Journal of Scheduling (2023) 26:113–136
https://doi.org/10.1007/s10951-022-00755-5

Scheduling a pick and place packaging line

Tommaso Schettini[1] · Federico Malucelli[1] · Helena Ramalhinho[2]

Accepted: 1 August 2022 / Published online: 18 October 2022

Abstract
In this paper, we introduce the Pick and Place Packaging Problem (P4), for optimally scheduling a packaging system with one input conveyor (pick) and one output conveyor (place). We give a formal definition of the underlying dynamic optimization problem. We describe two properties that hold for its solutions and present an efficient row generation approach exploiting these properties. Starting from a basic version of the problem, we introduce two variants where we account for the possibility of holding the grip of items and variable conveyor speed. We extend the proposed exact solution method to these two cases. Then, we present an efficient Iterated Local Search heuristic for the problem and its variants. Numerical results show the effectiveness of our approach.

Keywords Optimization · Rolling horizon · Packaging · Pick and place · Row generation · Iterated local search

1 Introduction and state of the art

We consider the problem of scheduling the operations of a Pick and Place system in a Packaging line. Pick and Place systems consist of a set of robotic arms and one or more conveyors. One of the conveyors serves as the input of the system; thus, we refer to it as the "input conveyor." The robots are tasked to remove the items flowing on the input conveyor and place them in a designated destination. All items that reach the end of the conveyor without being picked up are lost. A dynamic example of this type of systems can be watched on the site:

https://www.youtube.com/watch?v=PrZw-xp_VzE

In this paper, we consider the case of a Pick and Place line with two conveyors, one input conveyor and one output conveyor. The robots have to place the input items in some containers in the output conveyor. The items are identical, and the containers have a limited capacity. The operation of picking up one item from the input conveyor and placing it to the final destination is called *mission*. The problem that we study here involves making decisions about the scheduling of the missions and, in the extension of the problem, also about the conveyor's speed. The objective function consists of the minimization of the number of items that are not picked up. In such problems, efficiency is of paramount importance. Utilizing the robots to their fullest potential is essential due to their high cost.

We assume that the position of the incoming items is known in advance. Generally, in these systems, high-definition cameras monitor the inbound conveyors shortly before the items enter within the reach of the robots and provide information of the location of all incoming items. Typically, these cameras cover a few meters before the reach of the first robot. The position of the input items is usually known a very short time in advance (i.e., 5–10 seconds). Thus, the scheduling of the robots has to be decided in a timely manner. For this reason, the majority of methods proposed for this kind of system implement simple dispatching strategies, and the most commonly used approaches in the industry use queuing mechanisms. These techniques perform rather well when the system reaches the steady-state conditions. However, in systems with frequent transients and irregularities in their operating conditions, this kind of strategy often falls short, and the performance gets signifi-

✉ Tommaso Schettini
tommaso.schettini@polimi.it

Federico Malucelli
federico.malucelli@polimi.it

Helena Ramalhinho
helena.ramalhinho@upf.edu

1 DEIB, Politecnico di Milano, Via Ponzio 34/5, 20133 Milan, Italy

2 Department of Economics and Business, Universitat Pompeu Fabra, Carrer de Ramon Trias Fargas, 25-27, 08005 Barcelona, Spain

cantly deteriorated. The alternative to these strategies is to repeatedly re-optimize the scheduling of the line as more information becomes available. This is generally referred to as a rolling horizon approach, where the scheduling periodically is recomputed for the next planning horizon. Then, when the computed scheduling is applied, the scheduling of the next period is solved.

Most papers on the optimization of Pick and Place systems concern themselves with the optimization of single conveyor systems and propose techniques that do not generalize well in the two-conveyor case. In these cases, the optimization effort considers maximizing the system throughput and minimizing the wear. This comes in two main forms:

- The load balancing of the various machines, so that no machine is ever overworked;
- The minimization of the robot's travel time, by optimizing of the mission starting time.

In their papers, Mattone et al. (1998) and Mattone et al. (2000) presented augmented versions of the commonly used FIFO and LIFO dispatching rules for a single conveyor Pick and Place system. A more advanced approach is proposed by Huang et al. (2012), it combines heuristic dispatching rules, to obtain a performant and fast approach. Humbert et al. (2015) study the performance of several approaches mostly focusing on the various developed dispatching rules in an experimental setting. Huang et al. (2015) present part dispatching rules in a task-based environment.

Bozma and Kalalıoğlu (2012) propose a coordination approach based on cost-driven policies. These advanced methods allowed to directly pursue more elaborate objectives, and the heuristic and metaheuristic approaches ensure fast computations. For the minimization of robot travel times on a simple conveyor system (Daoud et al. 2010) concentrated on the computation speed using a metaheuristic approach.

Ahmadi and Kouvelis (1994) put their attention on a different aspect of the problem, trying to balance the usage of the different robots on the line as to minimize their wear. The energy optimization and the minimization of the damage of the components are studied by Pellicciari et al. (2013).

Ho and Ji (2009) focus on the stationary problem that is solved exactly to minimize the traveled distance of the robots, as in a sequence-dependent scheduling problem.

The problem of scheduling two conveyors however, to the best of our knowledge, has received little interest over the years. Despite this, the problem poses interesting challenges, both on the optimization of the missions of the robots themselves, and on the synchronization of the various robots acting on the system.

For the problem of controlling a packaging line for perishable Ferrari et al. (2015) and Pizzi et al. (2016) proposed the idea of using a rolling horizon approach paired with an exact model to dynamically compute the scheduling of the robots and the conveyor's speed. They were able to show significant benefits for the control of the line over the commonly used dispatching strategies. This approach is very promising, though it can be applied only to very small instances, due to the cumbersome scheduling computation.

In this paper, we give a formal definition of the scheduling problem proposed in Ferrari et al. (2015). We will refer to this problem as the Pick and Place Packaging Problem (P4). We will present an exact solution method for a basic version of the problem. This approach will be extended to consider the case when the robot can hold the grip of items, and the case when the conveyor speed can be varied. In order to cope with real instances in a manageable time that is compatible with a rolling horizon approach, we will propose an efficient heuristic based on an Iterated Local Search framework. We will test the developed algorithms on a set of instances based on three realistic lines to validate the developed solution method and Iterated Local Search heuristic.

The rest of the paper is organized as follows: in Sect. 2, we give the formal description of P4 and define two theoretical properties of the problem that can be exploited in its solution. We propose an efficient solution method based on row generation in Sect. 3. Then, we deal with the extensions of the problem to the case where robots can hold an item before moving it to the containers (Sect. 4.1) and to the case where the speed of one of the conveyors is controllable (Sect. 4.2). Afterward, we develop a heuristic method based on Iterated Local Search in Sect. 5. In Sect. 6, we present the conducted computational experiments on the problem. Lastly, in Sect. 7 we draw our conclusions and discuss future research.

2 Problem details

The Pick and Place Packaging Problem (P4) considers a system with several robots and two conveyors, denoted as the " Input " and the " Output " conveyors. In Fig. 1, we show a schematic view of one possible line. The two conveyors are parallel and move in the same direction. However, the methods presented here can be extended to systems with any configuration since they not require specifically this assumption. The velocity of the conveyors is fixed. In Sect. 4.2, we relax this assumption and allow our method to control the velocity profile of one of the conveyors.

The items to be handled by the system enter the line through the Input conveyor. The task of the system is to pick up those items and move them in one of the containers on the output conveyor. All items are identical, aside from their position on the input conveyor. We consider the items partitioned into *columns*, i.e., groups of items occupying the same longitudinal position on the conveyor. Let I be the set of the

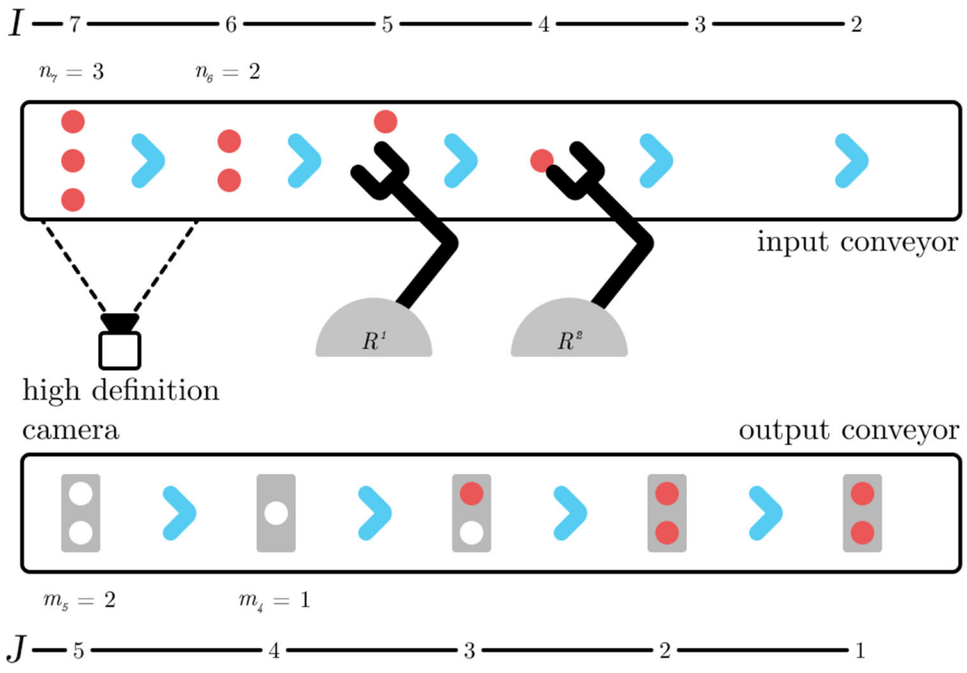

Fig. 1 Example of a pick and place line

columns $I = \{1, \ldots, n\}$. We denote by n_i the number of items in column i at the beginning of the planning horizon before any item is picked by the robots.

Let C be the set of the containers on the output conveyor $C = \{1, \ldots, m\}$. We denote by m_j the maximum capacity of container j at the start of the planning horizon. For the sake of brevity, we will use " item i " as a short-hand to refer to an item of column i, with $i \in I$, and " slot j " as a short-hand to refer to one of the empty spots in container $j \in C$.

The operation of moving the items to their destination is handled by the robots that are present along the line. We denote as R the set of robots. The robots are identical, single-grip robots, with non-overlapping working areas. Thus, the robots can move one item at a time, and the operations of each robot do not interfere with the other robots on a mechanical level.

A single pick-and-place operation performed by the robots is called " mission ". We use the tuple (i, j, r) to identify the mission where robot $r \in R$ picks up an item from column $i \in I$ and puts it into container $j \in C$.

A solution of the P4 involves three levels of decisions: (1) determining the set of missions performed by the robots, (2) determining the order in which the missions are performed, and (3) assigning a starting time to each of those missions.

Let us define an " assignment " A as the set of missions of the solution, and $A^r \subseteq A$ as the assignment of robot r, i.e., the set of missions assigned to r in A. We denote as $A^r_{i,j}$ the number of missions (i, j, r) in A, $\forall i \in I, \forall j \in C, \forall r \in R$. An assignment is valid, if:

$$\sum_{r \in R} \sum_{i \in I} A^r_{i,j} \leq m_j \qquad \forall j \in C$$

$$\sum_{r \in R} \sum_{j \in C} A^r_{i,j} \leq n_i \qquad \forall i \in I$$

Given A^r, we define a " sequencing " S^r_A as the ordered sets of missions performed by each robot r. We denote by $S^r_A[p]$ the p-th mission of S^r_A.

Additionally, given A, we define a " scheduling " T_A as a set of starting times $t_{(i,j,r)}$ of each mission $(i, j, r) \in A$, and let T^r_A denote the scheduling of A^r. We note that a scheduling of a given assignment implicitly entails a sequencing for the same assignment. When a sequencing S^r_A corresponds to the sequencing entailed in a scheduling T^r_A, we say that T^r_A "matches" S^r_A.

When A contains multiple copies of the same mission (i, j, r) (i.e., $A^r_{i,j} \geq 2$), $(i, j, r)^p$ indicates the p-th mission (i, j, r) in A. We denote by $t^p_{(i,j,r)}$ the starting time of mission $(i, j, r)^p$.

The problem input is given by the following data, for each robot r, column i and container j:

- a^r_i and b^r_i are the earliest and latest time at which item i can be picked up by robot r,
- α^r_j and β^r_j are the earliest and latest time at which one item can be placed in container j by robot r.

