Sizing and scheduling for a fleet of reconfigurable mobile robots

Benoît Albert[a], Mari Chaikovskaia[b], Jean-Philippe Gayon[a] and Alain Quilliot[a]

[a]INP Clermont Auvergne, Clermont Auvergne University, CNRS, Mines de Saint-Etienne, Clermont-Ferrand, France
[b]Univ. Grenoble Alpes, CNRS, Grenoble INP, G-SCOP, 38000 Grenoble, France

ARTICLE HISTORY
Compiled March 18, 2025

ABSTRACT
We consider a fleet of elementary robots that transport items within a warehouse. These elementary robots can be connected in different ways to transport loads of different types. The capacity of the resulting poly-robot depends on its configuration and on the load type being transported. For each load type, we have to meet a transportation demand within a given time interval. Several poly-robots may be required either simultaneously or sequentially to meet this demand. We have to decide how to decompose the demand into transportation tasks. A transportation task specifies the load type, the corresponding number of loads and the configuration of the involved poly-robot. The objective is to determine the transportation tasks and schedule them over time, in order to minimize the investment cost (related to the fleet size) and the running costs (related to the use of the fleet). We show that the resulting **Multi-Bot** problem is strongly NP-hard. We also study several special cases that can be solved in polynomial time and derive from our theoretical results an efficient heuristic for the general case. A numerical study shows that the heuristic algorithm is effective even for large instances.

KEYWORDS
Fleet Sizing, Reconfigurability, RCPSP, Line balancing

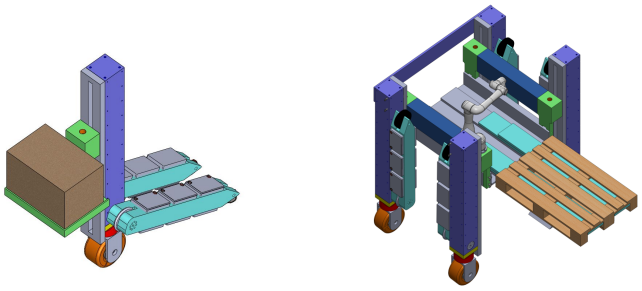
1. Introduction

1.1. *Industrial motivation*

This work is motivated by the increasing robotization of manufacturing and logistics processes. New generation of mobile robots is now able to cooperate in order to handle or transport large or heavy objects. For example, the Mecabotix company designs and prototypes poly-robots such as the M3-Cooper models (see Figure 1). A *poly-robot* refers to a vehicle obtained by combining together several *elementary robots*. A p-bot refers to a poly-robot involving exactly p elementary robots. We call the number p a *configuration*. These poly-robots can be reconfigured to adapt to the loads to be transported (in terms of size and mass) and to navigate independently in environments such as warehouses, manufacturing plants or construction sites. A *load* is a physical object (a box, a pallet, etc) to be transported, and not a number that would indicate a weight or a volume. A *load type* is a group of loads that share the same characteristics with respect to the transportation process. Figure 1 shows two examples of M3-Cooper

Corresponding author : j-philippe.gayon@uca.fr

poly-robots carrying different load types. The 1-bot in Figure 1(a) carries a box, while the 4-bot in Figure 1(b) carries a pallet.



(a) 1-bot (b) 4-bot

Figure 1. Examples of M3-Cooper reconfigurable poly-robots (MecaBotiX 2022)

Figure 1 Alt text: The first figure shows a mono-bot carrying a box and the second figure shows a quadri-bot carrying a palette.

In partnership with the LIMOS CNRS laboratory, we have collaborated with the Mecabotix company to address the following problem (Chaikovskaia, Gayon, and Marjollet 2022). Different load types (such as small boxes, large boxes, pallets, etc.) must be transported within a warehouse between a pickup and delivery platform and designated storage areas within a specified time horizon. Depending on the number of elementary robots in a given configuration, the resulting poly-robot can carry different types and quantities of loads. Poly-robots can be reconfigured over time in a negligible time. For example, a 4-bot and a 2-bot can be reconfigured into one 3-bot and three 1-bots. The time horizon is divided into elementary periods (e.g. intervals of 15 minutes). Dividing the time space into periods responds to the need for synchronization between the different processes (transportation, reconfiguration, production and logistics). Between two periods, poly-robots can be reconfigured, and these configurations remain fixed for the rest of the period. The number of loads (i.e. the number of physical objects) of a given load type that a poly-robot can transport in a given period depends on the load type and the configuration. For each load type, there is a *transportation demand* to be met on the time horizon. This demand can be satisfied through several *transportation tasks* that can be executed sequentially or in parallel, relying on the same poly-robot or on several poly-robots (possibly in different configurations). A transportation task specifies the load type, the number of loads and the configuration of the involved poly-robot. Deriving the transportation tasks from the demand is therefore a key aspect of the problem. Another important issue is how to assign these transportation tasks to the periods. There are two types of costs. Investment costs are related to the number of elementary robots in the fleet. Running costs are related to the use of the robot fleet (e.g. maintenance costs, energy costs, wear costs). The objective is to determine the transportation tasks and schedule them over time, in order to minimize the sum of investment and running costs. We call this problem **Multi-Bot**.

1.2. *Literature review*

In many industrial contexts, automated production must adapt to rapidly changing demands for a wide variety of customized products. This flexibility requirement can be

met through reconfiguration. Once an operation (whether related to the production of goods or their transportation) has been completed, the supporting infrastructure can be redesigned by adding, removing, or replacing components, or by modifying the connections between these components. These components may include hardware (such as robots and instruments), software, or human resources. They work as renewable resources (see Dauzère-Pérès, Roux, and Lasserre (1998); Lee et al. (2013); Hartmann and Briskorn (2022)) that move within the production area to match current production or transportation needs during a given production cycle. By strategically allocating these resources to specific operations, one can not only complete the task but also improve its speed, increase throughput, and reduce costs.

In what follows, we review several streams of research that share characteristics with the specific **Multi-Bot** problem, which arose from the concerns of the Mecabotix company. First, it is closely related to the sizing and control of AGV (Automated Guided Vehicles) fleets, since our poly-robots may be considered as some kind of AGV. Next, there are also similarities with the well-known RCPSP (Resource Constrained Project Scheduling Problem) and the RIP (Resource Investment Problem), because of the scheduling features contained in our problem (elementary robots as resources, and assignment of the transportation tasks to the periods). The **Multi-Bot** problem has also common features with the line balancing problem, due to the specific structure of our objective function which leads to a balanced distribution of the use of elementary robots between periods. Finally, it is related to the bin packing problem (Garey and Johnson 1979), as we shall see during the structural analysis of the problem (see Section 4).

The literature on AGV fleet sizing and operational planning is extensive (see Fragapane et al. (2021) for a survey on planning and control of autonomous mobile robots for intralogistics). It is important to note that robots are typically expensive, and the sizing of a robot fleet is a critical issue. The literature on robot fleet sizing deals with either stochastic (see e.g. Koo, Jang, and Suh (2004); Choobineh, Asef-Vaziri, and Huang (2012)) or deterministic models (see e.g. Egbelu (1987); Rjeb, Gayon, and Norre (2021)). The literature considers a variety of objective functions. Most often, the objective is to minimize the number of robots required to complete a set of transportation tasks within a given time interval. Several works consider more elaborate cost functions (see e.g. Etezadi and Beasley (1983); Beaujon and Turnquist (1991); Sinriech and Tanchoco (1992); Chaikovskaia et al. (2021)).

The proximity between the **Multi-Bot** problem and the RCPSP scheduling problem, in particular its multi-mode variants, arises from the role played by the elementary robots as renewable resources, that are transferred between transportation tasks. There is a substantial body of literature on RCPSP (see Hartmann and Briskorn (2022) for a survey). In multi-mode RCPSP (see Dauzère-Pérès, Roux, and Lasserre (1998); Wḱeglarz et al. (2011); Beşikci, Bilge, and Ulusoy (2015); Tao and Dong (2018); Liu et al. (2018)), a job can be executed in different modes, each of which affects both its duration and its resource requirements. In many cases, adapting resources to meet job targets may induce a setup that combines additional time and economic cost. The RIP is the dual problem of the RCPSP. While the RCPSP aims to find the shortest project duration given the available amount of resources, the RIP seeks to minimize the total resource cost given a time limit on the project. It was first introduced by Möhring (1984) who presents it as a problem of "scarce time" while the RCPSP is a problem of "scarce resource". The multi-mode variant of the RIP, where a job can be executed in different modes, was also investigated (Hsu and Kim 2005; Gerhards 2020).

The above performance criterion leads us to balance, as much as possible, the distribution of transportation tasks across periods. This balancing feature is typical of line balancing which usually refers to the distribution of jobs among machines and periods on an assembly line to minimize machine stress and enhance the robustness of the production system (see Leus and Herroelen (2004); Dolgui and Proth (2013); Özceylan et al. (2019); Boysen, Schulze, and Scholl (2022)). In some version of the line balancing problem, the objective is to minimize the number of workstations (Moon, Logendran, and Lee 2009). It's worth mentioning the work of Battaïa et al. (2015) who consider an assembly line in the automotive industry. In this paper, the resources are identical, and the processing time of a task depends on the number of workers assigned to it. Workers can move from one station to another at the end of each task, with negligible travel time. The objective is to assign tasks to workers in order the minimize the number of workers.

Although our **Multi-Bot** problem shares characteristics with the above problems, it differs in the following fundamental aspects. First, we consider poly-robots that can adapt their configuration to the objects to be carried, whereas the AGV literature usually deals with non-reconfigurable fleets of mobile robots. Secondly, there is a difference in the way jobs (or items for the bin packing problem) are defined. In RCPSP, RIP, line balancing or bin packing problems, the jobs are given in the input. In the **Multi-Bot** problem, the jobs are not part of the input. We are given a demand to satisfy over the time horizon for each load type and we need to decompose this demand into transportation tasks, each task being characterized by a load type, a quantity (the number of loads to carry) and a configuration. Determining the transportation tasks from the demand is therefore a key aspect of the problem which, to the best of our knowledge, has not been addressed in the above literature.

