty, 0]$. Instance $\mathcal{J}$ contains only one job type, which corresponds to the jobs of type $j+1$ in $\mathcal{I}$. Then, the capacities in $\mathcal{J}$ are exactly the capacities $c_{p(j+1)}$ of jobs of type $j+1$ in $\mathcal{I}$, except for the optimal configuration for which we fix $c_{p_0(j+1),j+1} = 0$. Concretely, $p_0(j+1)$ will not be used in the solutions of $\mathcal{J}$. We do not need to fix any demand constraint because, with only one job type, it will not intervene in algorithm Bot-ProgDyn.

Using Bot-ProgDyn on instance $\mathcal{J}$, we build vectors `Table` and `Pack` (with exactly one job type, `Bool` is not necessary) with limited volume $W(P)$. In brief, we stop the procedure when `Table`$[P, 1, W(P), 0]$ and `Pack`$[P, 1, W(P), 0]$ are obtained. Thanks to these vectors, we are able to obtain, for any volume

$0 \leq W \leq W(P)$, a packing which processes only jobs of type $j+1$, with volume at most $W$, and which maximizes the production.

By induction hypothesis, we know that packing $\boldsymbol{\pi}_j$ already minimizes the volume used to process the necessary demand for jobs of type $1, \ldots, j$. Thus, we focus only on jobs of type $j+1$. Moreover, the volume $W$ used to process jobs of type $j+1$ in packing $\boldsymbol{\pi}_{j+1}$ with non-optimal configurations does not exceed $W(P)$ (consequence of Lemma 1). Hence, we compute for all possible volumes $0 \leq W \leq W(P)$, the packing which maximizes the production of jobs of type $j+1$ while maintaining at most volume $W$. The production of such packing is given by $\texttt{Table}[P, 1, W, 0]$. It suffices to guess the volume $W$ taken by non-optimal configurations, to compute the packing maximizing the production of jobs of type $j+1$ with this volume and, eventually, to add optimal configurations $p_0(j+1)$ while the demand is not satisfied. At least one of these volumes $W$ will provide us with a packing $\boldsymbol{\pi}_{j+1}$ satisfying the properties stated above. Formally, we define:

$$W_{j+1} = \min_{0 \leq W \leq W(P)} W + \left\lceil \frac{d_{j+1} - \texttt{Table}[P, 1, W, 0]}{c_{p_0(j+1),(j+1)}} \right\rceil. \tag{25}$$

The right-hand side of Equation (25) is the total volume used for processing jobs of type $j+1$ when the demand has to be satisfied and also when exactly $W$ resources are assigned to non-optimum configurations.

Once the optimum volume $W_{j+1}$ for non-optimum configurations was determined, we fix, for the optimal configuration $p_0(j+1)$:

$$\pi^{(j+1)}_{p_0(j+1),j+1} = \left\lceil \frac{d_{j+1} - \texttt{Table}[P, 1, W_{j+1}, 0]}{c_{p_0(j+1),(j+1)}} \right\rceil$$

For $p \neq p_0(j+1)$, the number of $p$-resources assigned for the process of jobs of type $j+1$ is directly given by the packing which can be recovered from $\texttt{Pack}[P, 1, W_{j+1}, 0]$. □

**Theorem 7.** ***Multi_Bot_1P*** *can be solved in time* $O(KP^7)$.

*Proof.* We state the full inductive process here. First, we compute packing $\boldsymbol{\pi}_0$ with analytical formulas in time $O(KP)$. Then, we construct inductively all packings $\boldsymbol{\pi}_j$ for $1 \leq j \leq K$ thanks to Lemma 4. Eventually, packing $\boldsymbol{\pi}_K$ provides us with a solution of **Multi_Bot_1P**. As there are exactly $K$ inductive steps and each step runs in $O(P^7)$ according to Theorem 6, the total running time for computing $\boldsymbol{\pi}(\infty)$ is $O(KP^7)$. □

## 5.4 Computation of collection $\Pi$

Combining the results obtained in both Sections 5.2 and 5.3, we are now ready for the computation of the whole collection $\Pi$ of packings which will allow us to obtain a $\frac{4}{3}$-approximation for **Multi_Bot**.

*Proof of Theorem 1.* According to Theorem 6, each packing $\boldsymbol{\pi}(\lambda)$, $1 \leq \lambda \leq 3P-1$, can be produced in time $O(\lambda^3 T^3 K)$ (or the algorithm warns us that

such packing does not exist). Then, according to Theorem 7, packing $\boldsymbol{\pi}(\infty)$ can be computed in time $O(KP^7)$. Consequently, we build the collection described in Equation (16) with the time announced.