We assume that if $a^r_i < a^r_{i'}$ then $b^r_i < b^r_{i'}$, and if $\alpha^r_j < \alpha^r_{j'}$ then $\beta^r_j < \beta^r_{j'}$. Without loss of generality, we assume that the indices of I and C are in order of earliest starting time. Thus, $\forall i, i' \in I : i > i' \Rightarrow b^r_i > b^r_{i'}$ and $a^r_i > a^r_{i'}$ $\forall r \in R$, and $\forall j, j' \in C : j > j' \Rightarrow \beta^r_j > \beta^r_{j'}$ and $\alpha^r_j > \alpha^r_{j'}$ $\forall r \in R$. A mission (i, j, r) is feasible if its starting time falls within its availability window, i.e.,

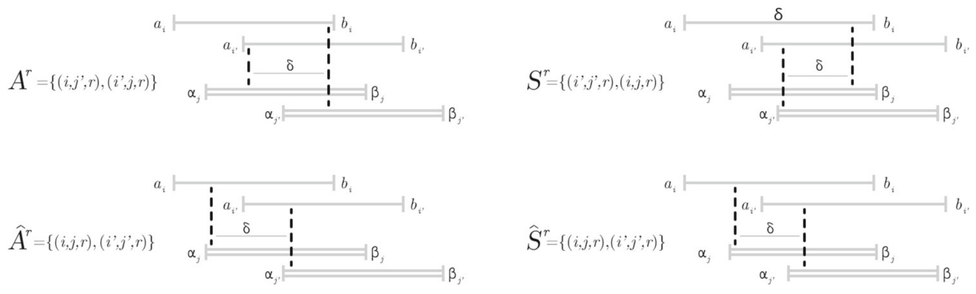
$A^r = \{(i,j',r),(i',j,r)\}$
$S^r = \{(i',j',r),(i,j,r)\}$
$\hat{A}^r = \{(i,j,r),(i',j',r)\}$
$\hat{S}^r = \{(i,j,r),(i',j',r)\}$
a_i
b_i
$a_{i'}$
$b_{i'}$
α_j
β_j
$\alpha_{j'}$
$\beta_{j'}$
δ

Fig. 2 Examples of the assignments and sequencing induced by properties 1(left figure) and 2(right figure)

$$\max(a_i^r, \alpha_j^r) \leq t_{(i,j,r)} \leq \min(b_i^r, \beta_j^r). \tag{1}$$

We initially assume that the time for performing a mission is constant and requires δ units of time. This implies that considering mission (i, j, r), if there is no intersection between the availability windows of item i and slot j for robot r, the mission is infeasible. However, in some systems, the robot may pick up an item and hold it until the destination slot becomes reachable. This would extend the set of possible feasible solutions but will introduce a variable mission time $\delta_{(i,j,r)}$. For the sake of simplicity, we discuss our exact solution method for the simple case with constant mission time to extend it afterward to the most general case.

Let E^r denote the set of the possible missions of robot $r \in R$, i.e., the set of missions (i, j, r) with $i \in I$, $j \in C$, such that $\max(a_i^r, \alpha_j^r) \leq \min(b_i^r, \beta_j^r)$.

A scheduling T_A^r is feasible if all the missions in A^r are feasible, and the starting times any two distinct missions $(i, j, r)^p, (i', j', r)^{p'} \in A^r$ are separated by at least the time needed to perform the first mission, i.e.,

$$\begin{aligned} &t^p_{(i,j,r)} - t^{p'}_{(i',j',r)} \geq \delta && \text{if } \; t^p_{(i,j,r)} > t^{p'}_{(i',j',r)} \\ &t^{p'}_{(i',j',r)} - t^p_{(i,j,r)} \geq \delta && \text{if } \; t^p_{(i,j,r)} < t^{p'}_{(i',j',r)} \\ &\quad \forall (i,j,r)^p, (i',j',r)^{p'} \in A^r : (i,j,r)^p \neq (i',j',r)^{p'} \end{aligned} \tag{2}$$

If there exists a feasible scheduling T_A^r for assignment A^r, we say that the assignment A^r is feasible. Similarly, if there exists a feasible scheduling T_A^r that matches a sequencing S_A^r, we say that S_A^r is feasible. Lastly, if an assignment A is feasible for all robots, we say that it is feasible.

If an item is not picked up by any mission in the assignment, the item is lost. The number of lost items is given by:

$$\sum_{i \in I} \left(n_i - \sum_{r \in R} \sum_{j \in C} A^r_{i,j} \right)$$

The P4 problem consists of finding an assignment A and a feasible scheduling T_A, such that the total number of lost items is minimized. An alternative objective function is minimizing the number of lost slots, i.e., slots that are not filled by any mission in the assignment.

2.1 Properties of the solutions

The solution methods that we propose are based on two properties.

Property 1 Let A^r be an assignment for robot r, such that $(i, j', r), (i', j, r) \in A^r$ with $i \leq i'$ and $j \leq j'$. If A^r is feasible, then assignment $\hat{A}^r = A^r \backslash \{(i, j', r), (i', j, r)\} \cup \{(i, j, r), (i', j', r)\}$ is also feasible.

Proof Assume without loss of generality that robot r performs the mission (i, j', r) before (i', j, r) (see Fig. 2). Additionally, for simplicity, assume that missions (i, j', r) and (i', j, r) are performed immediately one after the other, with no idle time.

Given that A^r is feasible, it holds:

$$\max(a_i^r, \alpha_{j'}^r) \leq t_{(i,j',r)} \leq \min(b_i^r, \beta_{j'}^r)$$
$$\max(a_{i'}^r, \alpha_j^r) \leq t_{(i',j,r)} \leq min(b_{i'}^r, \beta_j^r)$$

Where $t_{(i',j,r)} = t_{(i,j',r)} + \delta$.

On the other hand, $\hat{A}^r$ is feasible if:

$$\max(a_i^r, \alpha_j^r) \leq t_{(i,j,r)} \leq \min(b_i^r, \beta_j^r)$$
$$\max(a_{i'}^r, \alpha_{j'}^r) \leq t_{(i',j',r)} \leq min(b_{i'}^r, \beta_{j'}^r)$$

where $t_{(i',j',r)} = t_{(i,j,r)} + \delta$.

We can then rewrite the conditions that must hold for the feasibility of A^r as:

$$t_{(i',j,r)} \geq \max(a_i^r, a_{j'}^r + \delta, \alpha_j^r + \delta, \alpha_{j'}^r) \tag{3}$$
$$t_{(i',j,r)} \leq \min(b_i^r, b_{i'}^r + \delta, \beta_j^r + \delta, \beta_{j'}^r) \tag{4}$$

Similarly, $\hat{A}^r$ is feasible if:

$$t_{(i',j',r)} \geq \max(a_i^r + \delta, a_{i'}^r, \alpha_j^r + \delta, \alpha_{j'}^r) \tag{5}$$
$$t_{(i',j',r)} \leq \min(b_i^r + \delta, b_{j'}^r, \beta_j^r + \delta, \beta_{j'}^r) \tag{6}$$

Recalling that $b_i^r < b_{i'}^r, a_i^r < a_{i'}^r, \beta_j^r < \beta_{j'}^r$, and $\alpha_j^r < \alpha_{j'}^r$; conditions (3)–(4) imply conditions (5)–(6). □

Definition 1 An assignment A^r having no two missions (i, j', r), (i', j, r) with $i < i'$ and $j < j'$, is" 1-irreducible ".

Corollary 1 *Let A^r be a 1-irreducible assignment for robot r, if A^r is not feasible then the assignment:*

$$\bar{A}^r = A^r \setminus \{(i', j', r), (i, j, r)\} \cup \{(i, j', r), (i', j, r)\}$$

is also not feasible $\forall (i', j', r), (i, j, r) \in A^r$.

Property 2 Let A^r be a 1-irreducible assignment for robot r, such that $(i, j, r), (i', j', r) \in A^r$, with $i \le i'$ and $j \le j'$, and let there be a feasible sequencing S^r_A such that (i', j', r) precedes (i, j, r). Then, there exists another feasible sequencing $\hat{S}^r_A$ of A^r such that (i, j, r) precedes (i', j', r).

Proof Let us denote by T^r_A a feasible scheduling that matches S^r_A. Additionally, suppose that $\hat{T}^r_A$ is a scheduling that matches $\hat{S}^r_A$ where missions are performed as early as possible. We now show that if T^r_A is feasible, $\hat{T}^r_A$ is also feasible.

Without loss of generality, let us assume that in S^r_A and $\hat{S}^r_A$ all missions are performed as early as possible. Lastly, for simplicity, let us assume that robot r is idle before performing missions (i, j, r) and (i', j', r) (see Fig. 2). Under these assumptions, we can write the starting time of the two missions (i', j', r) and (i, j, r) in the two schedulings. In case T^r_A, it holds $t_{(i',j',r)} = \max(a^r_{i'}, \alpha^r_{j'})$, and $t_{(i,j,r)} = \max(t_{(i',j',r)} + \delta, a^r_i, \alpha^r_j)$. We explicitly write the feasibility conditions of T^r_A as such:

$$\max(a^r_{i'}, \alpha^r_{j'}) \le \min(b^r_{i'}, \beta^r_{j'}) \tag{7}$$

$$\max(a^r_{i}, \alpha^r_{j}) \le \min(b^r_{i}, \beta^r_{j}) \tag{8}$$

$$\max(a^r_{i'}, \alpha^r_{j'}) + \delta \le \min(b^r_{i}, \beta^r_{j}) \tag{9}$$

For scheduling $\hat{T}^r_A$, it holds $t_{(i,j,r)} = \max(a^r_i, \alpha^r_j)$ and $t_{(i',j',r)} = \max(t_{(i,j,r)} + \delta, a^r_{i'}, \alpha^r_{j'})$, Consequently, $\hat{T}_{A^r}$ is feasible if:

$$\max(a^r_{i'}, \alpha^r_{j'}) \le \min(b^r_{i'}, \beta^r_{j'}) \tag{10}$$

$$\max(a^r_{i}, \alpha^r_{j}) \le \min(b^r_{i}, \beta^r_{j}) \tag{11}$$

$$\max(a^r_{i}, \alpha^r_{j}) + \delta \le \min(b^r_{i'}, \beta^r_{j'}) \tag{12}$$

Conditions (7)–(8) and (10)–(11) coincide. Conversely, condition (9) implies (12). Thus, if S^r_A is feasible, $\hat{S}^r_A$ is also feasible. □

Definition 2 A sequencing S^r_A is "2-irreducible" if there are no two missions (i, j, r), $(i', j', r) \in A^r$, with $i \le i'$ and $j \le j'$, at least one of which holds strictly, such that (i, j, r) follows (i', j', r).

Corollary 2 *If a 1-irreducible Assignment A^r does not admit a feasible 2-irreducible sequencing, then A^r is infeasible.*

3 An exact method based on a row generation framework

We now present a MILP formulation for the P4. We express using variable $x^r_{i,j} \in Z^+$ the number of times mission (i, j, r) is performed in the scheduling, i.e., $x^r_{i,j} = A^r_{i,j}$. Binary variable $y^r_{i,j}$ is one if $x^r_{i,j} > 0$; and to zero otherwise.

Variable $\Pi^r_{i,j}$ is the starting time of mission the first mission (i, j, r) performed in the schedule, i.e., $= \Pi^r_{i,j} = t^1_{(i,j,r)}$. If no mission (i, j, r) is performed (i.e., $y^r_{i,j} = 0$), variable $\Pi^r_{i,j}$ is free to take any value. As for Corollary 2, without loss of generality, we assume that when multiple missions (i, j, r) are scheduled, those missions are performed consecutively starting at time $\Pi^r_{i,j}$. Thus, $t^p_{(i,j,r)} = \Pi^r_{i,j} + (p-1)\delta$. Lastly, variable λ_i is the number of items in column $i \in I$ that are not assigned to any mission (i.e., the number of items lost). The formulation is as follows.

$$\min \sum_{i \in I} \lambda_i \tag{13}$$

s.t.

$$\max(a^r_i, \alpha^r_j) \le \Pi^r_{i,j} \qquad \forall i \in I, j \in C, r \in R \tag{14}$$

$$\Pi^r_{i,j} \le \min(b^r_i, \beta^r_j) - \delta(1 - x^r_{i,j}) + M(1 - y^r_{i,j}) \qquad \forall i \in I, j \in C, r \in R \tag{15}$$

$$y^r_{i,j} + y^r_{i',j'} \le 1 \qquad \forall i, i' \in I, j, j' \in C, r \in R, i' > i, j' \le j \tag{16}$$

$$\Pi^r_{i,j} + \delta x^r_{i,j} \le \Pi^r_{i',j'} + M(1 - y^r_{i,j}) \qquad \forall i, i' \in I, j, j' \in C, r \in R, i' > i, j' \ge j \tag{17}$$

$$\sum_{j \in C}\sum_{r \in R} x^r_{i,j} = n_i - \lambda_i \qquad \forall i \in I \tag{18}$$

$$\sum_{i \in I}\sum_{r \in R} x^r_{i,j} \le m_j \qquad \forall j \in C \tag{19}$$

$$x^r_{i,j} \le n_i y^r_{i,j} \qquad \forall i \in I, j \in C, r \in R \tag{20}$$

$$x^r_{i,j}, \lambda_i \in Z^+ \qquad \forall i \in I, j \in C, r \in R \tag{21}$$

$$y^r_{i,j} \in \{0, 1\} \qquad \forall i \in I, j \in C, r \in R \tag{22}$$

The objective function (13) minimizes the number of items lost. Constraints (14)–(15) ensure the feasibility of all scheduled missions with respect to the availability of items and containers. Constraints (16) guarantee that the assignment is 1-irreducible. Constraints (17) prevent the robots from performing more than one mission at a time. In those constraints,

M is an arbitrarily large constant. Constraints (18) and (19) enforce the availability of items and the capacity of the containers. Constraints (20) link the $y^r_{i,j}$ and $x^r_{i,j}$ variables. Lastly, constraints (21) and (22) are the domain constraints. Note that if no mission (i, j, r) is performed (i.e., $y^r_{i,j} = 0$), the value of $\Pi^r_{i,j}$ has no significance to the solution. In this case, constraints (14)–(15), and (17) can always be satisfied by setting $\Pi^r_{i,j} = \max(a^r_i, \alpha^r_j)$.