1.3. *Main contributions and outline*

Inspired by a logistics problem presented by a company that designs reconfigurable poly-robots, we present the **Multi-Bot** problem. Given a demand for different load types, we have to determine the transportation tasks, as well as their distribution over time, with the goal to minimize the sum of investment and running costs. In Section 2, we begin by introducing the **Multi-Bot** model and casting it into the ILP (Integer Linear Programming) framework. In Section 3, we present additional constraints that restrict the search space for feasible solutions. In Section 4, we demonstrate that the general case is strongly NP-hard. We also show that the special cases with a single feasible unit configuration ($p = 1$), a single period or a single load type can be solved in polynomial time. Section 5 is devoted to an auxiliary Knapsack-like problem that is used as a pre-process for the heuristic algorithm presented in Section 6. Section 7 presents numerical tests that demonstrate the efficiency of the proposed algorithm.

2. The Multi-Bot problem

2.1. *Problem description*

We consider a fleet of mobile elementary robots that cooperate to transport loads of different types. As explained in the introduction, a p-bot is a poly-robot made of p elementary robots. The number p is called a configuration and belongs to the set of configurations $\mathcal{P} = \{1,\dots, P\}$, where P is the maximum number of elementary robots

in a configuration.

A load is a physical object and a load type is a group of loads that share the same characteristics. There are K load types and we denote by $\mathcal{K} = \{1, \ldots, K\}$ the set of load types.

The time is divided in T periods of equal length. We let $\mathcal{T} = \{1, \ldots, T\}$ be the set of periods. At the beginning of a given period t, the elementary robots are located in a pickup and delivery platform and can be reconfigured to get the best poly-robots for the current period. The configurations remain fixed for the rest of the period.

A poly-robot can only handle one load type in a given period. In period t, a p-bot can transport at most c_{pk} loads of type k. So c_{pk} represents the capacity of a p bot with respect to the load type k. For each load type k, there is at least one feasible configuration p such that the associated p-bot is able to transport that load type, which means that there exists $p \in \mathcal{P}$ such that $c_{pk} > 0$.

The transportation demand for load type k is denoted by d_k, which represents the number of individual loads that have to be transported over the entire time horizon. For a given load type, the transportation demand can be met with one or several poly-robots (possibly in different configurations) and over one or several periods. Hence, we have to decide how to decompose the demand into transportation tasks. A transportation task specifies the load type k, the configuration p of the poly-robot involved and the number of loads n of type k to be transported (with $n \leq c_{pk}$).

Let us introduce the objective function. Let α be the investment cost per elementary robot. Let β be the running costs incurred each time an elementary robot is involved in a transportation task. If a p-bot is used in some period, it counts for $p \cdot \beta$ in the running costs. So the running costs are proportional to the number of trips made by elementary robots. Denote by H_t the number of elementary robots used in period t for transportation tasks, by $H^{Max} = \max_t H_t$ the number of elementary robots involved in the whole process, and by $H = \sum_t H_t$ the number of elementary robot trips. The total cost is then:

$$Cost = \alpha \cdot H^{Max} + \beta \cdot H. \tag{1}$$

The investment cost $\alpha \cdot H^{Max}$ is proportional to the number of elementary robots in the fleet. The running costs $\beta \cdot H$ must be interpreted as the costs related to the use of the elementary robots. The goal is to determine the transportation tasks, as well as how to assign them to periods, in order to minimize overall costs while satisfying all the demand. We call **Multi-Bot** this optimization problem.

An illustrative example Let us illustrate the **Multi-Bot** problem with a simple example. Consider three load types 1, 2 and 3. Let $T = 4$ periods and the following demands: $d_1 = 3, d_2 = 4, d_3 = 1$. Capacities are given in Table 1.

Configuration	Load type		
	$k = 1$	$k = 2$	$k = 3$
1-bot	0	0	0
2-bot	1	0	0
3-bot	2	2	0
4-bot	3	2	1

Table 1. Capacities for a simple example

Figure 2 represents the Gantt chart of a feasible solution for this example. The ordinate axis represents the reference number of the elementary robots, while the abscissa axis shows the reference number of the periods. Each rectangle indicates the load type being transported and the number of loads. The height of a rectangle corresponds to the number of elementary robots involved in the poly-robot. For instance, in period 1, a single load of type 3 is carried by a 4-bot. Then it reconfigures into a 3-bot to carry two loads of type 2 in period 2, and so on. For this feasible solution, the number of elemeaentary robots required in each period are: $H_1 = 4$, $H_2 = H_3 = 3$ and $H_4 = 4$. The number of elementary robots to achieve the whole process is $H^{Max} = 4$. The total number of elementary robot trips is $H = \sum_t H_t = 14$. Note that this solution is optimal, since we can neither reduce H^{Max}, nor H.

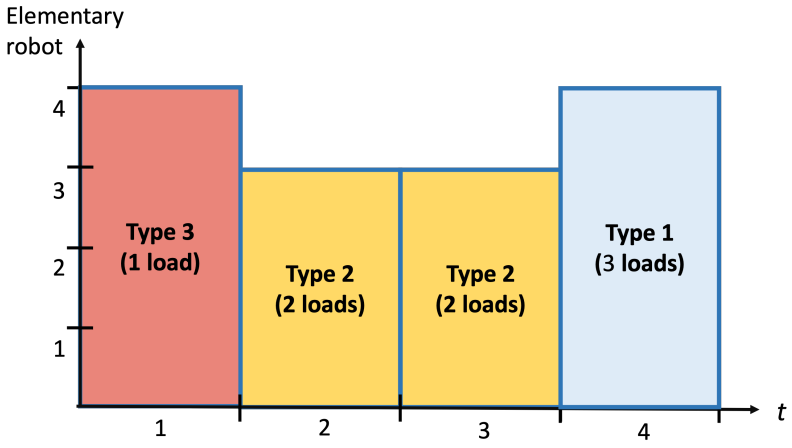


Figure 2. An example: Gantt chart

Figure 2 Alt text: The figure shows a Gantt chart with 4 time periods. In period 1, a load of type 3 is carried by a 4-bot. In periods 2 and 3, a load of type 2 is carried by a 3-bot. In period 4, three loads of type 1 by one 3-bot.

Discussion of the application context As a typical application, the Mecabotix company envisions an industrial context where the different load types correspond to materials stored in different areas within a large warehouse. A transportation task for type k consists in picking up a number of loads $n \leq c_{pk}$ from a pick-up and delivery platform A_0, loading it into a p-bot, transporting it, storing it in the dedicated area A_k and returning to the pick-up and delivery platform A_0. Alternatively, the task could involve moving to A_k, picking up a load $n \leq c_{pk}$ of type k, and carrying it back to A_0.

Each transportation task is likely to take between 10 and 15 minutes, with part of this time dedicated to the handling of the load within the target area A_k. Of course, the length of the period will depend on the context and the size of the warehouse.

With regard to the time and human resources required for the reconfiguration process, the reconfiguration of a poly-bot is estimated to take less that 2 minutes, at the junction between two consecutive periods, and is carried out by the workers involved in the warehouse. Moreover, it is expected in practice that reconfiguration will be infrequent, in the sense that in most cases, a p-bot achieving the transportation of loads of type k at period t will keep on with the same kind of transportation task at time

$t+1$. With the industrialization of poly-robots, reconfiguration times should become negligible in the future.

Uncertainty about the duration of trips, reconfiguration and load handling can be difficult to manage in practice. Structuring the time space into periods responds to the need for synchronization between the transportation tasks and the human operators in charge of the reconfiguration process, and between the transportation tasks and external processes (production or logistics). The length of a period is likely to be between 10 and 20 minutes, depending on the context and the size of the warehouse. The whole process would take place throughout the day, so the number of periods is about 40. The duration of a transportation task, which in our model is assumed to take place within a single period, should be less than that the duration of a period.

It is natural to assume that the capacity $c_{p,k}$ increases with p, but the correspondence between the parameter p and the capacity may not be linear. Typically, we may be forced to add at least two elementary robots to a given configuration p in order to make the resulting poly-robot able to carry additional loads.

Of course, some assumptions might need to be modified. At the end of Section 2.2, we will explain how to include in our ILP model scheduling constraints (time window and precedence) and how to consider transportation tasks that require more than one time period.

2.2. *ILP formulation*

We denote by x^t_{pk} the decision variable representing the number of poly-robots in configuration p transporting loads of type k in period t.

Multi-Bot ILP:

$$\min \ \alpha \cdot H^{Max} + \beta \cdot \sum_{t \in \mathcal{T}} H_t \tag{2}$$

subject to :

$$\sum_{t \in \mathcal{T}} \sum_{p \in P} c_{pk} \cdot x^t_{pk} \geq d_k \qquad \forall k \in \mathcal{K} \tag{3}$$

$$H_t = \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}} p \cdot x^t_{pk} \qquad \forall t \in \mathcal{T} \tag{4}$$

$$H^{Max} \geq H_t \qquad \forall t \in \mathcal{T} \tag{5}$$

$$x^t_{pk}, H_t, H^{Max} \in \mathbb{N}, \qquad \forall k \in \mathcal{K}, \forall p \in \mathcal{P}, \forall t \in \mathcal{T} \tag{6}$$

The objective function (2) consists in minimizing simultaneously the number of elementary robots H^{Max} and the number of elementary robot trips $\sum_t H_t$. Constraint (3) means that we must have a sufficient capacity to satisfy demand d_k. Quantity $\sum_{t=1}^{T} \sum_{p=1}^{P} c_{pk} \cdot x^t_{pk}$ represents the maximum number of loads of type k that can be transported over the horizon, given the x^t_{pk}. Constraint (4) expresses the number H_t of elementary robots used in period t as a function of the decision variables x^t_{pk}. Constraint (5) means that H^{Max}, the number of elementary robots necessary to achieve the whole process, must be greater than or equal to H_t for all t. Constraint (6) reminds that all decision variables are non negative integers.

Note that if variables x^t_{pk} are integral, then optimal H^{Max} and H_t are also integral. We deduce that we can relax the integrality constraints on H^{Max} and H, thus making

it easier for a MILP (Mixed Integer Linear Programming) solver to handle.

Including scheduling constraints We can easily restrict the transportation of certain load types to specific periods. Let us assume that loads of type k must be transported in a time window $\{\tau_k^{min}, \cdots, \tau_k^{max}\}$. Such a constraint may be expressed as:

$$x_{pk}^t = 0, \forall t \notin \{\tau_k^{min}, \cdots, \tau_k^{max}\}, \forall p \in \mathcal{P}$$

This constraint is not going to deeply modify the structure of the problem.