By definition, any packing $\boldsymbol{\pi}(\lambda)$, if it exists, satisfies all demands (Problem 3). Packing $\boldsymbol{\pi}(\infty)$, which necessarily exists, is the packing which satisfies all demands while minimizing the volume. As $\boldsymbol{\pi}^*$ is a (not necessarily optimal) solution of **Multi_Bot_1P**, then its volume is at least $\text{vol}(\boldsymbol{\pi}(\infty))$. Finally, if $H^* < 3P$, then packing $\boldsymbol{\pi}(H^*)$ exists since $\boldsymbol{\pi}^*$ is a solution of **Multi_Bot_1P**$[H^*, T]$: it satisfies all demands, its volume is at most $TH^*$, its maximum at most $H^*$ and its scale is at most $T$. □

# 6 Conclusion and perspectives

In this article, we present a $\frac{4}{3}$-approximation for a combinatorial problem - generalizing high-multiplicity **IMS** - which consists in finding the optimum schedule for a fleet of reconfigurable robots given some demands for all types of processed jobs. We believe that this problem, we called **Multi_Bot**, could describe many situations involving reconfigurable systems, such as work plannings for teams of employees for example. Therefore, our approximation algorithm is, in our opinion, not only an important achievement for the industrial use of reconfigurable robots but also for other scheduling areas. Indeed, it offers the theoretical guarantee to be close to the optimum schedule, and in practice the performance should be in fact much smaller than this upper bound.

A drawback of our algorithm could be the exponent 7 over parameter $P$ in its running time. For the reconfigurable robots application, it is not since in practice $P$ is smaller than 15 (no more than 14 elementary robots of MecaBotiX can be assembled together). Nevertheless, this time complexity has to be taken into account and perhaps some improvements could be proposed.

A natural question is whether this approximation factor $\frac{4}{3}$ can be improved. The answer is yes, as we are currently writing a PTAS for **Multi_Bot**. This is a very interesting theoretical result, as it shows us that any approximation factor can be achieved, but, at the same time, it does not offer a performing running time for industrial application. This contribution will be presented soon.

A possibility would be to design the same algorithm by replacing LPT-FIRST with another heuristic for **IMS**, for example MULTIFIT. Indeed, the approximation ratio of MULTIFIT is $\frac{13}{11}$. However, the analysis of MULTIFIT is much more involved [18] than the one for LPT-FIRST. As a consequence, it is more difficult to see which are the properties our packings should satisfy in order to obtain a lower approximation ratio for **Multi_Bot**. Nevertheless, it will be the next direction of research to look at on this subject.

# References


[1] O. Battaïa, X. Delorme, A. Dolgui, J. Hagemann, A. Horlemann, S. Kovalev, and S. Malyutin. Workforce minimization for a mixed-model assembly line in the automotive industry. *International Journal of Production Economics*, 170:489–500, 2015.

[2] U. Beşikci, U. Bilge, and G. Ulusoy. Multi-mode resource constrained multi-project scheduling and resource portfolio problem. *European Journal of Operational Research*, 240(1):22–31, 2015.

[3] N. Boysen, P. Schulze, and A. Scholl. Assembly line balancing: What happened in the last fifteen years? *European Journal of Operational Research*, 301(3):797–814, 2022.

[4] N. Brauner, Y. Crama, A. Grigoriev, and J. van de Klundert. A framework for the complexity of high-multiplicity scheduling problems. *J. Comb. Optim.*, 9(3):313–323, 2005.

[5] H. Brinkop and K. Jansen. High multiplicity scheduling on uniform machines in FPT-time. *CoRR*, abs/2203.01741, 2022.

[6] M. Chaikovskaia. *Optimization of a fleet of reconfigurable robots for logistics warehouses*. PhD thesis, Université Clermont Auvergne, France, 2023.

[7] M. Chaikovskaia, J.-P. Gayon, and M. Marjollet. Sizing of a fleet of cooperative and reconfigurable robots for the transport of heterogeneous loads. In *2022 IEEE 18th International Conference on Automation Science and Engineering*, pages 2253–2258, 2022.

[8] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, 1978.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the theory of NP-completeness*. 1979.

[10] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[11] S. Hartmann and D. Briskorn. An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 297(1):1–14, 2022.

[12] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. In *Procs. of FOCS*, pages 79–89. IEEE Computer Society, 1985.

[13] D. Knop, M. Koutecký, and M. Mnich. Combinatorial $n$-fold integer programming and applications. *Math. Program.*, 184(1):1–34, 2020.

[14] S. T. McCormick, S. R. Smallwood, and F. Spieksma. A polynomial algorithm for multiprocessor scheduling with two job lengths. *Mathematics of Operations Research*, 26(1):31–49, 2001.

[15] MecaBotiX. https://www.mecabotix.com/, 2023.

[16] M. Mnich and R. van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Comput. Oper. Res.*, 100:254–261, 2018.

[17] M. Mnich and A. Wiese. Scheduling and fixed-parameter tractability. *Math. Program.*, 154(1-2):533–562, 2015.

[18] M. Yue. On the exact upper bound for the multifit processor scheduling algorithm. *Annals of Operations Research*, 24(1):233–259, 1990.