We develop a dynamic constraint generation algorithm to solve the formulation. Specifically, our algorithm can be summarized in the following steps:

1. Generate and solve a Master Problem (MP) to obtain an initial assignment A.
2. Apply property 1 to the initial assignment to obtain an equivalent 1-irreducible assignment $\hat{A}$ and compute a 2-irreducible sequencing $S_{\hat{A}}$.
3. Generate $T_{\hat{A}}$ and verify its feasibility.
4. If $T_{\hat{A}}$ is feasible, return $\hat{A}$ and $T_{\hat{A}}$. Otherwise, separate one or more violated cut for the MP and iterate the procedure.

All the steps will now be described in detail.

3.1 Master problem

The Master Problem is used to generate an initial assignment A, accounting for the availability of items and containers. The MP is formulated as follows.

$$\min \sum_{i \in I} \lambda_i \tag{23}$$

$$\sum_{j \in C} \sum_{r \in R} x^r_{i,j} = n_i - \lambda_i \qquad \forall i \in I \tag{24}$$

$$\sum_{i \in I} \sum_{r \in R} x^r_{i,j} \le m_j \qquad \forall j \in C \tag{25}$$

$$x^r_{i,j}, \lambda_i \in Z^+ \qquad \forall i \in I, j \in C, r \in R \tag{26}$$

3.2 Refinement step

In the refinement step, we apply property 1 to A, to obtain an equivalent 1-irreducible assignment $\hat{A}$ and compute a 2-irreducible sequencing $S_{\hat{A}}$. This procedure produces a set of missions that dominates any other one when attempting to generate a scheduling of the assignment as for Corollary 1 and Corollary 2. Moreover, a 1-irreducible assignment is needed to generate tight constraints in the next phase.

The refinement step is carried out separately for each robot. For a given robot $r \in R$, we store in vector I^r the number of items of each column $i \in I$ assigned to $r \in R$ in A. Similarly, we use vector C^r to store the number of slots of each container $j \in C$ assigned to $r \in R$ in A. We construct $\hat{A}^r$ and $S^r_{\hat{A}}$ iteratively from an empty assignment and sequencing. At each iteration, we add (i', j', r) to $\hat{A}^r$, where $i' = min\{i : I^r[i] > 0\}$ and $j' = min\{j : C^r[j] > 0\}$. We append (i', j', r) to $S^r_{\hat{A}}$ and decrease both $I^r[i']$ and $C^r[j']$ by one. This process is iterated until all elements of I^r and C^r are zero. This procedure is described by Algorithm 1, whose time complexity is linear in the number of missions.

Algorithm 1: The procedure used for refining the assignment A^r of a robot $r \in R$

```
$\hat{A}^r \leftarrow \{\}$;
$S^r_{\hat{A}} \leftarrow \{\}$;
for $i \in I$ do
    $I^r[i] \leftarrow$ # of items $i$ assigned to robot $r$ in $A^r$;
for $j \in C$ do
    $C^r[j] \leftarrow$ # of slots $j$ assigned to robot $r$ in $A^r$;
$p \leftarrow 1$;
while $p \le |\hat{A}^r|$ do
    $i \leftarrow argmin\{I^r[i'] > 0\}$;
    $j \leftarrow argmin\{C^r[j'] > 0\}$;
    $I^r[i] \leftarrow I^r[i] - 1$;
    $C^r[j] \leftarrow C^r[j] - 1$;
    $\hat{A}^r \leftarrow \hat{A}^r \cup (i, j, r)$;
    $S^r_{\hat{A}}[p] \leftarrow (i, j, r)$;
    $p \leftarrow p + 1$;
return $\hat{A}^r$, $S^r_{\hat{A}}$;
```

3.3 Time feasibility check

After the refinement step of each robot r, the feasibility of $S^r_{\hat{A}}$ is checked in terms of time. The feasibility check is carried out separately for each robot, by considering the missions of the sequencing in the order, and iteratively assigning them a starting time. The process continues until either all missions have been considered, or an unfeasibility has been found. We assume without loss of generality that all missions are scheduled at their earliest possible time. Therefore (i, j, r) will start as soon as the robot r is idle after the previous mission, and the column i and the container j are both available to r.

Let us denote by $\psi^p_{(i,j,r)}$ the time instant when robot r becomes available for mission $(i, j, r)^p$. The ready time of the robot $\psi^p_{(i,j,r)}$ is determined as: $\psi^p_{(i,j,r)} = t^{p'}_{(i',j',r)} + \delta$, where $(i', j', r)^{p'}$ is the mission immediately proceeding $(i, j, r)^p$ in the scheduling of r. The starting time of $(i, j, r)^p$ is obtained as follows.

$$t^p_{(i,j,r)} = \max(a^r_i, \alpha^r_j, \psi^p_{(i,j,r)}) \tag{27}$$

If (i, j, r) is the first mission performed in the scheduling we set $\psi_{(i,j,r)} = -\infty$.

If the choice of $t^p_{(i,j,r)}$ given by (27) is not compatible with (1), then there exists no feasible scheduling for $\hat{A}^r$ as a consequence of corollary 2, nor for A^r as for corollary 1. When this occurs, we terminate the time feasibility check for r.

The time feasibility check of a given robot is described by Algorithm 2, whose complexity is linear in the size of A^r.

Algorithm 2: Time feasibility check for robot $r \in R$

```
$(i, j, r)^p \leftarrow S^r_{\hat{A}}[1]$;
$k \leftarrow 1$;
$\psi \leftarrow -\infty$;
unfeasible $\leftarrow$ false;
$T^r_{\hat{A}} \leftarrow \{\}$;
while $k < |S^r_{\hat{A}}|$ do
    $t^p_{(i,j,r)} \leftarrow \max(\psi, a^r_i, \alpha^r_j)$;
    if $t > b^r_i$ or $t > \beta^r_j$ then
        infeasible $\leftarrow$ true;
        break;
    else
        $T_{\hat{A}^r} \leftarrow T_{\hat{A}^r} \cup t^p_{(i,j,r)}$;
        $\psi \leftarrow t^p_{(i,j,r)} + \delta$;
        $(i, j, r)^p \leftarrow S^r_{\hat{A}}[k]$;
        $k \leftarrow k + 1$;
return $T^r_{\hat{A}}$;
```

If Algorithm 2 terminates having assigned a starting time to each mission, $\hat{A}^r$ is feasible. Conversely, if $\hat{A}^r$ is infeasible we generate a new constraint for robot r. Therefore, at each iteration, we can generate at most one constraint for each robot.

3.4 Dynamically generated constraints

When we detect an infeasible mission, we amend the MP generating an additional constraint. In the rest of this section, we first present a class of valid inequalities for the problem. Then, given an infeasible solution, we discuss how the parameters of the presented inequality are selected to generate a violated cut. Lastly, we present a lifting of the generated inequalities that produces stronger cuts for the problem.

For the sake of brevity, let us define a "section" as a subset of the possible missions of a robot contained between a starting and an ending missions $(f, \mathcal{F}, r)$ and $(l, \mathcal{L}, r)$ that is:

$$E^r(f, l, \mathcal{F}, \mathcal{L}) = \{(i, j, r) \in E^r \mid f \le i \le l \wedge \mathcal{F} \le j \le \mathcal{L}\}$$

Given a section $\bar{E}^r = E^r(f, l, \mathcal{F}, \mathcal{L})$, the earliest feasible starting time of any mission in $\bar{E}^r$ is $t_1 = \max(a^r_f, \alpha^r_{\mathcal{F}})$. Consequently, the earliest starting time of the p-th mission is $t_p = \max(a^r_f, \alpha^r_{\mathcal{F}} + (p-1)\delta)$. The latest feasible starting time of any mission in the section is $t_K = \min(b^r_l, \beta^r_{\mathcal{L}})$. Thus, the starting time of all the missions in $\bar{E}^r$ has to fall within the time interval $[t_1, t_K)$. The maximum number of missions that can be contained in $\bar{E}^r$ is given by:

$$u^r(f, l, \mathcal{F}, \mathcal{L}) = 1 + \left\lfloor \frac{min(b^r_l, \beta^r_{\mathcal{L}}) - max(a^r_f, \alpha^r_{\mathcal{F}})}{\delta} \right\rfloor \tag{28}$$

The following constraint constitute a valid inequality for assignment A:

$$\sum_{(i,j,r)\in\bar{E}^r} A^r_{i,j} \le u^r(f, l, \mathcal{F}, \mathcal{L}) \tag{29}$$

We now discuss the choice of $\bar{E}^r$ that yields a violated constraint upon detecting an unfeasibility in $\hat{A}^r$.

Let mission (i'', j'', r) be the infeasible mission detected in $\hat{A}^r$. Additionally, let (i', j', r) be the last mission before (i'', j'', r) in $\hat{A}^r$ such that:

$$t_{(i',j',r)} = \max(a^r_{i'}, \alpha^r_{j'}).$$

According to (27), all missions in $E^r(i', j', i'', j'')$ are performed with no idle time, and the starting time of (i'', j'', r) can be computed as:

$$t_{(i'',j'',r)} = \max(a^r_{i'}, \alpha^r_{j'}) + \delta\Big(\sum_{i\ge i'}^{i''}\sum_{j\ge j'}^{j''} \hat{A}^r_{(i,j,r)} - 1\Big).$$

Since $t_{(i'',j'',r)}$ violates (1), it results that:

$$\min(b^r_{i''}, \beta^r_{j''}) \le \max(a^r_{i'}, \alpha^r_{j'}) + \delta\Big(\sum_{i\ge i'}^{i''}\sum_{j\ge j'}^{j''} x^r_{i,j} - 1\Big).$$

Consequently, setting $\bar{E}^r = E^r(i', j', i'', j'')$ yields a violated constraint for $\hat{A}^r$. Writing (29) utilizing the MP variables leads to the following valid inequality:

$$\sum_{(i,j,r)\in\bar{E}^r} x^r_{i,j} \le u^r(i', j', i'', j'') \tag{30}$$

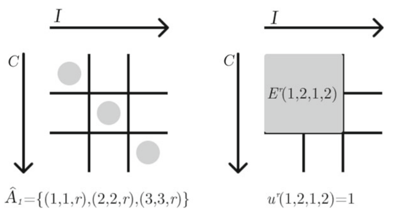

Fig. 3 Initial infeasible assignment of the example

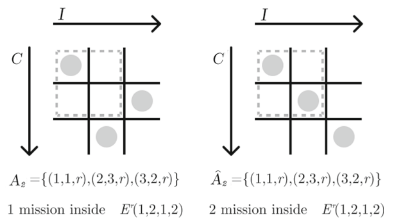
I
C
I
C
$A_2=\{(1,1,r),(2,3,r),(3,2,r)\}$
$\hat{A}_2=\{(1,1,r),(2,3,r),(3,2,r)\}$
1 mission inside $E^r(1,2,1,2)$
2 mission inside $E^r(1,2,1,2)$

Fig. 4 Example of how the refinement step may circumvent valid constraints

Despite being valid, constraint (30) is rendered ineffective by the refinement step that maps A^r to $\hat{A}^r$. Indeed, it is possible to find an assignment A^r that satisfies constraint (30), but that, refined to $\hat{A}^r$, violates it. To understand how this can happen, consider the following example.