We can also require that the transportation of all loads of some type k_1 be completed before the transportation of another load type k_2 begins. To include this constraint in the ILP, we introduce an additional binary variable $Z_t^{k_1}$ which is equal to 1 if some transportation task involving load type k is performed at period t, and 0 otherwise. Then we set the following constraints, M being a large number:

$$M \cdot Z_t^{k_1} \geq \sum_p x_{pk_1}^t \qquad \forall t \in \mathcal{T} \tag{7}$$

$$M \cdot Z_t^{k_1} + \sum_{p \in \mathcal{P}} \sum_{u=1}^{t} x_{pk_2}^u \leq M \qquad \forall t \in \mathcal{T} \tag{8}$$

Constraint (7) means that $Z_t^{k_1}$ can be equal to 0 only if no load of type k_1 is processed during period t. Thus Constraint (8) tells that if any load of type k_1 is processed at period t, then loads of type k_2 cannot be processed during periods $1, \ldots, t$. The structure of these constraints, which linearize logical implications through Big M formulations, tends to significantly increase the gap to optimality induced by relaxing the integrality constraints of the ILP and makes the problem more complex.

Non-identical transportation times To simplify, we have assumed so far that a transportation task can take place in a single period. However, allowing some transportation tasks to last more than one period does not significantly change the structure of the resulting **Multi-Bot** problem. Let Δ_{pk} be the transportation time for loads of type k in configuration p. It represents the number of consecutive periods required to complete a transportation task.

The ILP must be adapted as follows. The decision variable x_{pk}^t must now be interpreted as the number of p-bots that start transporting loads of type k in period t . The variable H_t must include all the transportation tasks started between $t - \Delta_{pk} + 1$ and t: $H_t = \sum_{p,k} \sum_{u=t-\Delta_{pk}+1}^{t} p \cdot x_{pk}^u$.

3. Enhancing the model with additional constraints

The goal of this section is to show that it is possible to restrict the space for the search of good solutions without inducing any loss in the quality of the optimal solution.

3.1. *The most profitable configuration*

We first introduce the notion of most profitable configuration and show that the other configurations play a marginal role in the optimal solution. This will be used in section

5 and 6 to design an efficient heuristic algorithm.

We define the *marginal capacity* of a p-bot, with respect to load type k, as the capacity per elementary robot $\frac{c_{pk}}{p}$.

Definition 1. The *most profitable configuration* for load type k is the configuration $p_0(k)$ that maximizes the marginal capacity:

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

If several configurations maximize the marginal capacity, we choose the smallest one.

The most profitable configuration $p_0(k)$ is not necessarily the one that should be systematically chosen in the optimal solution. However, we can show that the other configurations play a marginal role in the optimal solution. The following lemma gives an upper bound for the number x_{pk}^t of p-bots used in period t to transport loads of type k, when p is not the most profitable configuration. The proof of this lemma is given in Appendix A.

Lemma 1. *When a configuration p is not the most profitable ($p \neq p_0(k)$), adding the constraint*

$$x_{pk}^t \leq \frac{p_0(k)}{GCD(p_0(k), p)} - 1 \tag{9}$$

*does not modify the optimal value of **Multi-Bot** (GCD means here Greatest Common Divisor).*

We deduce from Lemma 1 that we should focus on the most profitable configurations. The algorithms we describe in Section 5 and 6 derive from this interpretation of Lemma 1. Moreover, the higher the demand, the more we should use the most profitable configuration.

We end by introducing the following quantities for $p \neq p_0(k)$.

$$x_{pk}^{Max} = \frac{p_0(k)}{GCD(p_0(k), p)} - 1 \tag{10}$$

$$w_k^{Max} = \sum_{p \neq p_0(k)} p \cdot x_{pk}^{Max} \tag{11}$$

The quantity x_{pk}^{Max} is the bound established in Lemma 1 and represents the maximum number of p-bots used in a given period to transport loads of type k, when p is not the most profitable configuration. The quantity w_k^{Max} represents the maximum number of elementary robots, in configurations other than $p_0(k)$, that should be used in a given period to transport loads of type k. Since, $p \leq P$, we have immediately the following bounds: $x_{pk}^{Max} \leq P$ and $w_k^{Max} \leq P^3$. These bounds will be useful to study the complexity of the **Multi-Bot** problem (see Section 4) and of the complexity of the heuristic algorithm (see Section 6).

An illustrative example We end this section by considering a simple example to illustrate the notion of most profitable configuration. For some type load k_0, Table 2 gives for each configuration the capacities, marginal capacities and $x_{pk_0}^{Max}$.

Configuration p	1	2	3	4
Capacity c_{pk_0}	1	3	3	8
Marginal capacity $\frac{c_{pk_0}}{p}$	1	1.5	1	2
Maximum number of p-bots ($x_{pk_0}^{Max}$)	3	1	3	N/A

Table 2. Marginal capacity: An example

On this example, we see that the most profitable configuration for loads of type k_0 is configuration 4. We also have $w_{k_0}^{Max} = 14$, which represents the maximum number of elementary robots (in configurations 1, 2 or 3) used to transport loads of type k_0 in a given period.

3.2. *Other additional constraints*

The cumulative number of periods induced by all the trips made in order to satisfy demand must be at least $Q = \sum_k \left\lceil \frac{d_k}{c_{p_0(k),k}} \right\rceil$. Based on this fact, we derive the following two additional constraints that H and H^{Max} must satisfy:

$$H^{Max} \geq \left\lceil \frac{Q}{T} \right\rceil$$

$$\sum_t H_t \geq Q$$

Finally, we also see that if t is such that the difference between H^{Max} and H_t is greater than P, then it is possible to shift some trips from the time value t^{max} associated with H^{Max} to t without deteriorating the current objective value. We conclude that we can impose the following constraint:

$$\forall t, \quad H_t \geq H^{Max} - P$$

This constraint is used in the next section to prove that the special case with a single load type ($K = 1$) can be solved in polynomial time.

4. Complexity of the Multi-Bot problem

Though our main focus is the design of efficient and flexible solution approaches for the **Multi-Bot** problem, we first analyze its complexity. In particular, we show that the well-known bin packing problem can be polynomially reduced to **Multi-Bot**, thereby proving that **Multi-Bot** is strongly NP-hard. This analysis paves the way for the design of an efficient heuristic algorithm for solving **Multi-Bot** (see Section 6).

Theorem 1. *The **Multi-Bot** problem is strongly NP-Hard, even if we restrict it to the case where there is a single feasible configuration $p(k)$ for each load type k and $\beta = 0$ in the objective function.*

Let us consider the special case of **Multi-Bot** when $\beta = 0$ and when, for each product type k, there is single feasible configuration $p(k)$. This means that only one configuration $p(k)$ can be used to transport loads of type k. The number of transporta-

tion tasks for load type k is then simply $n_k = \left\lceil \frac{d_k}{c_{p(k),k}} \right\rceil$. Since $\beta = 0$, the objective becomes to minimize the number H^{Max} of elementary robots.

This special case of **Multi-Bot** resembles the well-known bin packing problem. The periods correspond to the bin packing boxes. The load types $1, \cdots, K$ corresponds to the bin packing items. Each item k has a size $p(k)$ and we have n_k copies of this item.

Therefore, our intuition suggests that **Multi-Bot** should be strongly NP-hard, as is the bin packing problem. However, because the weights $p(k)$ are not arbitrary numbers but belong to a set $\{1, \ldots, P\}$, where P is part of the structural components of the encoding size of a **Multi-Bot** instance, we must proceed with caution. For this reason, we provide a detailed proof of Theorem 1 in Appendix B.

The **Multi-Bot** problem is NP-Hard, but fixing the value of one of the key parameters P, T or K to 1 turns it into a problem that can be solved in polynomial time, as stated in the following theorem (see the proof in Appendix C.)

Theorem 2. *The **Multi-Bot** problem can be solved in polynomial time if $T = 1$ or $P = 1$ or $K = 1$.*

We end this section by discussing the case when the marginal capacity $\frac{c_{pk}}{p}$ decreases in p. More precisely, assume that for each load type k, $\frac{c_{p'k}}{p'} \leq \frac{c_{pk}}{p}$ for every $p' > p$. The most profitable configuration is then configuration 1 for every k. We don't have to use configurations $p > 1$ in the optimal solution, since we can always replace a p-bot by p 1-bots with a higher total capacity. This reduces the **Multi-Bot** problem to the special case $P = 1$ which is polynomial in time and can be solved analytically (see Appendix C). For this reason, in the numerical study we will choose capacities with marginal capacities that do not decrease.

5. Pre-processing less profitable configurations

In this section, we present a pre-processing procedure that will be used in the heuristic algorithm presented in Section 6. The general idea is to solve the following auxiliary problem. For a given load type k and a given number w of elementary robots, the question is how to determine the set of poly-robots that maximizes the total capacity while using configurations that are not the most profitable one ($p \neq p_0(k)$). This auxiliary problem is a Knapsack-like problem and will be called **Aux-Bot**(k, w). Thanks to Lemma 1, we show that we can solve it in a very efficient way with a Dynamic Programming (DP) algorithm.

From Section 3.1, we know that most loads of type k should be handled as much as possible by the most profitable configuration $p_0(k)$. More precisely, Lemma 1 tells us that x^t_{pk} can be chosen to be no more than x^{Max}_{pk} when $p \neq p_0(k)$. Moreover the number of elementary robots that are not in the most profitable configuration in period t (i.e. $\sum_{p \neq p_0(k)} p \cdot x^t_{pk}$) never exceeds w^{Max}_k. So, a natural way to get rid of the variables x^t_{pk}, $p \neq p_0(k)$, during our resolution process is to replace them, for any period t, by the weight $w = \sum_{p \neq p_0(k)} p \cdot x^t_{pk}$.

Once we have this weight value, it is sufficient to retrieve related values x^t_{pk} by setting $x^t_{pk} = y_p$, where $y = (y_p)_{p \neq p_0}$ is an optimal solution of the following Knapsack-like problem **Aux-Bot**(k, w):

$$\textbf{Aux-Bot}(k,w): \quad v = \max \sum_{p \neq p_0(k)} c_{pk} \cdot y_p \tag{12}$$

subject to :

$$\sum_{p \neq p_0(k)} p \cdot y_p \leq w \tag{13}$$

$$y_p \leq x_{pk}^{Max} \qquad \forall p \tag{14}$$

$$y_p \in \mathbb{N} \qquad \forall p \tag{15}$$

In the above ILP formulation, w is the maximum number of elementary robots that are not in the most profitable configuration in a given period. Accordingly, we want to maximize the capacity $v = \sum_{p \neq p_0(k)} c_{pk} \cdot y_p$ that is achieved by vector y at period t. Constraint (13) expresses the fact that performing y should not involve more than w elementary robots. Constraint (14) tells us that, according to Lemma 1, we can bound each value y_p by x_{pk}^{Max}.