Example 1 Let us consider an instance with three columns $I = \{1, 2, 3\}$ and three containers $C = \{1, 2, 3\}$. Each column contains a single item, and each container has a single empty slot. The parameters of the problem are such that any scheduling containing missions $(1, 1, r)$ and $(2, 2, r)$ is infeasible. This implies, by corollary 1, that any scheduling containing $(1, 2, r)$ and $(2, 1, r)$ is also infeasible. Assume that the solution obtained by the MP, after the refinement step is $\hat{A}_1 = \{(1, 1, r), (2, 2, r), (3, 3, r)\}$. The assignment contains both missions $(1, 1, r)$ and $(2, 2, r)$ and it is infeasible; therefore, a constraint should be generated (see Fig. 3). According to (30): $x^r_{1,1} + x^r_{1,2} + x^r_{2,1} + x^r_{2,2} \leq 1$. If the Master Problem is solved again with the additional constraint a valid solution would be the assignment $A_2 = \{(1, 1, r), (2, 3, r), (3, 2, r)\}$. However, after the refinement step, A_2 is reordered into $\hat{A}_2 = \{(1, 1, r), (2, 2, r), (3, 3, r)\}$, which is the same, infeasible assignment obtained before (see Fig. 4). Therefore, the method would cycle, generating the same constraint at each iteration.

We strengthened (30) by enlarging section $\bar{E}^r$ to $\tilde{E}^r = E^r(f, l, \mathcal{F}, \mathcal{L})$. Specifically, we resolve the min and max operators in (28) to determine the largest section $\tilde{E}^r \supseteq \bar{E}^r$, such that $u^r(f, l, \mathcal{F}, \mathcal{L}) = u^r(i', j', i'', j'')$. Let $f^r = inf\{i \in I : (i, j, r) \in E^r\}$, $l^r = sup\{i \in I : (i, j, r) \in E^r\}$, $\mathcal{F}^r = inf\{j \in C : (i, j, r) \in E^r\}$, and $\mathcal{L}^r = sup\{j \in C : (i, j, r) \in E^r\}$. The extremes of $\tilde{E}^r = E^r(f, l, \mathcal{F}, \mathcal{L})$ are determined as follows.

$$f = \begin{cases} i', & \text{if } a^r_{i'} \geq \alpha^r_{j'} \\ f^r, & \text{otherwise} \end{cases} \tag{31}$$

$$\mathcal{F} = \begin{cases} j', & \text{if } a^r_{i'} < \alpha^r_{j'} \\ \mathcal{F}^r, & \text{otherwise} \end{cases} \tag{32}$$

$$l = \begin{cases} i'', & \text{if } b^r_{i''} \leq \beta^r_{j''} \\ l^r, & \text{otherwise} \end{cases} \tag{33}$$

$$\mathcal{L} = \begin{cases} j'', & \text{if } b^r_{i''} > \beta^r_{j''} \\ \mathcal{L}^r, & \text{otherwise} \end{cases} \tag{34}$$

The resulting constraint added to the MP is the following:

$$\sum_{(i,j,r)\in\tilde{E}^r} x^r_{i,j} \leq u^r(i', j', i'', j'') \tag{35}$$

In our previous example, this would correspond to extending the section of the constraint from $E^r(1, 1, 2, 2)$ to either $E^r(1, 1, 3, 2)$ or $E^r(1, 1, 2, 3)$, depending the availability intervals of the items and boxes. Respectively giving rise to either constraint:

$$x^r_{11} + x^r_{12} + x^r_{21} + x^r_{22} + x^r_{31} + x^r_{32} \leq 1$$

or constraint:

$$x^r_{11} + x^r_{12} + x^r_{21} + x^r_{22} + x^r_{13} + x^r_{23} \leq 1.$$

Both of these constraints would prevent the cycling behavior presented.

These " extended " parameters strengthen the constraint, reducing the number of times the MP needs to be solved and preventing the MP to bypass some of the added constraints by exploiting the refinement step.

4 Problem extensions

So far, we have assumed a simple problem setting. Namely, we assumed a constant mission time, deriving from the fact that the robots place the items immediately after they have picked them up; and a constant conveyor speed. This setting is enough representative when the input and output flows are reasonably steady. Indeed, if items and containers arrive on the conveyors with good regularity, placing items just after having picked them up is by far more efficient. However, when the items and the containers start to be highly scattered, the impossibility of holding the items by the robots to wait for the arrival of a container, or to modify the speed of one of the conveyors may be limiting. Thus, excluding some missions that may improve the quality of the scheduling. In this section, we propose two extensions. The first one allows the robot to hold the grip of the items. The second one allows us to modify the speed of one of the two conveyors.

4.1 Item holding

When a robot is allowed to hold an item, the assumption of having a constant mission time does no longer hold true. Indeed, a mission (i, j, r) may start when the items i is available to r but the container j is not yet in the reach of the robot. Thus, formally, $t_{(i,j,r)}$ is not required to be in the intersection of the two intervals $[a_i^r, b_i^r)$ and $[\alpha_j^r, \beta_j^r)$. The feasibility condition (1) is extended as follows.

$$a_i^r \leq t_{(i,j,r)} \leq \min(b_i^r, \beta_j^r). \tag{36}$$

If i and j are such that $a_i^r < \alpha_j^r$ and $t_{(i,j,r)} \in [a_i^r, \alpha_j^r)$ the mission is feasible; however, its duration will be larger than δ. The robot has to wait $\alpha_j^r - t_{(i,j,r)}$ before the j is reachable; thus, the total mission time is $\delta + \alpha_j^r - t_{(i,j,r)}$. We note that in this case, the mission finishes at time $\delta + \alpha_j^r$, regardless of the choice of $t_{(i,j,r)}$.

Without loss of generality, we assume that the starting time of the missions will be selected such that its duration is minimized. Thus, we distinguish two cases, either (1) $[a_i^r, b_i^r)$ and $[\alpha_j^r, \beta_j^r)$ have a non-empty intersection, or (2) $[a_i^r, b_i^r)$ and $[\alpha_j^r, \beta_j^r)$ have an empty intersection. In the first case, mission (i, j, r) has a duration of δ when $t_{(i,j,r)}$ satisfies (1). If the availability of i and j has an empty intersection, and $\beta_j^r < a_i^r$, mission (i, j, r) cannot be performed. If $b_i^r < \alpha_j^r$, the mission is feasible but will involve some waiting. The minimum duration of the mission is $\delta + \alpha_j^r - b_i^r$, and is achieved when $t_{(i,j,r)} = b_i^r$. Let us define $\delta_{(i,j,r)}$ as the minimum duration of mission (i, j, r), i.e.,

$$\delta_{(i,j,r)} = \begin{cases} \delta, & \text{if } a_i^r \leq \beta_j^r \text{ and } \alpha_j^r \leq b_i^r, \\ \delta + \alpha_j^r - b_i^r, & \text{if } b_i^r < \alpha_j^r. \end{cases}$$

Using $\delta_{(i,j,r)}$, condition (2) is updated as follows.

$$\begin{aligned} & t^p_{(i,j,r)} - t^{p'}_{(i',j',r)} \geq \delta_{(i,j,r)} \quad \text{if} \quad t^p_{(i,j,r)} > t^{p'}_{(i',j',r)} \\ & t^{p'}_{(i',j',r)} - t^p_{(i,j,r)} \geq \delta_{(i'j'r)} \quad \text{if} \quad t^p_{(i,j,r)} < t^{p'}_{(i',j',r)} \\ & \forall (i,j,r)^p, (i',j',r)^{p'} \in A^r : (i,j,r)^p \neq (i',j',r)^{p'} \end{aligned} \tag{37}$$

It can be shown that properties 1 and 2 also hold in this extended case.

4.1.1 Extension of the formulation

In order to adapt the formulation to the case where the robots may hold an item during a mission, we explicitly distinguish the case where $\delta_{(i,j,r)} = \delta$ (i.e., the holding case) from the case where $\delta_{(i,j,r)} > \delta$. We replace constraints (14)–(15) with the following ones:

$$a_i^r \leq \Pi_{i,j}^r \leq \min(b_i^r, \beta_j^r) + M(1 - y_{i,j}^r) \qquad \forall i \in I, j \in C, r \in R \tag{38}$$

$$\max(a_i^r, \alpha_j^r) \leq \Pi_{i,j}^r \qquad \forall i \in I, j \in C, r \in R \ : \ \delta_{(i,j,r)} = \delta \tag{39}$$

$$\Pi_{i,j}^r \leq \min(b_i^r, \beta_j^r) - \delta(1 - x_{i,j}^r) + M(1 - y_{i,j}^r) \qquad \forall i \in I, j \in C, r \in R \ : \ \delta_{(i,j,r)} = \delta \tag{40}$$

$$\Pi_{i,j}^r = b_i^r \qquad \forall i \in I, j \in C, r \in R \ : \ \delta_{(i,j,r)} > \delta \tag{41}$$

$$x_{i,j}^r \leq 1 \qquad \forall i \in I, j \in C, r \in R \ : \ \delta_{(i,j,r)} > \delta \tag{42}$$

Constraints (38) enforce the feasibility of $t^1_{(i,j,r)}$. Constraints (39)–(40) enforce the feasibility of all missions (i, j, r) when $\delta_{(i,j,r)} = \delta$. We observe that $\delta_{(i,j,r)} > \delta$ implies that in a feasible scheduling $x_{i,j}^r \leq 1$. Constraints (41)–(42) enforce $x_{i,j}^r \leq 1$ when $\delta_{(i,j,r)} > \delta$ and fix the starting time of those missions to b_i^r.
We update constraint (17) to consider the variable mission duration.

$$\begin{aligned} & \Pi_{i,j}^r + \delta_{(i,j,r)} y_{i,j}^r \leq \Pi_{i',j'}^r + M(1 - y_{i,j}^r) \\ & \forall i, i' \in I, j, j' \in C, r \in R, i' > i, j' \geq j \end{aligned} \tag{43}$$

4.1.2 Extension of feasibility check

We adjust the time feasibility check to account for item holding by updating the computation of the starting time of the missions (27) as follows.

$$t^p_{(i,j,r)} = \begin{cases} \max(a_i^r, \alpha_j^r, \psi^p_{(i,j,r)}) & \text{if } \delta_{(i,j,r)} = \delta; \\ \max(b_i^r, \psi^p_{(i,j,r)}) & \text{otherwise} \end{cases} \tag{44}$$

Additionally, the computation of the ready time of the robots is updated to use $\delta_{(i,j,r)}$ in place of δ.

4.1.3 Extension of the generated constraints

When the item holding is allowed, the earliest starting time of the first mission of a section is only bounded by the availability of the items, $t_1 = a_l$. However, any mission that starts before the first container is available will have to wait. Thus, the earliest starting time of the second mission in the section is $t_2 = \max(a_f, \alpha_{\mathcal{F}}) + \delta$, and the earliest starting time of the p-th mission is $t_p = \max(a_l^r, \alpha_{\mathcal{F}}^r) + (p-1)\delta$. In general, if $(\min(b_l^r, \beta_{\mathcal{L}}^r) - \max(a_f^r, \alpha_{\mathcal{F}}^r)) > 0$, the capacity of a section $E^r(f, l, \mathcal{L}, \mathcal{F})$ follows the same formula as the non-holding case (28). On the other hand, if $(\min(b_l^r, \beta_{\mathcal{L}}^r) - \max(a_f^r, \alpha_{\mathcal{F}}^r)) \leq 0$, we need to distinguish two cases, $\beta_{\mathcal{L}}^r - a_f^r \leq 0$, and $b_l^r - \alpha_{\mathcal{F}}^r <= 0$. In the first case, the first item of the section becomes available after the last container is no longer within reach of the robot. Therefore, the capacity of the section is $u^r(f, l, \mathcal{L}, \mathcal{F}) = 0$. In the second case, the robot may pick up an item and wait for the first container of the section to become available; thus, the capacity of the section is $u^r(f, l, \mathcal{L}, \mathcal{F}) = 1$.

Lastly, we discuss the choice of the section on which to impose the generated constraints. We recall that when generating constraints, we consider the section $\bar{E}^r = E^r(i', j', i'', j'')$, and later expand it using (31)–(34). Where mission (i'', j'', r) is the infeasible mission detected in the current solution, and (i', j', r) is the last mission before (i'', j'', r) in $\hat{A}^r$ such that:

$$t_{(i',j',r)} = \max(a_i^r, \alpha_j^r).$$

So that all missions in $E^r(i', j', i'', j'')$ are performed with no idle time.

We update the choice of mission (i', j', r) to account for the possibility of holding the items, as the last mission before $(i'', j'', r) \in \hat{A}^r$ such that:

$$t^p_{(i',j',r)} = \begin{cases} \max(a_i^r, \alpha_j^r) & \text{if } \delta_{(i,j,r)} = \delta; \\ b_i^r & \text{otherwise.} \end{cases}$$

4.2 Controllable output conveyor

We now consider the possibility of varying the output conveyor speed. We adapt the exact algorithm of Sect. 3 to this case.