What happens now is that we do not need to solve an instance of **Aux-Bot**(k,w) each time we try a decision about w. As we will see, for any k, all instances **Aux-Bot**(k,w), $0 \leq w \leq w_k^{Max}$, may be solved in an exact way and in polynomial time with the DP algorithm *A_Bot(k)*. Related solutions can be stored inside a table *Table_Bot*$[k]$. For each load type k, *Table_Bot*$[k]$ provides us with a list of 3-uples (w,v,y), such that v is the optimal value of **Aux-Bot**(k,w) and $y = (y_p)_{p \neq p_0(k)}$ is the related optimal solution. Then, each time we have to deal with some pair (t,k) and decide about values $x_{pk}^t, p \neq p_0(k)$, we only have to decide about value w, next retrieve v and y from the table *Table_Bot*$[k]$ and finally set $x_{pk}^t = y_p$ for any $p \neq p_0(k)$.

According to this, it seems natural to build the table *Table_Bot* in advance, as part of a pre-process distinct and independent from the resolution of **Multi-Bot**. This approach is fully justified in practice. The construction of the table *Table_Bot* only involves infrastructure data about poly-robot capacities. Conversely, solving **Multi-Bot** must take place at the beginning of each time horizon and adapt to current demands d_k.

Let us explain how algorithm *A_Bot(k)* works. *A_Bot(k)* is a DP algorithm and, like any DP algorithm, it relies on a time space, a state space, and a set of transitions commanded by decisions. In the present case:

- The DP time space (not to be confused with $\mathcal{T}$) is the set $\{0\} \cup \{p = 1, \ldots, P, p \neq p_0(k))$; The successor $Succ(p)$ of a time value p is $p+1$ in case $p+1 \neq p_0(k)$ and $p+2$ otherwise.
- The state space is the set of all non negative number w such that $w \leq w_k^{Max}$. Each state w is given together its best cumulated capacity v.
- A decision d at time p is any non negative integral number between 0 and x_{pk}^{Max}, which means at time p the value of $y_{Succ(p)}$.
- Related transition increases w by $Succ(p) \cdot d$ and cumulated capacity v by $c_{Succ(p),k} \cdot d$.
- Initial state is 0 with related value 0; Related capacity v is 0. Final states are all w which could be reached at time P, provided with a final cumulative capacity v.
- Bellman principle: for any p, we only keep Pareto pairs (w,v), which means that

we forbid two pairs (w, v) and (w', v') related to a same p to be such that $w \leq w'$ and $v \geq v'$.

In order to easily retrieve the solution vectors y, *A_Bot(k)* implements this scheme according to a backward driven strategy (see Algorithm 1), that means while scanning the time space downward from a fictitious period $P+1$ to a period $Start$, equal to 2 if $p_0(k) = 1$ and equal to 1 otherwise. It computes, for each $p \neq p_0(k)$, a list $STATE[p]$ of 3-uples (w, v, d), where (w, v) is Pareto optimal in the above sense and d means the decision at time p, that means the value y_p. Scanning the time space backward means, for each $p \neq p_0(k)$, defining its predecessor $Pred(p)$, in such a way that $q = Pred(p)$ if and only if $p = Succ(q)$.

Algorithm 1 *A_Bot*(k)

Periods $P+1, Start$ and operator $Pred$ are defined in a natural way as described above;
Initialize vector $STATE = (STATE[p], p = Start, \ldots, P+1)$ whose values are lists of 3-uple (w, v, d) with:
    $STATE[P+1] = \{(0, 0, -1)\}$;
    For any $p \neq P+1$, $STATE[p] = Null$ = Empty list;
Set $p = P+1$;
While $p \neq Start$ do
    For any (w, v, d) in $STATE[p]$ do
        For $a = 0, \ldots, x_{pk}^{Max}$ do
            $w1 = w + a \cdot p$;
            $v1 = v + a \cdot c_{Pred(p),k}$;
            If no $(w2, v2, a2)$ exists in $STATE[Pred(p)]$ such that $w2 \leq w1$ and $v2 \geq v1$ then
                Insert $(w1, v1, d)$ into $STATE[Pred(p)]$;
                Remove from $STATE[Pred(p)]$ any $(w2, v2, a)$ such that $w2 \geq w1$ and $v2 \leq v1$;
    Set $p = Pred(p)$;
For any $(w, v, d) \in STATE[Start]$ do
    Retrieve from $STATE$ the optimal solution y of **Aux_Bot**(k, w);
    Insert (w, v, y) into $Table\_Bot[k]$;

Algorithm *A_Bot*(k) solves **Aux-Bot**(k, w) in a single run. It is polynomial time since the number of states w is bounded by $w_k^{Max} \leq P^3$. The number of possible decisions is bounded by P and the number of time steps is $P+1$. Hence the algorithmic complexity is in $O(P^5)$, as stated in the following lemma.

Lemma 2. *A_Bot(k) solves all instances* ***Aux-Bot***(k, w), $0 \leq w \leq w_k^{Max}$, *in polynomial time with an algorithmic complexity in* $O(P^5)$.

Speeding *A_Bot*(k) and reducing the size of *Table_Bot*$[k]$. We may adapt the construction of *Table_Bot* in such a way that we better control the number of states generated by the procedure *A_Bot*(k) and by the same way we limit its running time. In order to dot it, we rewrite exact *A_Bot*(k) as a heuristic procedure *A_Bot*(k, S_{max}), with the restriction that state values w must not exceed S_{max}. According to Section 3.1, we should choose S_{max} smaller than P^3.

6. A heuristic algorithm *H_Bot* for the general case

In this section, we focus on designing an efficient algorithm for the **Multi-Bot** problem. The *H_Bot* algorithm proceeds in two steps, with the second step relying on the pre-process described in Section 5. We first present the general idea of *H_Bot* before going into more detail.

First step (use only the most profitable configurations): In the first step, *H_Bot* uses exclusively poly-robots in the most profitable configuration $p_0(k)$, for every k. By doing so, the **Multi-Bot** problem can be seen as a variant of the bin packing problem (see Section 4). We choose the following algorithm to fill the boxes. First, sort the items by decreasing order of their sizes $p_0(k)$. Second, assign them sequentially to the boxes in such a way that the maximal size of a box increases as slowly as possible.

Second step (redistribute to less profitable configurations): After the first step, we have a feasible solution that uses only the most profitable configurations. In the second step, for each load type k and period t, *H_Bot* tries to redistribute w elementary robots (with $w = 0, 1, \ldots, S_{max}$) to less profitable configurations according to the pre-processing procedure described in Section 5. It selects the w that decreases the objective function the most.

6.1. *First step of H_Bot: Use only the most profitable configurations*

The first step of *H_Bot*, called *H_Bot_First_Step*, focuses on most profitable configurations. Algorithm 2 details *H_Bot_First_Step*, which works as follows. First sort the load types by decreasing order $p_0(k)$. Secondly, for each k, do the following. Compute the number $n_k = \lceil d_k / c_{p_0(k),k} \rceil$ of transportation tasks that are required to satisfy all the demand d_k. These n_k transportation tasks are then distributed evenly between the T periods. The remaining transportation tasks are then allocated to the periods using the smallest number of elementary robots.

Algorithm 2 *H_Bot_First_Step*

Order values k by decreasing $p_0(k)$ values and put them in a list L^K;
For any t, initialize H_t as 0;
For any t, k initialize $x^t_{p_0(k),k}$ as 0;
For k in L^K do
    $\lceil d_k / c_{p_0(k),k} \rceil = B \cdot T + F$; /*Euclidean division by T*/
    For any t, set $x^t_{p_0(k),k} = B$ and $H_t = B \cdot p_0(k)$;
    While $F \geq 1$ do
        Pick up t such that H_t is minimal;
        $H_t \leftarrow H_t + p_0(k)$;
        $x^t_{p_0(k),k} \leftarrow x^t_{p_0(k),k} + 1$;
        $F \leftarrow F - 1$;

6.2. *Second step of H_Bot: Redistribute to less profitable configurations*

The principle of the second step of the algorithm, called *H_Bot_Second_Step*, is as follows. From the first step, we have a feasible solution using only the most profitable

configuration and which can be described by the variables $x^t_{p_0(k),k}$ with $t \in \mathcal{T}$ and $k \in K$. In the second step, for each load type k and period t, *H_Bot_Second_Step* tries to redistribute elementary robots to less profitable configurations. More precisely, it iteratively picks up some pair (t,k), some 3-uple (w,v,y) from *Table_Bot*$[k]$ (see Section 5) and redistributes part of variable $x^t_{p_0(k),k}$ among variables $x^t_{pk}, p \neq p_0(k)$, so that the objective function decreases and the number of redistributed elementary robots does not exceed w.

We need to introduce some notations before providing a detailed version of *H_Bot_Second_Step* (see Algorithm 3). We use the following representation of a **Multi-Bot** solution:

- A vector $z = (z^t_k)_{\substack{t\in\mathcal{T}\\k\in\mathcal{K}}}$ such that $z^t_k = x^t_{p_0(k),k}$;
- A vector $\delta = (\delta^t_k)_{\substack{t\in\mathcal{T}\\k\in\mathcal{K}}}$ where each δ^t_k is a 3-uple (w^t_k, v^t_k, y^t_k), belonging to *Table_Bot*$[k]$.

Such a pair (z,δ) gives rise to the **Multi-Bot** solution $x = (x^t_{pk})_{\substack{t\in\mathcal{T}\\p\in\mathcal{P}\\k\in\mathcal{K}}}$ such that:

$$x^t_{pk} = \begin{cases} z^t_k & \text{if } p = p_0(k) \\ (y^t_k)_p & \text{if } p \neq p_0(k) \end{cases}$$

So *H_Bot_Second_Step* starts from the values $z^t_k = x^t_{p_0(k),k}$ computed by *H_Bot_First_Step*. Then, *H_Bot_Second_Step* proceeds in a greedy way. At each iteration, it selects some pair (t,k) that has not yet been selected. For this pair (k,t), it tries every (w,v,y) in *Table_Bot*$[k]$ and keeps the one that decreases the most the objective function.