4.2.1 Representation of the conveyor speeds

To represent the speed dimension, we apply a discretization of the conveyor length into fixed steps. Let $\{1, \ldots, S\}$ be the set of steps of length l dividing the conveyor space. We denote by τ_s the time the conveyor takes to move along step $s \in \{1, \ldots, S\}$, we refer to this quantity as " step time ". During that time, we assume that the conveyor maintains a constant speed given by $v_s = \frac{l}{\tau_s}$. See an example of discretization and speed profile in Fig. 5.

Using this representation, we can rewrite any distance d on the conveyor as $d = \gamma l + \sigma l$, with $\gamma \in \{0, \ldots, S-1\}$ and $\sigma \in [0, 1)$. Meaning that to cover distance d, the conveyor needs to make γ steps of length l and then cover a portion σ

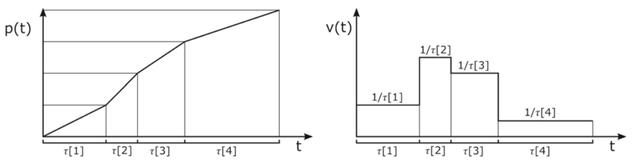

Fig. 5 An example of the resulting motion that can be obtained decoding of the speed profile. On the left the position of the conveyor over time, on the right the speed of the conveyor

of the next step. Consequently, the time needed to cover d is:

$$t = \begin{cases} \sigma\tau_1, & \text{if } \gamma = 0; \\ \sum_{j=1}^{\gamma} \tau_j + \sigma\tau_{\gamma+1}, & \text{otherwise} \end{cases}$$

Using the position of the containers on the line, it is possible to rewrite the availability interval of a container as a function of the conveyor speed. Let the quantities $\overline{\gamma}_j^r$, $\overline{\gamma}_j^r$, $\overline{\gamma}_j^r$, and $\overline{\gamma}_j^r$ describe the position of the containers on the conveyor. We express α_j^r and β_i^r as follows.

$$\alpha_j^r = \sum_{j=1}^{\underline{\gamma}_j^r} \tau_j + \underline{\sigma}_j^r \tau_{\underline{n}_j^r+1} \tag{45}$$

$$\beta_j^r = \sum_{j=1}^{\overline{\gamma}_j^r} \tau_j + \overline{\sigma}_j^r \tau_{\overline{n}_j^r+1} \tag{46}$$

4.2.2 Extension of the method

To adapt the model and the solution method to the variable conveyor speed case, we redefine the parameters α and β depending on variables τ_s using (45) and (46).

For each generated constraint, we introduce an additional binary helper variable z, which equals one if the constraint is deactivated and zero otherwise. The generated constraints are modified as follows.

$$\sum_{(i,j,r) \in \tilde{E}^r} x_{i,j}^r \leq u + Mz \qquad \forall i \tag{47}$$

$$x_{i,j}^r \leq n_i(1-z) \qquad \forall f \leq i \leq l, \mathcal{F} \leq j \leq \mathcal{L} \tag{48}$$

Where M is an arbitrarily large constant. To generate the modified constraints, we follow the same procedure described in the fixed speed case. Let $\tilde{\beta}_{j''}^r$ and $\tilde{\alpha}_{j'}^r$ be equal to the values that $\beta_{j''}^r$ and $\alpha_{j'}^r$ assume under the optimal conveyor speed at the moment of the constraint generation. The parameters that define the constraint are set as follows:

$$u = \begin{cases} \frac{(b^r_{i''} - a^r_{i'})}{\delta}, & \text{if } (b^r_{i''} < \tilde{\beta}^r_{j''}) \wedge (a^r_{i'} \geq \tilde{\alpha}^r_{j'}). \\ \frac{(b^r_{i''} - \alpha^r_{j'})}{\delta}, & \text{if } (b^r_{i''} < \tilde{\beta}^r_{j''}) \wedge (a^r_{i'} < \tilde{\alpha}^r_{j'}). \\ \frac{(\beta^r_{j''} - a^r_{i'})}{\delta}, & \text{if } (b^r_{i''} \geq \tilde{\beta}^r_{j''}) \wedge (a^r_{i'} \geq \tilde{\alpha}^r_{j'}). \\ \frac{(\beta^r_{j''} - \alpha^r_{j'})}{\delta}, & \text{if } (b^r_{i''} \geq \tilde{\beta}^r_{j''}) \wedge (a^r_{i'} < \tilde{\alpha}^r_{j'}). \end{cases} \tag{49}$$

and $\tilde{E}^r = E^r(f, l, \mathcal{F}, \mathcal{L})$ with:

$$f = \begin{cases} i', & \text{if } a^r_{i'} \geq \tilde{\alpha}^r_{j'} \\ f^r, & \text{otherwise} \end{cases} \tag{50}$$

$$\mathcal{F} = \begin{cases} j', & \text{if } a^r_{i'} < \tilde{\alpha}^r_{j'} \\ \mathcal{F}^r, & \text{otherwise} \end{cases} \tag{51}$$

$$l = \begin{cases} i'', & \text{if } b^r_{i''} \leq \tilde{\beta}^r_{j''} \\ l^r, & \text{otherwise} \end{cases} \tag{52}$$

$$\mathcal{L} = \begin{cases} j'', & \text{if } b^r_{i''} > \tilde{\beta}^r_{j''} \\ \mathcal{L}^r, & \text{otherwise} \end{cases} \tag{53}$$

These constraints have been modified to handle three issues.

Firstly, the section $E^r(f, l, \mathcal{F}, \mathcal{L})$ of constraints (35) is dependent on α and β. We use (50–53) to resolve the outcomes of (31–34) at the generation of the constraints.

Secondly, the expression of the capacity of a section (28) is nonlinear in the availability interval of the containers. We use (49) to resolve the outcome of the min and max operators in (28) beforehand.

Lastly, the capacity of a section can be a function of the availability of the containers. Therefore, the capacity of the constraints can become negative under some velocity profiles, which would make the problem infeasible. The constraint (47) can be deactivated to account for this occurrence and guarantee feasibility. In this case, the constraint (48) guarantees that no mission in the section is performed.

Finally, M is set to the maximum negative value that the capacity of the constraint can assume, that is:

$$M = \max(0, -\min_{\tau_s, s \in \{i \ldots S\}} u)$$

5 An ILS based heuristic

The real-time nature of the problem and its complexity requires timely solution approaches that are not compatible with the exact ones. To cope with the requirements of a rolling horizon approach, we develop a metaheuristic algorithm for the problem based on an Iterated Local Search (ILS) approach (Lourenço et al. 2019). The advantage of using a metaheuristic is twofold. Firstly, metaheuristic algorithms are generally much faster than exact ones, making them generally more suited to real-time applications. Secondly, iterated heuristics can be terminated early and still provide a feasible solution, whereas this is not guaranteed to be the case for exact algorithms. The ILS alternates a local improvement phase and a perturbation phase to escape local optima. The algorithm terminates after N_{ILS} iterations of perturbation and local improvement. The basic framework of Iterated Local Search is detailed in Algorithm 3.

Algorithm 3: The ILS framework.

```
$s_0 \leftarrow initialSolution()$;
$s_{best} \leftarrow s^* \leftarrow improve(s_0)$;
for $i$ 1 to $N_{ILS}$ do
    $s_n \leftarrow perturb(s^*)$;
    $\hat{s}_n \leftarrow improve(s_n)$;
    if $acceptanceCriterion(\hat{s}_n)$ then
        $s^* \leftarrow \hat{s}_n$;
    if $f(\hat{s}_n) > f(s_{best})$ then
        $s_{best} \leftarrow \hat{s}_n$;
return $s_{best}$;
```

In the following sections, we will detail the components of the implemented ILS.

5.1 Solution representation

In the ILS, the solutions are represented as a separate allocation of items and slots to the robots. For each robot $r \in R$, vector I^r stores the number of items allocated to robot r for each column $i \in I$. Vector C^r stores the number of slots allocated to robot r for each container $j \in C$. To evaluate the solutions, we need to translate vectors I^r and C^r into a feasible assignment and scheduling.

This decoding can be computed as the solution of the following optimization problem. Given the vector sets: I^r and C^r $\forall r \in R$, find the assignment A and scheduling T_A that minimizes the number of lost items and that satisfies the following condition.

$$I^r[i] \geq \sum_{j \in C} A^r_{i,j} \quad \forall i \in I, r \in R,$$

$$C^r[j] \geq \sum_{i \in I} A^r_{i,j} \quad \forall j \in C, r \in R.$$

The decoding is carried out separately for each robot r constructing A^r and T^r_A iteratively. At each iteration, we select mission (i', j', r) as a candidate to be added to the assignment, where $i' = \min\{i : I^r_i > 0\}$ and $j' = \min\{j : C^r_j > 0\}$. We attempt to generate a starting time for mission $t_{(i',j',r)}$ following (27). If the generated starting time is compatible with (1), we add mission (i', j', r) to A and $t_{(i',j',t)}$ to T^r_A and decrease both $I^r[i']$ and $C^r[j']$ by one. Otherwise, the candidate mission is infeasible. When this is the case, either the current column or the current container has to be discarded and considered lost. If $b^r_{i'} > \beta^r_{j'}$, we discard the current container j' and we set $C^r[j'] \leftarrow 0$. Otherwise we discard the current column i' and set $I^r[i'] \leftarrow 0$. The process is iterated as long as both I^r and C^r have at least one nonzero element.

The pseudocode of the decoding procedure is presented in Algorithm 4.

Algorithm 4: The procedure to decode the solution of robot $r \in R$

```
$i \leftarrow \min_{I^r[i']>0}(i')$;
$j \leftarrow \min_{C^r[j']>0}(j')$;
$p \leftarrow 1$;
nextIdle $\leftarrow -\infty$;
$A^r \leftarrow \{\}$;
$T^r_A \leftarrow \{\}$;
while $\exists i' : I^r_{i'} > 0$ and $\exists j' : C^r_{j'} > 0$ do
    if $I^r[i] = 0$ then
        $i \leftarrow \min_{I^r[i']>0}(i')$;
        $p \leftarrow 1$;
    if $C^r[j] = 0$ then
        $j \leftarrow \min_{C^r[j']>0}(j')$;
        $p \leftarrow 1$;
    $t \leftarrow \max(\text{nextIdle}, a^r_i, \alpha^r_j)$;
    if $t > \min(b^r_i, \beta^r_j)$ then
        if $b^r_i < \beta^r_j$ then
            $I^r[i] \leftarrow 0$;
            continue;
        else
            $C^r[j] \leftarrow 0$;
            continue;
    $t^p_{(i,j,r)} \leftarrow t$;
    $A^r \leftarrow A^r \cup (i, j, r)^p$;
    $T^r_A \leftarrow T^r_A \cup (i, j, r)^p$;
    $I^r[i] \leftarrow I^r[i] - 1$;
    $C^r[j] \leftarrow C^r[j] - 1$;
    $p \leftarrow p + 1$;
return $A^r, S^r_A$;
```

5.2 Local improvement phase

The local improvement phase of the Iterated Local Search is carried out through a Local Search strategy. The move of the Local Search consists of moving a single element from the allocation of one robot to another. We define two moves:

- $moveI(i, r_1, r_2)$:
  $I^{r1}[i] \leftarrow I^{r1}[i] - 1, I^{r2}[i] \leftarrow I^{r2}[i] + 1$
- $moveC(j, r_1, r_2)$:
  $C^{r1}[j] \leftarrow C^{r1}[j] - 1, C^{r2}[j] \leftarrow C^{r2}[j] + 1$

Note that moving a single element from one robot to another does not guarantee to produce an improvement in the objective function. Instead, to observe an improvement, it is frequently necessary to concatenate multiple moves. Thus, we accept all moves that do not decrease the objective. This strategy incurs the risk of making the Local Search rather ineffective and prone to cycling. To overcome this issue, we utilize heuristic information to guide the Local Search. Specifically, we estimate the benefit of all moves before attempting them so that only the most promising moves are considered.

5.3 Estimation of the move impact

The estimation of the move impact is carried out in three steps. First, we gather a set of local features of the scheduling, describing how each item and container relates to the rest of the current solution. Afterward, the gathered features are combined to describe a more general set of aggregated features, describing the status of the system at each point of the scheduling. Lastly, the features are evaluated to give an estimate of the quality of each available move. After having estimated the impact of the moves, we use a random approach to determine which move to be attempted, coupled with a short tabu list to prevent the method from cycling.

5.3.1 Computation of the local features

The first step of the evaluation computes the local features for the items and containers. All features are computed in relation to the current solution s and associated assignment A and scheduling T_A. These features account for the contribution of the individual items and containers to the overall workload of the robots.

In the following discussion, for the sake of brevity, we will only present the local features associated with the items.

The information we gather is summarized by a set of binary features for each robot-column pair $(r, i), r \in R, i \in I$ with at least one item allocated, i.e., $I^r[i] \geq 1$.

- $w^r_1[i] = 1$ if there is a mission (i, j, r) in A, such that $t_{(i,j,r)} = a^r_i$; zero otherwise.
- $w^r_2[i] = 1$ if there is a mission (i, j, r) in A, such that $t_{(i,j,r)} = \alpha^r_j$; zero otherwise.
- $w^r_3[i] = 1$ if during the decoding of s, item i has been discarded due to a mission (i, j, r) not being feasible, with $\alpha^r_j < b^r_i$; zero otherwise.
- $w^r_4[i] = 1$ if during the decoding of s, item i has been discarded due to a mission (i, j, r) not being feasible, with $\alpha^r_j > b^r_i$; zero otherwise.
- $w^r_5[i] = 1$ if there is a mission (i, j, r) in A such that $a^r_i > \psi_{(i,j,r)} + \delta$; zero otherwise.

- $w^r_6[i] = 1$ if it is possible to add another item i to the allocation of r without violating the feasibility; zero otherwise.
- $w^r_7[i] = 1$ if it is possible to remove one item i from the allocation of r without delaying the rest of the scheduling; zero otherwise.

$w^r_1[i]$, $w^r_2[i]$, $w^r_3[i]$, $w^r_4[i]$, and $w^r_5[i]$ are easily computed during the decoding of the solutions. On the other hand, features $w^r_6[i]$ and $w^r_7[i]$ require more considerations to be computed. In the case of $w^r_6[i]$, let us consider the neighboring solution $\hat{s}$, which corresponds to solution s with one more item i assigned to robot r. In this case, the addition of another item might have one of two effects. Either the added item i is discarded, and the decoded solution of $\hat{s}$ corresponds with the decoded solution of s, or the item is not discarded, and the two decoded solutions differ. If the item is discarded, $w^r_6[i]$ equals zero, otherwise it equals one. In the case of $w^r_7[i]$, let us consider the neighboring solution $\hat{s}$, which corresponds to solution s with one less item i assigned to robot r. In this case, the removal of one item can have several possible outcomes. If an item i was discarded in the decoding of s, the decoded solutions of $\hat{s}$ and s correspond and $w^r_7[i]$ equals one. If no item i was discarded, the last item i associated to robot r in s is part of a mission $(i, j, r)^p$. In $\hat{s}$, slot j is either lost, or part of a different mission (i', j, r) with $i' > i$. If slot j is lost, $w^r_7[i]$ equals zero. If slot j is part of a different mission $(i', j, r)^{p'}$ we further distinguish two cases: either $t^p_{(i,j,r)} < t^{p'}_{(i',j,r)}$ or $t^p_{(i,j,r)} = t^{p'}_{(i',j,r)}$. In the first case $w^r_7[i]$ equals zero, otherwise it equals one.

The local features $\bar{w}^r_{1-7}[i]$ of the containers are computed in a similar way.

5.3.2 Computation of the aggregated features

We now describe how the heuristic features discussed earlier can be aggregated into higher-level features, which describe the condition of the scheduling from a higher-level point of view. Those aggregated features are computed propagating the information obtained at the local level, to neighboring items, when it can be applied. They are evaluated iteratively, starting from the last items on the conveyor. For the purposes of the computation, we consider $W^r_{1-5}[n+1] = 0$ and $W^r_6[n+1] = 1$.

$$\begin{aligned}
W^r_1[i] = 1 \quad & \text{if } w^r_1[i] = 1 \\
& \text{or } (W^r_1[i+1] = 1 \text{ and } w^r_6[i] = 1), \\
W^r_2[i] = 1 \quad & \text{if } w^r_2[i] = 1 \\
& \text{or } (W^r_2[i+1] = 1 \text{ and } W^r_1[i+1] = 0), \\
W^r_3[i] = 1 \quad & \text{if } w^r_3[i] = 1 \\
& \text{or } (W^r_3[i+1] = 1 \text{ and } w^r_7[i] = 1), \\
W^r_4[i] = 1 \quad & \text{if } w^r_4[i] = 1 \\
& \text{or } (W^r_4[i+1] = 1 \text{ and } w^r_7[i] = 1), \\
W^r_5[i] = 1 \quad & \text{if } w^r_5[i] = 1 \\
& \text{or } (W^r_5[i+1] = 1 \text{ and } w^r_6[i] = 1), \\
W^r_6[i] = 1 \quad & \text{if } \exists j \in C : ((i, j, r) \in E^r \text{ and } \bar{w}^r_4[j] = 1).
\end{aligned}$$

5.3.3 Evaluation of the moves

After the features of the items and slots have been computed, the impact of $moveI(i, r_1, r_2)$ or $moveC(i, r_1, r_2)$ on the solution value is estimated.

Let us define $V^{r1}_-[i]$ and $V^{r2}_+[i]$ as the heuristic value assigned to the removal of item i from robot $r1$ and the addition of an item i to robot $r2$, respectively. They are computed as follows:

$$\begin{aligned}
V^{r1}_-[i] &\leftarrow -\infty && \text{if } I^{r1}[i] = 0, \\
V^{r1}_-[i] &\leftarrow l1 && \text{if } W^{r1}_i[4] = 1, \\
V^{r}_- &\leftarrow l2 && \text{if } W^{r1}_i[5] = 1, \\
V^{r1}_-[i] &\leftarrow 0 && \text{otherwise.} \\
V^{r2}_+[i] &\leftarrow -g1 && \text{if } (I^{r2}[i] > 0 \text{ and } W^{r2}_i[1] = 0), \\
V^{r2}_+[i] &\leftarrow g2 && \text{if } W^{r2}_i[3] = 1, \\
V^{r2}_+[i] &\leftarrow g3 && \text{if } W^{r2}_i[2] = 1, \\
V^{r2}_+[i] &\leftarrow g4 && \text{if } W^{r2}_i[6] = 1, \\
V^{r2}_+[i] &\leftarrow 0 && \text{otherwise.}
\end{aligned}$$

where $l1$, $l2$, $g1$, $g2$, $g3$, and $g4$ are positive constants. Formally, the heuristic evaluation of $MoveI(i, r1, r2)$ is $V^{r1,r2}[i] = V^{r1}_-[i] + V^{r2}_+[i]$.

5.4 Move selection scheme

To select the moves to be applied in the search, we systematically apply the move evaluation scheme presented in the previous section. The selection is performed iterating over the columns $i \in I$. For each iteration step, we compute the heuristic evaluation of all moves $MoveI(i, r1, r2)$, for all $r1, r2 \in R$. A move is attempted if its evaluation $V^{r1,r2}[i]$ exceeds a move selection threshold th. If none of the moves is attempted, we decrease the threshold by one and iterate to the next column.

At the start of the search and whenever a move is attempted, the move selection threshold th is initialized to a random value in the interval $[0, th_{max}]$. If the attempted move was accepted, th is also increased by the current iteration number. This is done to encourage the method to explore moves uniformly.

To discourage cycling of the moves, we also implement a simple tabu mechanism in the search. When a move

$moveI(i, r1, r2)$ is successful, for the next $tabu\_length$ iterations of the search we mark as tabu any move where an item i is added in the scheduling of robot $r1$, or where an item i is removed from the scheduling of robot $r2$. On the other hand, if a move is rejected, we mark as tabu any move where an item i is added in the scheduling of robot $r2$, or where an item i is removed from the scheduling of robot $r1$. The moves that are marked as tabu are penalized by a factor of $tabu\_penalty$ in the heuristic evaluation of the move.
The approach is summarized in algorithm 5.

Algorithm 5: The move selection process to select and apply the next moveI to the solution.

```
Data: The initial threshold $th_{init}$
$i \leftarrow n$ ;
$k \leftarrow 0$ ;
$th \leftarrow th_{init}$ ;
while $i > 0$ do
    for $r1 \in R$ do
        for $r2 \in R \ : \ r1 \neq r2$ do
            $V \leftarrow V_{-}^{r1}[i] + V_{+}^{r2}[i]$;
            if $V \geq th$ then
                apply $moveI(i, r1, r2)$;
                if $moveI(i, r1, r2)$ is accepted then
                    $th \leftarrow rand(th_{max}) + k$;
                    return $th$;
                else
                    revert $moveI(i, r1, r2)$;
                    $th \leftarrow rand(th_{max})$;
            else
                $th \leftarrow th - 1$ ;
            $k \leftarrow k + 1$ ;
    $i \leftarrow i - 1$ ;
return;
```

The Local Search is stopped after N_{LS} iterations without an improvement in the objective function.

5.5 Initial solution

The initial solution of the ILS is provided by a greedy constructive heuristic. Starting from an empty allocation, we construct the solution iteratively, building the solution of one robot at a time. For each robot, we generate the optimal single robot solution, with the remaining item and containers that are not part of the scheduling.

Practically, this is achieved starting from an empty allocation. For each robot, we temporarily allocate to it all the currently unallocated items and slots, compute the missions of the allocation, and remove all items and slots that are not assigned to any mission. The procedure is iterated until all robots have been processed.

5.6 Perturbation phase

The perturbation phase is used to escape the local optima reached at the end of the local improvement phase. To perturb the solution, we utilize a strategy to destroy a portion of the current solution, surrounding a critical item or container. Specifically, we select an item or a slot that is not assigned to any mission in the current solution. We remove from the allocation of the robots the selected element (column $i \in I$ or container $j \in C$) and all items $i' \in I$ or slots $j' \in C$ such that $|i' - i| \leq P_I$ $|j' - j| \leq P_C$, where P_I and P_C are parameters of the search.

We evaluate the perturbed solution and remove from the allocation all items and containers that did not fit in any mission. We then reconstruct the solution by applying the constructive heuristic presented in Sect. 5.5. To introduce additional diversification in the procedure, we reconstruct the allocation of the robots in a random order, instead of proceeding sequentially.

To explore uniformly the solution space, in our approach, we will alternate between applying the perturbation to items and to containers.

5.7 Extensions of the ILS

The metaheuristic approach we described so far is only able to handle the fixed speed version of the scheduling problem. In the next sections, we will discuss how to extend the metaheuristic method to handle the variable speed case and the item holding case.

5.7.1 Variable speed

We will use the same representation we proposed for the exact methods in Sect. 4.2.1. In addition, we define the following move as far as the speed is concerned.

$$inc(\mu, i, h) : \tau_{i+j} \leftarrow \tau_{i+j} + \mu \qquad \forall j \in [-h, h].$$

In case the move reaches the minimum or maximum speed of the conveyor, we will limit the corresponding τ to that value. To avoid making moves with no effect on the speed profile, when this happens, we increase h by one.

The search for an effective speed profile for the problem is conducted using an iterative improvement strategy. At the start of the algorithm, we initialize the speed profile to a uniform speed across the planning horizon, with the speed set to the steady-state speed of the line. We then use a Local Search strategy to improve the initial speed profile.

To evaluate the quality of a speed profile, a high-quality assignment for that speed profile is required. Therefore, at each iteration we greedily generate an allocation for the line, and improve the assignment using the local improvement strategy presented in Sect. 5.4. We use the generated allocation to evaluate the quality of the current conveyor speed profile. All moves that do not deteriorate the solution are accepted.

We select the moves attempted in the search according to a greedy randomized heuristic. Let s be the incumbent speed profile. Additionally, let A be the assignment that has been generated for s. On A, we select a random lost item i and a random lost slot j. We note that if the current assignment does not have any lost item or slot, the current generated solution is already optimal, and the search may terminate.

Depending on the selection of i and j, two cases may occur, either $a_i^r < \alpha_j^r$ or $a_i^r \geq \alpha_j^r$, for a given reference robot $r \in R$. In the first case, we can hypothesize that the speed of the output conveyor may be increased, so as to move closer the availability intervals of i and j. In the second case, the speed of the output conveyor may be decreased to obtain the same effect. Therefore, in the first case we apply move $inc(x, -\mu, h)$, where $x \in \{1, \dots, k\} : \sum_{j=1}^{x} \tau_j < a_i^r$. In the second case, we apply move $inc(x, \mu, h)$, where $x \in \{1, \dots, k\} : \sum_{j=1}^{x} \tau_j < \alpha_j^r$.

The Local Search strategy is stopped after a total of $N_{S1} + N_{S2}$ iterations. During the first N_{S1} iterations, we set $\mu = \mu_1$ and $h = h_1$. Afterward, we set $\mu = \mu_2$ and $h = h_2$. With $\mu_1 < \mu_2$ and $h_1 > h_2$, doing so causes the initial moves to apply broad corrections to the speed profile and move the solution away from the steady-state solution, and later focus on finer adjustments of the profile. After $N_{S1} + N_{S2}$ total iterations, we fix the speed profile and apply the ILS algorithm to obtain a final solution.