We now give some details on how we update the solution, given a pair (k,t) and a 3-uple (w,v,y). For load type k, the transportation capacity for the most profitable configuration over the entire horizon is

$$B_k = c_{p_0(k),k} \sum_t z^t_k.$$

Note that this capacity B_k is greater than or equal to the demand d_k and we have therefore an unused transportation capacity $B_k - d_k \geq 0$. The number of removed $p_0(k)$-bots is then

$$R = \left\lfloor \frac{B_k - d_k + v}{c_{p_0(k),k}} \right\rfloor.$$

We update the solution as follows: z^t_k must be increased by R and δ^t_k becomes (w,v,y).

Algorithm 3 *H_Bot_Second_Step*

Start with values $z_k^t = x^t_{p_0(k),k}$ computed according to *H_Bot_First_Step* and from a null vector δ;
Initialize a vector $MARK = (MARK_k^t, t = 1, \ldots, T, k = 1, \ldots, K)$ with Boolean values, to the null vector;
Initialize a vector $NMARK = (NMARK^t, t = 1, \ldots, T)$ with integral values all equal to K;
For any k, set $B_k = c_{p_0(k),k} \sum_t z_k^t$;
For any t, set $H_t = \sum_k p_0(k) z_k^t$;
Set $H = \sum_t H_t$;
Initialize counter value $COUNT$ to $T \cdot K$;
While $COUNT \neq 0$ do
    Choose t and k; **(Instruction 1)**
    For any (w, v, y) in *Table_Bot*$[k]$ compute:
        $R = \lfloor (B_k - d_k + v)/c_{p_0(k),k} \rfloor$;
        $HAux = H + w - R \cdot p_0(k)$;
        $HAux2 = H_t + w - R \cdot p_0(k)$;
        $HMax = \max(\max_{s \neq t} H_s, HAux2)$;
        $Cost = \alpha \cdot HMax + \beta \cdot HAux$;
    Choose (w, v, y) such that $R \leq z_k^t$ and which provides us with the smallest value H^*; **(Instruction 2)**
    Update:
        $MARK_k^t$ is set to 1; $COUNT$ and $NMARK^t$ are decremented by 1;
        B_k is set to $B_k - R \cdot c_{p_0(k),k} + v$;
        H_t is set to $HAux2$ and H is set to $HAux$;
        z_k^t is decremented by R and δ_k^t becomes (w, v, y).

Let us detail instructions 1 and 2.

Instruction 1

- Choice of t: We choose t such that $NMARK^t \neq 0$ and H_t is maximal;
- Choice of k: Once t has been chosen, we target k such that $MARK_k^t = 0$ and that $p_0(k)$ is the largest possible.

Instruction 2

- If the 3-uple (w, v, y) is null, there is no improvement of our current solution (z, δ) and the "update" instruction is not applied.

6.3. *Complexity of H_Bot*

As explained earlier, we consider both the construction of the table *Table_Bot* and the instruction 1 of *H_Bot_First_Step* to be part of a pre-process, and so do not include them in the computation of the complexity of the algorithm *H_Bot*. This is fully justified by the fact that the construction of *Table_Bot* does not aim at any decision making, but only at the processing of data related to existing infrastructures, while the algorithm *H_Bot* is supposed to be run at the beginning of each production cycle (time horizon) in order to make scheduling decisions related to current transportation demands.

The main loop of *H_Bot_First_Step* is indexed on k and its internal "While $F \geq 1$ do" loop is indexed on $F \leq T$. Since the instruction which commands the computation of t may be performed in $O(T)$, we deduce that the complexity of *H_Bot_First_Step* is $O(T^2 \cdot K)$.

The main loop of *H_Bot_Second_Step* is indexed on $K \cdot T$. As it is written (Instruction 1), the instruction "Choose t and k" may be performed in constant time. Since the construction of *Table_Bot* is performed in such a way that the number of elements in any *Table_Bot*$[k]$ never exceeds the parameter S_{max}, we see that the complexity of *H_Bot_Second_Step* is $O(K \cdot T \cdot S_{max})$. We can now state the complexity of *H_Bot* in the following theorem.

Theorem 3. *The complexity of the H_Bot algorithm is* $O(K \cdot T \cdot S_{max} + T^2 \cdot K)$

7. Numerical experiments

We describe here several numerical experiments that were carried out with a twofold purpose. On the one hand, we want to observe some characteristics of the solutions of the **Multi-Bot** problem, specifically the size H^{Max} of the robot fleet. On the other hand, we are interested in the performance of the *H_Bot* algorithm, in particular its ability to approach optimality.

The algorithms are implemented in C++, on a PC AMD Opteron 2.1GHz, while using gcc 4.1 compiler. ILP models are handled by the PuLP library. PuLP is an application programming interface and it can generate data files created in the Mathematical Programming System (MPS) format or Linear Programming (LP) files and call one of the solvers (GLPK, CBC, CPLEX) to solve linear problems. Here we use CBC (Coin Branch and Cut) solver.

7.1. *Instances*

We build an instance by first choosing T, P, K, α, β and next generating the capacities and the demands as follows.

- For each load type k, we randomly select a_k in $\{1, \cdots, 5\}$ and b_k in $\{0, \cdots, a_k\}$. Then we set

$$c_{pk} = a_k p - b_k + \epsilon$$

  with ϵ randomly selected in $\{0, 1\}$ for each configuration p. We denote by $c_k^{mean} = \frac{1}{P} \sum_p c_{pk}$ the mean capacity.

  Our goal is to build here instances that are both non trivial and realistic. The above formula means that we are implementing a capacity c_{pk} that increases with p and is such that the marginal capacity $\frac{c_{pk}}{p}$ also increases with p. This realistic assumption implies that all configurations p are likely to be involved in an optimal solution (see the discussion at the end of Section 4).
- Then we generate the demand for load type k as

$$d_k = \lfloor \gamma \cdot J \cdot T \cdot P \cdot c_k^{mean} \rfloor$$

  where J is randomly selected in $\{1, \cdots, 10\}$. Note that demand d_k increases with demand factor γ.

Parameter γ, called the demand factor, allows us to observe the effect of high demands, that make the processing of the items rely mostly on the most profitable configuration $p_0(k)$. Conversely, it also allows us to see what happens when the demands are low, so that most transportation tasks related to type k are performed by small configurations ($p < p_0(k)$). For intermediate values of γ we expect most profitable configurations to be used.

We call *nominal instance* the instance with parameters ($T = 10$, $P = 6$, $K = 6$, $\alpha = 9$, $\beta = 1, \gamma = 1$). To carry out a sensitivity analysis, we first apply our algorithms to this nominal instance, then change one parameter at a time among $T, P, K, \alpha, \beta, \gamma$, while keeping the others unchanged, as follows:

- $T = 1, 5, 10, 20, 50$
- $K = 1, 3, 6, 8, 10$
- $P = 1, 3, 6, 12, 18$
- $\alpha = 0, 1, 5, 9, 10$ and $\beta = 10 - \alpha$
- $\gamma = 0.1, 0.5, 1, 100, 1000$

Table 3 summarizes the list of instances and their main characteristics. Each instance is identified by an identifier Id. The nominal instance has the identifier $Id = 1$, and appears in bold in the various tables.

Table 3. Characteristics of instances

Id	α	β	γ	T	P	K	c_k^{mean} (*)	d_k
1	**9**	**1**	**1**	**10**	**6**	**6**	**16, 10, 13, 6, 9, 13**	**1920, 1800, 3120, 1800, 3780, 7020**
2	9	1	1	1	6	6	14, 8, 7, 14, 5, 4	840, 288, 252, 420, 270, 120
3	9	1	1	5	6	6	4, 15, 4, 6, 6, 15	840, 1800, 360, 1260, 360, 900
4	9	1	1	20	6	6	17, 4, 13, 12, 6, 13	8160, 480, 6240, 2880, 6480, 10920
5	9	1	1	50	6	6	4, 18, 3, 13, 3, 8	4800, 16200, 8100, 15600, 1800, 12000
6	9	1	1	10	6	1	3, 6, 10, 13, 15, 19	3300
7	9	1	1	10	6	3	10, 18, 7	2400, 10800, 2520
8	9	1	1	10	6	8	17, 10, 14, 13, 15, 19, 12, 17	4080, 1800, 6720, 3120, 9000, 4560, 5760, 10200
9	9	1	1	10	6	10	7, 10, 13, 19, 16, 18, 5, 11, 11, 5	2100, 5400, 1560, 5700, 4800, 3240, 900, 1980, 1980, 1800
10	9	1	1	10	1	6	2, 3, 1, 1, 1, 3	200, 210, 50, 60, 20, 270
11	9	1	1	10	3	6	5, 3, 3, 3, 3, 4	750, 180, 360, 630, 720, 120
12	9	1	1	10	12	6	7, 12, 19, 12, 21, 14	4200, 4320, 2280, 7200, 12600, 13440
13	9	1	1	10	18	6	36, 37, 19, 28, 46, 9	45360, 19980, 20520, 5040, 82800, 12960
14	10	0	1	10	6	6	11, 18, 16, 15, 18, 17	5940, 1080, 1920, 3600, 7560, 3060
15	5	5	1	10	6	6	13, 14, 4, 6, 6, 16	1560, 1680, 2160, 720, 2880, 5760
16	1	9	1	10	6	6	6, 11, 6, 9, 5, 17	1080, 4620, 2160, 3240, 1200, 4080
17	0	10	1	10	6	6	4, 18, 13, 13, 11, 13	2400, 7560, 7020, 780, 3960, 2340
18	9	1	0.1	10	6	6	4, 13, 7, 7, 4, 4	168, 468, 210, 126, 168, 192
19	9	1	0.5	10	6	6	7, 13, 10, 15, 4, 19	1890, 780, 1200, 4050, 1080, 1710
20	9	1	100	10	6	6	14, 4, 11, 4, 13, 15	84000, 48000, 66000, 240000, 156000, 90000
21	9	1	1000	10	6	6	11, 8, 7, 9, 16, 4	5280000, 2880000, 4200000, 3240000, 5760000, 240000

(*) c_k^{mean} and d_k are listed by ascending index: $c_1^{mean}, c_2^{mean}, \ldots$ and $d_1, d_2, \ldots$

7.2. *Solution characteristics*

For each instance, we apply both the *H_Bot* Algorithm and the ILP model (with PuLP library). We get the following characteristics for the solutions provided by the ILP:

- $H^{Max}(ILP)$ = Number of elementary robots necessary to achieve the whole process.
- $Cost(ILP) = \alpha \cdot H^{Max}(ILP) + \beta \cdot H(ILP)$, where $H(ILP)$ is the number of elementary robot trips globally performed . Thus $Cost(ILP)$ is the value of the objective function as expressed in the ILP model.