5.7.2 Item holding

The developed ILS can be easily adapted to allow for item holding. The adaptation does not require any structural change of the heuristic. The only change needed is to adjust the solution decoding procedure to account for the variable mission time.

6 Computational results

We performed a series of computational tests to evaluate the performance of the proposed approach and compare it to the more direct modeling approach used in Ferrari et al. (2015). The authors of Ferrari et al. (2015) provided us with data from three 4-manipulators plant configurations, representative of the characteristics of real-world lines where a dynamic scheduling approach could be applied. We report in Table 1 the parameters of the three considered lines are details. For each line, the table reports the column distance(d_i) and the maximum number of items in each column(n_i), the container distance(d_j) and the number of slots in each container(m_j), the velocity of

Table 1 The parameters used for the three configurations for our testing

#	Items		Containers		Input conveyor	Output conveyor		
	d_i	n_i	d_j	n_j	v_i	v_{j_min}	v_{j_max}	v_{j_nom}
0	0.25 m	3	0.45 m	9	20 m/min	10 m/min	20 m/min	12 m/min
1	0.24 m	3	0.18 m	9	20.8 m/min	4 m/min	7.5 m/min	5.2 m/min
2	0.092 m	3	0.3 m	3	8 m/min	12 m/min	40 m/min	26 m/min

Table 2 Row generation, synthetic results

Instace			Base			Holding			One speed			n speeds		
Config	I	C	#S	obj	$t(s)$	#S	obj	$t(s)$	#S	obj	$t(s)$	#S	obj	$t(s)$
0	15	5	11	3.1	0.8	11	2.1	1.4	11	0.1	3.8	11	0.1	2.8
0	30	10	11	3.3	7.6	11	2.1	10.9	8	–	171.3	11	0.3	63.7
0	45	15	11	3.9	50.6	11	2.6	74.4	0	–	250.0	0	–	250.0
0	60	20	9	–	181.8	4	–	236.2	0	–	250.0	0	–	250.0
1	15	5	11	23.4	0.8	11	21.4	0.6	11	17.4	1.2	11	17.4	1.1
1	30	10	11	35.9	15.5	11	34.8	28.5	11	19.1	46.8	11	19.1	35.3
1	45	15	11	36.2	127.5	11	35.1	135.3	11	18.3	119.7	11	18.3	97.0
1	60	20	0	–	250.0	0	–	250.0	1	–	248.7	1	–	245.2
2	15	15	11	19.3	2.8	11	19.3	2.1	11	0.0	4.4	11	0.0	3.6
2	30	30	11	21.9	99.4	11	21.8	54.6	8	–	185.0	10	–	139.6
2	45	45	0	–	250.0	2	–	245.2	0	–	250.0	0	–	250.0
2	60	60	0	–	250.0	0	–	250.0	0	–	250.0	0	–	250.0

the input conveyor(v_i), and the minimum(v_j_min), maximum(v_j_max) and nominal(v_j_nom) velocity of the output conveyors.

For each line configuration, we generated three instances with 15, 30, 45, and 60 columns of items, simulating the start-up conditions of the systems. Additionally, each of the generated instances was used as the basis for ten additional instances where some of the items are missing from the input conveyor and some slots are already filled at the start of the planning horizon.

The exact methods presented in this section were implemented with the AMPL and solved using CPLEX on a 2.00GHz Intel Xeon Processor L5335. The CPLEX solver uses 1 core, and the time limit has been set to 250 seconds. The metaheuristic developed have been written in C++ and ran on a 3.20Ghz AMD Ryzen 5 1600, in single-thread mode. The tuning parameters of the Iterated Local Search have been determined experimentally (Schettini 2017) and are reported in Table 9, in the "Appendix."

6.1 Exact methods

In Table 2, we report the summary of the results of the row generation approach in the basic version of the problem (base) and using the two presented extensions (holding, one speed, n speeds). For each line configuration and instance size, we report the number of instances solved to optimality (#S), the average number of lost items(obj), and the average solution time(t(s)). Note that the developed row generation algorithm does not provide a feasible solution when terminated early. Indeed, the developed cut generation algorithm solves a relaxed version of the problem and iteratively adds violated inequalities until a feasible solution is found. When the optimal solution to the relaxed problem is feasible, that solution is optimal by construction. Conversely, when the instance is not solved to optimality due to early termination, the solution of the relaxed problem is still not feasible; therefore, the algorithm only provides lower bound. Thus, for sake of fairness, we report its average objective only when all averaged instances are solved to optimality; when this is not the case, the value is reported as "-".

In the controllable conveyor case, two versions of the problem are considered, the case where the speed of the output conveyor can be controlled but can assume one value for the entire scheduling, and the case of a more finely controllable speed profile.

In practical terms, the first case (one speed) corresponds to having a speed profile composed of exactly one section, i.e., $l = \infty$. In the second case (n speeds), we consider sections of unitary length, i.e., $l = 1$. All versions of the problem were comparable in terms of computational time and instances solved.

Both extensions of the problem caused an improvement of the objective function. However, the benefits of the controllable conveyor strategy significantly outperform the holding strategy. The resolution of the speed profile did not affect the quality of the computed solutions, nor the difficulty of the problems, despite the increased freedom of the system.

The full experimental results are provided in the "Appendix" on Tables 6 and 7. Those tables, for each instance, report the best known lower bound of the optimal solution for each version of the problem, the computational time used to compute the solution, and the optimality of the solution ($*$ denotes solutions solved to optimality).

6.2 Comparison with Ferrari et al. (2015)

We compared our exact algorithm (ROW) with the model developed by Ferrari et al. (2015) (FERR) This comparison was carried out on the unmodified problem (base) and on the extended problem with controllable conveyor (one speed). In the latter case, to align with the model presented in Ferrari et al. (2015) we consider a constant speed velocity profile, meaning that the velocity profile is controllable, but only one speed can be selected for the entire planning horizon. Table 3 reports the summary of results. For each line configuration and instance size, we report the number of instances solved to optimality (#S), the average number of lost items(obj), and the average solution time(t(s)). The results are reported both for the basic problem and the controllable conveyors extension. The gap reported in the table is computed from the complement of the objective function, which maximizes the number of items moved into the containers. Table 5 reports the complete experimental results, for each instance, and for each solution method, the table reports the objective value and the solution time. We also report the optimality of the solution in the case of ROW and the optimality gap in the case of FERR.

We observe that ROW outperforms FERR in terms of computation time, being able to solve to optimality a larger portion of instances. However, we note that due to its nature, when the row generation does not reach an optimal solution, the solution provided at the end of the computation is not feasible. Conversely, the final solution provided by FERR is feasible, even if suboptimal. In the case of ROW, it is possible to repair the solutions obtained at the end of the computation utilizing the decoding algorithm presented in Sect. 5.1. However, on such occasions, it is more convenient to directly apply the ILS heuristic on the instance.

Overall, neither ROW nor FERR provide computational times that are compatible with a real-time application of the algorithm on any of the lines considered. However, their results are still useful for the purposes of understanding the problem and evaluating the developed heuristic.

Table 3 Synthetic comparison with Ferrari et al. (2015)

Instance			FERR—base			ROW–base			FERR–one speed			ROW–one speed		
Config	I	C	#S	obj	t(s)	#S	obj	t(s)	#S	obj	t(s)	#S	obj	t(s)
0	15	5	5	3.2	185.5	11	3.1	0.8	11	0.1	59.2	11	0.1	3.8
0	30	10	0	4.4	250	11	3.3	7.6	1	2.7	248.4	8	–	171.3
1	15	5	11	23.4	0.5	11	23.4	0.8	11	17.4	36	11	17.4	1.2
1	30	10	8	35.9	133.7	11	35.9	15.5	0	19.6	250	11	19.1	46.8
2	15	15	11	19.3	45.8	11	19.3	2.8	11	0	51.4	11	0	4.4
2	30	30	0	22.8	250	11	21.9	99.4	0	18.5	250	8	–	185

Table 4 Iterated Local Search, synthetic results

Instance			Base			Holding			*n* speeds		
Config	I	C	obj	LB	t(s)	obj	LB	t(s)	obj	LB	t(s)
0	60	20	4.5	4.2	1.4	4.0	2.8	1.4	0.0	0.0	0.1
1	60	20	36.8	36.1	1.2	35.7	35.0	1.2	20.2	18.3	1.1
2	60	60	22.4	22.0	1.3	22.4	22.1	1.3	0.0	0.0	0.2

6.3 Metaheuristic approach

In Table 4, we report the summary of the results of the developed heuristic approach on the 60 column instances. For each line configuration, we report the average number of items lost(obj), the best known lower bound to the number of lost items(LB), and the average solution time of the metaheuristic approach(t(s)). The problem has been solved in its basic version and considering the item holding and controllable conveyors extension. From the results, we observe that the computation time associated with the developed heuristic is limited to a couple of seconds, maintaining a high solution quality. Thus, the computational time achieved by the ILS is compatible with a real-time application of the algorithm to the considered lines.

The full experimental results are provided in the “Appendix” in Table 8. The table, for each instance, reports the solution obtained by the Iterated Local Search, and the computational time utilized. We also report the best known lower bound of the corresponding instances.

7 Conclusions and future work

In this paper, we discussed the Pick and Place Packaging Problem for the optimization of a two-conveyor packaging line. We presented an integer linear programming model for the problem and an efficient exact algorithm based on row generation. We also developed an effective and time efficient heuristic based on Iterated Local Search. We presented two extensions of the problem. Specifically, we consider the case where one of the conveyors can be controlled, and the case where the robots may hold items without immediately moving to their destination. Both the exact algorithm and the Iterated Local Search have been extended to handle these two extensions. The developed model and algorithms have been tested on a series of testing instances simulating realistic plant configurations. Our results show the effectiveness of the developed approaches and the utility of the optimization of robot scheduling. Some extensions of the problem can deserve further investigations. One is the multi-grip case when robots can pick up more than one item at a time and place them in multiple containers. Another case considers items of different types mixed in the system. In both cases, the extension of the mathematical model and of the heuristic can be particularly interesting.

Funding Open access funding provided by Politecnico di Milano within the CRUI-CARE Agreement.

Appendix

See Tables 5, 6, 7, 8 and 9.

Table 5 Comprehensive comparison of the row generation with Ferrari et al. (2015)

Instance			FERR—fixed			ROW—fixed		FERR—speed			ROW—speed	
config	I	C	obj	gap	t(s)	obj	t(s)	#S	obj	t(s)	obj	t(s)
0	15	5	4	2.44	250	4*	1	0	0.00	53	0*	4
			2	0.00	233	2*	1	0	0.00	32	0*	4
			3	0.00	5	3*	0	0	0.00	146	0*	3
			3	2.78	250	3*	1	0	0.00	27	0*	4
			2	0.00	88	2*	1	0	0.00	26	0*	3
			2	0.00	48	2*	1	1	0.00	34	1*	2
			3	2.50	250	3*	0	0	0.00	132	0*	6
			4	4.98	250	4*	2	0	0.00	27	0*	4
			4	5.00	250	4*	0	0	0.00	45	0*	4
			5	7.36	250	4*	1	0	0.00	81	0*	5
			3	0.00	164	3*	1	0	0.00	48	0*	3
			3.2	2.28	185	3.1(11)	1	0.1	0.00	59	0.1(11)	4
0	30	10	6	7.14	250	4*	8	7	8.43	250	0*	214
			4	5.06	250	2*	6	2	2.47	250	0*	150
			3	3.70	250	3*	7	2	2.44	250	0*	137
			4	2.60	250	4*	9	3	1.28	250	2*	118
			3	4.00	250	2*	5	0	0.00	228	0*	70
			3	4.11	250	2*	6	1	1.33	250	0*	95
			5	6.02	250	3*	6	1	1.15	250	0*	197
			5	4.76	250	4*	7	2	1.15	250	1	250
			6	7.23	250	4*	9	2	2.30	250	0	250
			5	6.02	250	4*	12	1	1.15	250	0	250
			4	4.82	250	4*	9	9	11.54	250	0*	153
			4.4	5.04	250	3.3(11)	8	2.7	3.02	248	0.3(8)	171
1	15	5	26	0.00	1	26*	1	20	0.00	42	20*	2
			21	0.00	1	21*	1	15	0.00	61	15*	1
			21	0.00	0	21*	1	15	0.00	65	15*	1
			22	0.00	1	22*	1	16	0.00	26	16*	1
			20	0.00	0	20*	1	14	0.00	46	14*	1
			21	0.00	1	21*	0	15	0.00	27	15*	1
			25	0.00	0	25*	1	19	0.00	39	19*	1
			26	0.00	1	26*	1	20	0.00	14	20*	1
			25	0.00	0	25*	1	19	0.00	21	19*	1
			25	0.00	1	25*	1	19	0.00	32	19*	1
			25	0.00	0	25*	0	19	0.00	23	19*	2
			23.4	0.00	1	23.4(11)	1	17.4	0.00	36	17.4(11)	1
1	30	10	41	0.00	96	41*	17	24	18.18	250	23*	110
			32	0.00	56	32*	16	16	9.23	250	16*	14
			30	14.75	250	30*	24	15	9.38	250	15*	19
			33	0.00	164	33*	15	17	9.23	250	17*	17
			31	5.73	250	31*	15	16	12.50	250	15*	17
			30	7.70	250	30*	10	17	11.48	250	16*	18
			39	0.00	37	39*	14	22	16.67	250	21*	62
			40	0.00	73	40*	17	22	14.93	250	22*	57
			40	0.00	99	40*	15	23	16.67	250	22*	48
			39	0.00	157	39*	19	21	14.93	250	21*	81
			40	0.00	39	40*	9	23	16.67	250	22*	72