- $CPU(ILP)$ = CPU time (in seconds) required by the PuLP library in order to run the ILP model. We do not allow this CPU time to exceed 600 seconds. If this time is not enough to ensure optimality, $Cost(ILP)$ represents the upper bound computed by the library, and so we mention the gap to optimality between this upper bound and related lower bound provided by the library.

For *H_Bot* Algorithm, we get the following characteristics for the solutions:

- $H^{Max}(H\_Bot)$ = Number of elementary robots necessary to achieve the whole process according to the *H_Bot* Algorithm.
- $Cost(H\_Bot) = \alpha \cdot H^{Max}(H\_Bot) + \beta \cdot H(H\_Bot)$, where $H(H\_Bot)$ is the number of elementary robot trips globally performed by those robots according to the *H_Bot* Algorithm.
- $H^{Mean}(H\_Bot) = \frac{H(H\_Bot)}{T}$ = average number of elementary robots that are active per period.
- $Gap\_Cost(H\_Bot) = \frac{Cost(H\_Bot) - Cost(ILP)}{Cost(ILP)}$: Gap to optimality induced by the *H_Bot* algorithm.
- $Gap\_Cost(1\_Step)$ = Gap to optimality when we restrict ourselves to the first step of *H_Bot*.
- $Gap\_Fleet(H\_Bot) = \frac{H^{Max}(H\_Bot) - H^{Max}(ILP))}{H^{Max}(ILP)}$: Gap between the value H^{Max} computed by *H_Bot* and the same value computed by the ILP model.
- $Gap\_Fleet(1\_Step)$ = Same as GAP_fleet if we restrict ourselves to the first step of *H_Bot*.
- $CPU(H\_Bot)$ = CPU time (in seconds) required by the *H_Bot* Algorithm.

Note that H^{Max}/H^{Mean} represents the utilization rate of elementary robots.

7.3. *Analysis of results*

The numerical results are summarized in tables 4 and 5. In both tables, we identify the nominal instance in bold. We can make the following comments.

Global performance of the *H_Bot* Algorithm. For all instances, the computation time of the *H_Bot* algorithm is less than 0.1 seconds, while the maximum gap to optimality related to cost value is 0.71%, and the maximum gap related to fleet size is 1.49%. The *H_Bot* algorithm achieves optimality for 8 out of the 21 instances. Additionally, in the case of instance 9, *H_Bot* outperforms the ILP solver. Thus, the trade-off between accuracy and computation time clearly favors the *H_Bot* algorithm, indicating it is very efficient. It is noteworthy that the largest gaps mentioned above are associated with instance 18 and the value $\gamma = 0.1$, i.e. when low demands diminish the role of the optimal configurations $p_0(k)$, forcing the scheduler to rely on other configurations. We also observe that the running times induced by *H_Bot* are relatively stable, which is expected since *H_Bot* works as a greedy algorithm.

Impact of the second step of *H_Bot*. The second step of *H_Bot* improves the cost value obtained after the first step for 16 out of the 21 instances. However, the associated improvement margin is generally very small and never exceeds 0.75%, suggesting that most transportation tasks must rely on the optimal configurations $p_0(k)$. Not surprisingly, the second largest improvement margin corresponds to the case with

Table 4. Fleet size: In each sub-table, we vary the corresponding target parameter with respect to the bolded nominal instance ($T = 10$, $P = 6$, $K = 6$, $\alpha = 9$, $\beta = 1, \gamma = 1$)

Id	Varied parameter	H^{Max} (*ILP*)	H^{Max} (*H_Bot*)	H^{Mean} (*H_Bot*)	H^{Max} (*1_Step*)	Gap Fleet (*H_Bot*)	Gap Fleet (*1_Step*)
2	$T = 1$	796	796	796	802	0%	0%
3	$T = 5$	469	471	468.4	471	0.43%	0.43%
1	**T = 10**	**609**	**610**	**608.4**	**610**	**0.16%**	**0.16%**
4	$T = 20$	568	569	567.8	569	0.18%	0.18%
5	$T = 50$	520	520	519.8	520	0%	0%
					Average	0.15%	0.15%
6	$K = 1$	99	99	99	99	0%	0%
7	$K = 3$	422	422	422	422	0%	0%
1	**K = 6**	**609**	**610**	**608.4**	**610**	**0.16%**	**0.16%**
8	$K = 8$	1008	1009	1007.7	1009	0.11%	0.11%
9	$K = 10$	827	827	826	827	0%	0%
					Average	0.05%	0.05%
10	$P = 1$	39	39	39	39	0%	0%
11	$P = 3$	147	147	146.8	147	0%	0%
1	**P = 6**	**609**	**610**	**608.4**	**610**	**0.16%**	**0.16%**
12	$P = 12$	1871	1872	1870.2	1872	0.05%	0.05%
13	$P = 18$	5870	5873	5869.5	5873	0.05%	0.05%
					Average	0.05%	0.05%
14	$\alpha = 10$	506	507	505.8	507	0.20%	0.20%
1	$\alpha =$ **9**	**609**	**610**	**608.4**	**610**	**0.16%**	**0.16%**
15	$\alpha = 5$	568	569	568	570	0.18%	0.35%
16	$\alpha = 1$	585	586	584.9	586	0.17%	0.17%
17	$\alpha = 0$	2426	689	687.7	689	$\alpha = 0$	$\alpha = 0$
					Average	0.18%	0.22%
18	$\gamma = 0.1$	67	68	66.9	68	1.49%	1.49%
19	$\gamma = 0.5$	321	321	320.8	321	0%	0%
1	$\gamma =$ **1**	**609**	**610**	**608.4**	**610**	**0.16%**	**0.16%**
20	$\gamma = 100$	34916	34917	34915.9	34917	0.003%	0.003%
21	$\gamma = 1000$	762982	762983	762981.6	762983	0.001%	0.001%
					Average	0.33%	0.33%
					Total [(†)] ***average***	0.15%	0.16%

[(†)] The nominal instance is counted only once in the average.

Table 5. Cost and CPU time: In each sub-table, we vary the corresponding target parameter with respect to the bolded nominal instance ($T = 10$, $P = 6$, $K = 6$, $\alpha = 9$, $\beta = 1, \gamma = 1$)

Id	Varied parameter	Cost (*ILP*)(*)	Cost (*H_Bot*)	Cost (*1_Step*)	Gap Cost (*H_Bot*)	Gap Cost (*1_Step*)	CPU time (*ILP*)	CPU time (*H_Bot*)
2	$T = 1$	7960	7960	8020	0%	0.75%	0.01 s.	0.01 s.
3	$T = 5$	6563	6581	6591	0.003%	0.43%	0.09 s.	0.01 s.
1	**T = 10**	**11565**	**11574**	**11578**	**0.08%**	**0.11%**	**0.08 s.**	**0.01 s.**
4	$T = 20$	16467	16476	16478	0.05%	0.07%	9.81 s.	0.01 s.
5	$T = 50$	30668	30669	30675	0.003%	0.02%	61.45 s.	0.01 s.
				Average	0.03%	0.28%	14.29 s.	0.01 s.
6	$K = 1$	1881	1881	1881	0%	0%	0.01 s.	0.002 s.
7	$K = 3$	8018	8018	8018	0%	0%	0.12 s.	0.003 s.
1	**K = 6**	**11565**	**11574**	**11578**	**0.08%**	**0.11%**	**0.08 s.**	**0.01 s.**
8	$K = 8$	19149 (0.005%)	19158	19164	0.05%	0.09%	600 s.	0.01 s.
9	$K = 10$	15704 (0.07%)	15703	15712	-0.01%	0.05%	600 s.	0.02 s.
				Average	0.02%	0.05%	240 s.	0.01 s.
10	$P = 1$	741	741	741	0%	0%	0.01 s.	0.01 s.
11	$P = 3$	2791	2791	2793	0%	0.07%	0.01 s.	0.004 s.
1	**P = 6**	**11565**	**11574**	**11578**	**0.08%**	**0.11%**	**0.08 s.**	**0.01 s.**
12	$P = 12$	35541	35550	35568	0.03%	0.08%	0.22 s.	0.02 s.
13	$P = 18$	111525 (0.002%)	111552	111584	0.02%	0.05%	600 s.	0.07 s.
				Average	0.03%	0.06%	120 s.	0.02 s.
14	$\alpha = 10$	5060	5070	5070	0.20%	0.20%	0.09 s.	0.01 s.
1	$\alpha =$ **9**	**11565**	**11574**	**11578**	**0.08%**	**0.11%**	**0.08 s.**	**0.01 s.**
15	$\alpha = 5$	31240	31245	31275	0.02%	0.11%	15.36 s.	0.01 s.
16	$\alpha = 1$	53226 (0.014%)	53227	53290	0.002%	0.12%	600 s.	0.01 s.
17	$\alpha = 0$	68770	68770	68860	0%	0.13%	0.01 s.	0.01 s.
				Average	0.06%	0.13%	123 s.	0.01 s.
18	$\gamma = 0.1$	1272	1281	1288	0.71%	1.26%	0.02 s.	0.01 s.
19	$\gamma = 0.5$	6097	6097	6099	0%	0.03%	0.11 s.	0.01 s.
1	$\gamma =$ **1**	**11565**	**11574**	**11578**	**0.08%**	**0.11%**	**0.08 s.**	**0.01 s.**
20	$\gamma = 100$	663403	663412	663417	0.001%	0.002%	0.13 s.	0.01 s.
21	$\gamma = 1000$	14496655 (6x10^{-6} %)	14496663	14496671	0.00005%	0.00011%	600 s.	0.06 s.
				Average	0.16%	0.28%	120 s.	0.02 s.
				Total (†) average	0.06%	0.17%	147 s.	0.02 s.

(*) Values in brackets represent the gap to optimality in percentage at the time limit of 600 s.

(†) The nominal instance is counted only once in the average.

the smallest demand ($\gamma = 0.1$), which leads the scheduler to involve non-optimal configurations.

The behavior of the ILP model. The ILP model fails to ensure optimality within 600 seconds for 5 out of the 21 instances. However, it still provides a very efficient upper bound even in these cases. We observe that while the ILP software is relatively robust with respect to variations in the time parameter T, it is significantly more sensitive to variations in the number of load types K and the number of configurations P (instances 8, 9, 13). Additionally, similar cost weights α and β tend to complicate the model structure and thus challenges the ILP solver (instances 15 and 16). Finally, increasing the size of the demands (instance 21) makes it difficult for the branch-and-bound process performed by the ILP solver to accurately identify the best-fit values of the variables, thereby increasing the computational effort.

Impact of scaling parameters α and β. Since our problem may be viewed as a multi-criteria problem, with 2 different cost components H and H^{Max} that we merge in order to get a 1-criterion formulation, we must ask about the impact of the scaling parameters α and β. Table 4 shows that this effect is rather small. We may evaluate it by comparing H^{Max} with the mean value $H^{Mean} = \frac{H}{T}$ of the average number of robots involved per period, and observing the correlation between the distortion of H^{Max} with respect to H^{Mean} and the value of α. In fact H^{Mean} happens to be always very close to H^{Max}, with a gap $\frac{H^{Max}-H^{Mean}}{H^{Max}}$ that never exceeds 1%. Even though we can observe that this gap tends to slightly decrease as α increases, we may roughly say that both H^{Max} and H^{Mean} criteria tend to converge.

We conclude this section by putting the algorithms into perspective with regard to their industrial application. In practice, if the optimization process is run once a day, the computation time can be set to 10 minutes or even 1 hour. In this case, the ILP approach can be used for small to medium solutions. However for large instances, or if the **Multi-Bot** problem involves temporal constraints or more complex configurations, the ILP will be in trouble. In practice, the transportation process in an industrial warehouse is dynamic, and we have to deal with new demands and random events, that force decision makers to frequently update the planning of transportation tasks. For small instances, the ILP model can produce good solutions in less than that 2 minutes, which is the time between periods that can be used for manual reconfiguration. For larger instances, we need a fast algorithm to quickly reschedule transportation tasks. Finally, the choice between ILP and heuristic is not just a matter of running time. Relying on a sophisticated tool like a MILP library in a warehouse is expensive for an industrial player. It also raises questions about worker training and the potential inflexibility of such a tool. It follows that a heuristic approach may be useful even if we are able to significantly reduce the running times of the ILP model.

8. Conclusion

This paper studies the **Multi-Bot** problem which consists in sizing a fleet of reconfigurable robots. The transport of loads from one area to another is performed by reconfigurable mobile robots. A new configuration of robots can be formed after each period. We show that the problem is strongly NP-hard and that, in some special cases,

the problem can be solved in polynomial time. We then derive from our theoretical results an efficient heuristic algorithm for the general case. A numerical study shows that the heuristic algorithm can successfully be applied even for large instances and has very good performances on the tested instances.

We have assumed in this work that a configuration can be identified with a number of elementary robots. This number may not be sufficient to characterize a poly-robot. It would be interesting in the future to consider poly-robots whose capacity depends not only on the number of elementary robots, but also on their geometry. Another research perspective is to include reconfiguration times or costs in the model.

Data availability statement

The data supporting the findings of this study are available within the article.

Acknowledgments

This research was financed by the French government IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25) and by the Auvergne-Rhône-Alpes region under the program "Pack Ambition Recherche".

The authors would like to thank the company Mecabotix for providing us the images of Figure 1.

References

Battaïa, Olga, Xavier Delorme, Alexandre Dolgui, Johannes Hagemann, Anika Horlemann, Sergey Kovalev, and Sergey Malyutin. 2015. "Workforce minimization for a mixed-model assembly line in the automotive industry." *International Journal of Production Economics* 170: 489–500.

Beaujon, George J, and Mark A Turnquist. 1991. "A model for fleet sizing and vehicle allocation." *Transportation Science* 25 (1): 19–45.

Beşikci, Umut, Ümit Bilge, and Gündüz Ulusoy. 2015. "Multi-mode resource constrained multi-project scheduling and resource portfolio problem." *European Journal of Operational Research* 240 (1): 22–31.

Boysen, Nils, Philipp Schulze, and Armin Scholl. 2022. "Assembly line balancing: What happened in the last fifteen years?" *European Journal of Operational Research* 301 (3): 797–814.

Chaikovskaia, Mari, Jean-Philippe Gayon, Zine Elabidine Chebab, and Jean-Christophe Fauroux. 2021. "Sizing of a fleet of cooperative robots for the transport of homogeneous loads." In *2021 IEEE 17th International Conference on Automation Science and Engineering*, 1654–1659.

Chaikovskaia, Mari, Jean-Philippe Gayon, and Mairtin Marjollet. 2022. "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*, 2253–2258.

Choobineh, F. Fred, A. Asef-Vaziri, and X. Huang. 2012. "Fleet sizing of automated guided vehicles: a linear programming approach based on closed queuing networks." *International Journal of Production Research* 50 (12): 3222–3235.

Dauzère-Pérès, Stéphane, W Roux, and Jean B Lasserre. 1998. "Multi-resource shop scheduling with resource flexibility." *European Journal of Operational Research* 107 (2): 289–305.

Dolgui, Alexandre, and Jean-Marie Proth. 2013. "Assembly line balancing: Conventional methods and extensions." *IFAC Proceedings Volumes* 46 (9): 43–48.

Egbelu, Pius J. 1987. "The use of non-simulation approaches in estimating vehicle requirements in an automated guided based transport system." *Material flow* 4 (1-2): 17–32.

Etezadi, T, and JE Beasley. 1983. "Vehicle fleet composition." *Journal of the Operational Research Society* 34 (1): 87–91.

Fragapane, Giuseppe, René de Koster, Fabio Sgarbossa, and Jan Ola Strandhagen. 2021. "Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda." *European Journal of Operational Research* 294 (2): 405–426.

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

Gerhards, Patrick. 2020. "The multi-mode resource investment problem: a benchmark library and a computational study of lower and upper bounds." *Or Spectrum* 42 (4): 901–933.

Hartmann, Sönke, and Dirk Briskorn. 2022. "An updated survey of variants and extensions of the resource-constrained project scheduling problem." *European Journal of operational research* 297 (1): 1–14.

Hsu, Chih-Cheng, and David S Kim. 2005. "A new heuristic for the multi-mode resource investment problem." *Journal of the Operational Research Society* 56 (4): 406–413.

Koo, Pyung-Hoi, Jaejin Jang, and Jungdae Suh. 2004. "Estimation of part waiting time and fleet sizing in AGV systems." *International journal of flexible Manufacturing Systems* 16 (3): 211–228.

Lee, Kangbok, Lei Lei, Michael Pinedo, and Shengbin Wang. 2013. "Operations scheduling with multiple resources and transportation considerations." *International Journal of Production Research* 51 (23-24): 7071–7090.

Leus, Roel, and Willy Herroelen. 2004. "Stability and resource allocation in project planning." *IIE transactions* 36 (7): 667–682.

Liu, Changchun, Xi Xiang, Li Zheng, and Jing Ma. 2018. "An integrated model for multi-resource constrained scheduling problem considering multi-product and resource-sharing." *International Journal of Production Research* 56 (19): 6491–6511.

MecaBotiX. 2022. https://www.mecabotix.com/.

Möhring, Rolf H. 1984. "Minimizing costs of resource requirements in project networks subject to a fixed completion time." *Operations Research* 32 (1): 89–120.

Moon, Ilkyeong, Rasaratnam Logendran, and Jeonghun Lee. 2009. "Integrated assembly line balancing with resource restrictions." *International Journal of Production Research* 47 (19): 5525–5541.

Özceylan, Eren, Can B Kalayci, Aşkıner Güngör, and Surendra M Gupta. 2019. "Disassembly line balancing problem: a review of the state of the art and future directions." *International Journal of Production Research* 57 (15-16): 4805–4827.

Rjeb, Achraf, Jean-Philippe Gayon, and Sylvie Norre. 2021. "Sizing of a heterogeneous fleet of robots in a logistics warehouse." In *2021 IEEE 17th International Conference on Automation Science and Engineering*, 95–100.

Sinriech, D., and JMA Tanchoco. 1992. "An economic model for determining AGV fleet size." *International Journal of Production Research* 30 (6): 1255–1268.

Tao, Sha, and Zhijie Sasha Dong. 2018. "Multi-mode resource-constrained project scheduling problem with alternative project structures." *Computers & Industrial Engineering* 125: 333–347.

Wḱeglarz, Jan, Joanna Józefowska, Marek Mika, and Grzegorz Waligóra. 2011. "Project scheduling with finite or infinite number of activity processing modes–A survey." *European Journal of operational research* 208 (3): 177–205.

Appendix A. Proof of Lemma 1

Let x be a feasible solution and let us suppose that, for some (t, k) and for some $p_1 \neq p_0(k)$ we may write: $p_0(k) = u{\cdot}GCD(p_0(k), p_1); p_1 = v{\cdot}GCD(p_0(k), p_1); x^t_{p_1,k} \geq u$.

Then we increase $x^t_{p_0(k),k}$ by v and decrease $x^t_{p_1,k}$ by u. Doing this maintains the value (2) of the solution since $u \cdot p_1 = v \cdot p_0$. But the inequality $p_0(k) \cdot c_{p_1,k} \leq p_1 \cdot c_{p_0(k),k}$ also keeps the quantity $\sum_{l,p} c_{pk} x^l_{pk}$ from decreasing and so (3) keeps holding. We can do this until (9) becomes satisfied and so we conclude.

Appendix B. Strong NP-hardness of Multi-Bot

Our proof comes in 2 steps.

Step 1: Turning Multi-Bot into a satisfiability problem with same complexity

We get a feasible **Multi-Bot** solution by setting $x^t_{pk} = d_k$ for any t, p, k. If we set $d^{Max} = \sup_k d_k$ then we deduce that optimal cost value of **Multi-Bot** is bounded by $2d^{Max} \cdot \sup(\alpha, \beta) \cdot K \cdot P \cdot T$. It comes that solving **Multi-Bot** may be achieved by successively solving (binary search process) no more than $\lceil \log_2(2d^{Max} \cdot \sup(\alpha, \beta) \cdot K \cdot P \cdot T) \rceil$ instances of the following decision problem **Multi-Bot-Dec**(S), $S \leq 2d^{Max} \cdot \sup(\alpha, \beta) \cdot K \cdot P \cdot T$ being a threshold parameter.

Multi-Bot-Dec(S): Compute non negative integral vector $x = (x^t_{pk}, k \in \mathcal{K}, p \in \mathcal{P}, t \in \mathcal{T})$ such that:

$$\forall k, \qquad \sum_{t,p} c_{pk} \cdot x^t_{pk} \geq d_k \tag{B1}$$

$$\alpha \cdot (\max_t \sum_{k,p} p \cdot x^t_{pk}) + \beta \sum_{t,p,k} p \cdot x^t_{pk} \leq S \tag{B2}$$

Since $\lceil \log_2(2d^{Max} \cdot \sup(\alpha, \beta) \cdot K \cdot P \cdot T) \rceil$ may be bounded by a polynomial function of $Size(Multi\text{-}Bot) = P + T + K + \sum_{k,p}(1 + \lceil \log_2 c_{pk} \rceil) + \sum_k (1 + \lceil \log_2 d_k \rceil) + (1 + \log_2 \alpha) + (1 + \log_2 \beta)$, we must only check that **Multi-Bot-Dec**(S) is strongly NP-Complete.

Step 2: Multi-Bot-Dec(S) is strongly NP-Complete

Its ILP formulation shows that **Multi-Bot-Dec**(S) is in NP. So what remains to check is that, as suggested at the beginning of the section, the bin packing problem can be polynomialy reduced to the **Multi-Bot-Dec** problem (Garey and Johnson 1979), or, in other words, that the bin packing problem may be viewed as a specific case of **Multi-Bot-Dec.** Let us recall that a bin packing instance BP is characterized by:

- A set $\mathbf{I} = \{1, \ldots, I\}$ of items such that for any item $i \in \mathbf{I}$, is provided with a weight w_i;
- A set $\mathbf{B} = \{1, \ldots, B\}$ of identical boxes, all with a same capacity ρ.

Solving BP means computing an assignment σ from $\mathbf{I}$ to $\mathbf{B}$, consistent of the capacities of the boxes, that means such that for any $b \in \mathbf{B}$, $\sum_{i \ s.t. \ \sigma(i)=b} w_i \leq \rho$.
We know that the bin packing problem is strongly NP-Complete: Strongly means that there exists a polynomial function Q of I and B such that if we restrict ourselves to

instances BP such that weights w_i and the capacity ρ are bounded by $Q(I, B)$, then resulting problem remains NP-Complete (Garey and Johnson 1979). In order to prove that **Multi-Bot-Dec** is strongly NP-Complete, we only need to design a polynomial-time algorithm *Code* which turns any bin packing instance BP into a **Multi-Bot-Dec** instance *Multi-Bot-Dec* = *Code*(BP), and a polynomial-time algorithm *Decode* which turns any **Multi-Bot-Dec** output x into a bin packing output σ in such a way that: BP admits a feasible solution σ if and only if *Code*(BP) admits a feasible solution x such that *Decode*(x) = σ. Let us start from such a bin packing instance BP such that number ρ and coefficients w_i are all bounded by $Q(I, B)$, with polynomial function Q as above. We notice that if we add $B \cdot \rho - \sum_i w_i$ items with weight 1 then we do not modify the feasibility of BP. So we may suppose that BP is such that: $B \cdot \rho = \sum_i w_i$. Then we derive from BP a **Multi-Bot-Dec** instance *Multi-Bot-Dec* as follows:

- We set $T = B$: that means that we identify any box b with a period, i.e. an index value t of *Multi-Bot-Dec*.
- We set $\alpha = 1$ and $\beta = 0$.
- We set K = Number of distinct values w_i involved into vector w. We identify any existing weight w_i with an index value k of *Multi-Bot-Dec*. K may be interpreted as an item type, characterized by its weight $w(k)$. For instance, if we have 5 items with respective weights $w_1 = 2, w_2 = 6, w_3 = 3, w_4 = 2$ and $w_5 = 6$, then we get $K = \{1, 2, 3\}$, with $w(1) = 2, w(2) = 6, w(3) = 3$.
- We set $P = \sup_i w_i$. We identify any possible weight w_i with an index value p of *Multi-Bot-Dec*.
- According to this, we set: $c_{pk} = w(k)$ if $p = w(k)$ and $c_{pk} = 0$ else.
- For any k, we set $d_k = w(k) \cdot R_k$, where R_k denotes the number of items i with weight $w_i = w(k)$.
- Finally we set $S = \rho$.

According to this, x^t_{pk} refers to the number of items with weight p and item type k which are assigned to box t. Constraint (B1) means that all items should be assigned to some box t. Every sum $H_t = \sum_{p,k} p x^t_{pk}$ should be equal to ρ.

If σ is a feasible solution of the bin packing instance BP then we see that we get a feasible solution x of the instance *Multi-Bot-Dec* by setting: $x^t_{w(k),k}$ = number of items i with weight $w(k)$ assigned to box t, and $x^t_{pk} = 0$ if $w(k) \neq p$. Conversely, if x is a feasible solution of the instance *Multi-Bot-Dec*, then we get a solution σ of the instance BP by assigning $x^t_{w(k),k}$ items with weight $w(k)$ to box t. The *Code* procedure which computes T, P, K together with coefficients S and $d_k, k \in K$, clearly works in polynomial time as a function of $Q(I, B)$, I and B and thus also as a function of I and B. The *Decode* procedure which retrieves assignment σ from x works the same way. So we conclude the proof.

The **Multi-Bot-Dec** instance *Multi-Bot-Dec* involved in the above reduction process is specific in the sense that, for any load type k, only one configuration p may be applied to k. This configuration becomes in an obvious way the most profitable configuration $p_0(k)$. This means that even if we restrict **Multi-Bot** to the case where, for any load type k, only one configuration $p = p_0(k)$ fits with k, then **Multi-Bot** remains NP-Hard.

Appendix C. Proof of Theorem 2

Case $P = 1$

Assume that $P = 1$. In this case, we use exclusively 1-bots. Since it is optimal to use 1-bots to their maximum capacity, the number of elementary robot trips is $\left\lceil \frac{d_k}{c_k} \right\rceil$ for loads of type k in the optimal solution. It follows that the optimal number of robots is

$$H^{max} = \left\lceil \frac{\sum_{k \in \mathcal{K}} \left\lceil \frac{d_k}{c_k} \right\rceil}{T} \right\rceil .$$

The transportation tasks can then be assigned to the different periods, while not exceeding H^{max} transportation tasks per period. We conclude.

Case $T = 1$

Assume that $T = 1$. Let us simplify x^t_{pk} as x_{pk}. We have $H^{Max} = H = H$ and the objective function (1) reduces to $(\alpha + \beta)H^{Max}$. So, we simply want to minimize the number of elementary robots in a single period. **Multi-Bot** can be decomposed into K independent sub-problems $\mathbf{MB}_k$ with $k \in \mathcal{K}$.

$$\begin{aligned} \mathbf{MB}_k \quad & \min \sum_{p \in \mathcal{P}} p \cdot x_{pk} \\ & \text{subject to :} \\ & \sum_{p \in \mathcal{P}} c_{pk} \cdot x_{pk} \geq d_k \\ & x_{pk} \in \mathbb{N}, \forall p \in \mathcal{P} \end{aligned}$$

So we only need to check that $\mathbf{MB}_k$ can be solved in polynomial time. Lemma 1 tells that every values $x_{p,k}, p \neq p_0(k)$ is bounded by P, and so that $\sum_{p \neq p_0(k)} p \cdot x_{p,k} \leq P^2$. The pseudo-polynomiality of the Knapsack-like problem makes that for every value $H \leq P^2$ of $\sum_{p \neq p_0(k)} p \cdot x_{p,k}$, we are able to compute in polynomial time the maximal value $V(H)$ of $\sum_{p \neq p_0(k)} c_{p,k} \cdot x_{p,k}$ and thus derive the smallest value $x_{p_0(k),k} = \left\lceil \frac{d_k - V(H)}{c_{p_0 k,k}} \right\rceil$ that will make x meet the demand constraint. We conclude.

Case $K = 1$

Assume that $K = 1$. Let us simplify $x^t_{p,k}$ as x^t_p, $p_0(k)$ as p_0 and d_k as d, let us consider some feasible solution x of **Multi-Bot** and let us now denote by Min the smallest value $\sum_p p \cdot x^t_p, t = 1, \ldots, T$. Let us denote by S the smallest value $\sum_p p \cdot y_p$ that makes possible to get $\sum_p c_p \cdot y_p \geq d$. We saw, while enhancing the **Multi-Bot** MILP program, that we may impose, for any t: $(\sum_p p \cdot x^t_p) – Min \leq P$ (*Bounded Difference Constraint*). Let us now denote by S the smallest value $\sum_p p \cdot y_p$ that makes possible to get $\sum_p c_p \cdot y_p \geq d$. Then the *Bounded Difference Constraint* tells us that $(T-1).(Min + P) + Min$ must be at least equal to S, which means $Min \geq$

$\frac{S-(T-1)\cdot P}{T}$. Also, decreasing some value x_p^t will violate the feasibility of x. We deduce that $T \cdot Min - P < S$ and so that $Min \leq \frac{(P+S-1)}{T}$. Now we see that computing S means solving a Knapsack problem of the same form as the $\mathbf{MB}_k$ involved in the proof of Theorem 1, and so that it may be done in polynomial time. We also see that the number integral values between $\frac{S-(T-1)\cdot P}{T}$ and $\frac{(P+S-1)}{T}$ may be bounded by a polynomial function of the encoding size of **Multi-Bot**. Thus, we may apply a DP scheme involving the set $\{1, \ldots, T\}$ as time space, the values $\sum_p p \cdot (x_p^t - \frac{(S-(T-1))\cdot P}{T})$ as decisions, and, for any t_0, the cumulative values $\sum_{p,t<t_0} p \cdot (x_p^t - \frac{(S-(T-1))\cdot P}{T})$ as states at the time value t_0. Both decision and state spaces have polynomial size. Then we become able to conclude if we can prove that any decision $\sum_p p \cdot (x_p^t - \frac{(S-(T-1))\cdot P}{T})$ characterizes a decision vector $x^t = (x_p^t, p = 1, \ldots, P)$, that can be retrieved in polynomial time. Once again, we proceed as in the proof of Theorem 1 in order to get that, for any integral value H between $\frac{S-(T-1)\cdot P}{T}$ and $\frac{(P+S-1)}{T}$, the Knapsack problem induced by imposing a non negative integral vector $y = (y_p, p = 1, \ldots, P)$ to be such that $\sum_p p \cdot y_p = H$ and $\sum_p c_{p,k} \cdot y_p$ is maximal, can be solved in polynomial time, its optimal solution being stored in some table. We conclude.