Table 5 continued

Instance			FERR—fixed			ROW—fixed		FERR—speed			ROW—speed	
config	I	C	obj	gap	t(s)	obj	t(s)	#S	obj	t(s)	obj	t(s)
			35.9	2.56	134	35.9(11)	16	19.6	13.62	250	19.1(11)	47
2	15	15	22	0.00	4	22*	3	0	0.00	46	0*	5
			17	0.00	5	17*	3	0	0.00	76	0*	5
			14	0.00	16	14*	3	0	0.00	74	0*	3
			18	0.00	21	18*	3	0	0.00	34	0*	7
			17	0.00	21	17*	2	0	0.00	49	0*	3
			19	0.00	5	19*	3	0	0.00	44	0*	6
			21	0.00	4	21*	2	0	0.00	158	0*	5
			21	0.00	5	21*	2	0	0.00	15	0*	4
			21	0.00	3	21*	4	0	0.00	36	0*	3
			21	0.00	210	21*	3	0	0.00	17	0*	4
			21	0.00	210	21*	3	0	0.00	16	0*	3
			19.3	0.00	46	19.3(11)	3	0.0	0.00	51	0.0(11)	4
2	30	30	26	10.92	250	24*	94	23	34.33	250	0*	217
			19	14.52	250	19*	101	9	12.50	250	0*	104
			15	8.34	250	16*	54	18	31.58	250	0*	158
			21	13.11	250	20*	97	33	65.31	250	1	250
			22	11.67	250	22*	105	6	5.26	250	2	250
			20	14.75	250	19*	96	2	2.53	250	0*	184
			26	12.72	250	24*	107	24	36.92	250	0*	132
			25	10.25	250	24*	137	15	20.55	250	0	250
			25	15.91	250	25*	100	18	24.29	250	1*	125
			26	12.37	250	24*	102	37	71.15	250	0*	182
			26	12.37	250	24*	100	18	25.35	250	0*	183
			22.8	12.45	250	21.9(11)	99	18.5	29.98	250	0.4(8)	185

Table 6 Row generation, comprehensive results

Instance			Base		Holding		1 speed		*n* speeds	
config	I	C	obj	t(s)	obj	t(s)	obj	t(s)	obj	t(s)
0	15	5	4*	1	3*	2	0*	4	0*	3
			2*	1	1*	1	0*	4	0*	3
			3*	0	2*	1	0*	3	0*	3
			3*	1	2*	1	0*	4	0*	2
			2*	1	1*	2	0*	3	0*	4
			2*	1	1*	1	1*	2	1*	1
			3*	0	2*	2	0*	6	0*	3
			4*	2	3*	1	0*	4	0*	3
			4*	0	3*	2	0*	4	0*	3
			4*	1	3*	1	0*	5	0*	4
			3*	1	2*	1	0*	3	0*	2
			3.1(11)	1	2.1(11)	1	0.1(11)	4	0.1(11)	3

Table 6 continued

Instance			Base		Holding		1 speed		*n* speeds	
config	I	C	obj	t(s)	obj	t(s)	obj	t(s)	obj	t(s)
0	30	10	4*	8	3*	12	0*	214	0*	53
			2*	6	1*	7	0*	150	0*	48
			3*	7	2*	15	0*	137	0*	75
			4*	9	3*	14	2*	118	2*	68
			2*	5	1*	8	0*	70	0*	43
			2*	6	0*	6	0*	95	0*	58
			3*	6	2*	12	0*	197	0*	53
			4*	7	3*	14	1	250	1*	74
			4*	9	3*	12	0	250	0*	125
			4*	12	3*	11	0	250	0*	70
			4*	9	2*	9	0*	153	0*	34
			3.3(11)	8	2.1(11)	11	0.3(8)	171	0.3(11)	64
0	45	15	4*	72	3*	57	0	250	0	250
			2*	32	1*	44	1	250	1	250
			4*	85	2*	120	1	250	0	250
			7*	73	6*	152	3	250	3	250
			2*	35	1*	87	1	250	0	250
			3*	37	0*	24	0	250	0	250
			3*	30	2*	65	1	250	0	250
			4*	34	3*	42	1	250	1	250
			5*	48	4*	90	1	250	1	250
			5*	43	4*	75	1	250	1	250
			4*	68	3*	62	1	250	0	250
			3.9(11)	51	2.6(11)	74	1.0(0)	250	0.6(0)	250
0	60	20	4*	170	3*	182	1	250	0	250
			3*	140	2*	228	0	250	0	250
			4*	228	2	250	1	250	0	250
			7*	188	5	250	0	250	0	250
			2*	143	1*	246	0	250	0	250
			3	250	1*	192	1	250	0	250
			4*	143	2	250	0	250	0	250
			5*	142	4	250	0	250	0	250
			5*	156	4	250	0	250	0	250
			5*	190	4	250	0	250	0	250
			4	250	3	250	1	250	0	250
			4.2(9)	182	2.8(4)	236	0.4(0)	250	0.0(0)	250
1	15	5	26*	1	24*	0	20*	2	20*	1
			21*	1	19*	1	15*	1	15*	1
			21*	1	19*	0	15*	1	15*	1
			22*	1	20*	1	16*	1	16*	1
			20*	1	18*	0	14*	1	14*	1
			21*	0	19*	1	15*	1	15*	2

Table 6 continued

Instance			Base		Holding		1 speed		*n* speeds	
config	I	C	obj	t(s)	obj	t(s)	obj	t(s)	obj	t(s)
			25*	1	23*	1	19*	1	19*	1
			26*	1	24*	0	20*	1	20*	1
			25*	1	23*	2	19*	1	19*	1
			25*	1	23*	1	19*	1	19*	2
			25*	0	23*	0	19*	2	19*	0
			23.4(11)	1	21.4(11)	1	17.4(11)	1	17.4(11)	1
1	30	10	41*	17	40*	30	23*	110	23*	28
			32*	16	31*	26	16*	14	16*	17
			30*	24	29*	15	15*	19	15*	13
			33*	15	32*	31	17*	17	17*	15
			31*	15	30*	35	15*	17	15*	17
			30*	10	28*	18	16*	18	16*	13
			39*	14	38*	29	21*	62	21*	20
			40*	17	39*	29	22*	57	22*	44
			40*	15	39*	40	22*	48	22*	18
			39*	19	38*	38	21*	81	21*	140
			40*	9	39*	23	22*	72	22*	63
			35.9(11)	16	34.8(11)	29	19.1(11)	47	19.1(11)	35

Table 7 Row generation, comprehensive results

Instance			Base		Holding		1 speed		*n* speeds	
config	I	C	obj	t(s)	obj	t(s)	obj	t(s)	obj	t(s)
1	45	15	41*	176	40*	184	21*	139	21*	104
			32*	93	31*	112	16*	89	16*	55
			31*	103	30*	116	15*	92	15*	83
			33*	132	32*	104	17*	116	17*	115
			31*	110	30*	118	15*	95	15*	73
			30*	118	28*	105	16*	82	16*	55
			40*	158	39*	192	20*	143	20*	117
			40*	103	39*	125	21*	105	21*	88
			40*	118	39*	133	20*	133	20*	88
			40*	117	39*	143	20*	179	20*	161
			40*	175	39*	156	20*	144	20*	128
			36.2(11)	128	35.1(11)	135	18.3(11)	120	18.3(11)	97
1	60	20	41	250	40	250	21	250	21	250
			32	250	31	250	16*	236	16	250
			31	250	30	250	15	250	15	250
			33	250	32	250	17	250	17*	197
			31	250	30	250	15	250	15	250
			30	250	28	250	16	250	16	250
			39	250	38	250	20	250	20	250
			40	250	39	250	21	250	21	250
			40	250	39	250	20	250	20	250
			40	250	39	250	20	250	20	250
			40	250	39	250	20	250	20	250
			36.1(0)	250	35.0(0)	250	18.3(1)	249	18.3(1)	245

Table 7 continued

Instance			Base		Holding		1 speed		*n* speeds	
config	I	C	obj	t(s)	obj	t(s)	obj	t(s)	obj	t(s)
2	15	15	22*	3	22*	2	0*	5	0*	4
			17*	3	17*	3	0*	5	0*	3
			14*	3	14*	1	0*	3	0*	3
			18*	3	18*	3	0*	7	0*	6
			17*	2	17*	2	0*	3	0*	3
			19*	3	19*	2	0*	6	0*	2
			21*	2	21*	2	0*	5	0*	5
			21*	2	21*	2	0*	4	0*	5
			21*	4	21*	2	0*	3	0*	3
			21*	3	21*	2	0*	4	0*	4
			21*	3	21*	2	0*	3	0*	2
			19.3(11)	3	19.3(11)	2	0.0(11)	4	0.0(11)	4
2	30	30	24*	94	24*	48	0*	217	0*	150
			19*	101	19*	42	0*	104	0*	147
			16*	54	15*	38	0*	158	0*	125
			20*	97	20*	37	1	250	1*	76
			22*	105	22*	65	2	250	2*	101
			19*	96	19*	40	0*	184	0*	147
			24*	107	24*	58	0*	132	0*	217
			24*	137	24*	79	0	250	0*	74
			25*	100	25*	54	1*	125	1*	157
			24*	102	24*	70	0*	182	0*	92
			24*	100	24*	70	0*	183	0	250
			21.9(11)	99	21.8(11)	55	0.4(8)	185	0.4(10)	140
2	45	45	24	250	24	250	1	250	0	250
			23	250	23	250	4	250	3	250
			17	250	17*	212	0	250	0	250
			19	250	19*	235	1	250	0	250
			21	250	21	250	2	250	1	250
			21	250	21	250	1	250	0	250
			23	250	23	250	1	250	0	250
			24	250	24	250	1	250	0	250
			24	250	24	250	1	250	0	250
			23	250	23	250	0	250	0	250
			23	250	23	250	0	250	0	250
			22.0(0)	250	22.0(2)	245	1.1(0)	250	0.4(0)	250
2	60	60	24	250	24	250	10	250	0	250
			19	250	20	250	11	250	0	250
			19	250	19	250	1	250	0	250
			19	250	19	250	0	250	0	250
			20	250	20	250	1	250	0	250

Table 7 continued

Instance			Base		Holding		1 speed		*n* speeds	
config	I	C	obj	t(s)	obj	t(s)	obj	t(s)	obj	t(s)
			21	250	21	250	16	250	0	250
			24	250	24	250	1	250	0	250
			24	250	24	250	4	250	0	250
			24	250	24	250	1	250	0	250
			24	250	24	250	0	250	0	250
			24	250	24	250	0	250	0	250
			22.0(0)	250	22.1(0)	250	4.1(0)	250	0.0(0)	250

Table 8 Iterated Local Search, comprehensive results

Instace			Base			Holding			*n* speeds		
config	I	C	obj	LB	t(s)	obj	LB	t(s)	obj	LB	t(s)
0	60	20	4	4	2	3	3	1	0	0	0
			4	3	1	4	2	1	0	0	0
			5	4	1	4	2	2	0	0	0
			7	7	1	6	5	0	0	0	0
			3	2	1	3	1	2	0	0	0
			4	3	2	4	1	1	0	0	0
			4	4	1	3	2	2	0	0	0
			5	5	2	5	4	1	0	0	0
			5	5	1	5	4	2	0	0	0
			5	5	2	4	4	1	0	0	0
			4	4	1	3	3	2	0	0	1
			4.5	4.2	1.4	4.0	2.8	1.4	0.0	0.0	0.1
1	60	20	41	41	2	40	40	1	25	21	1
			32	32	1	31	31	1	16	16	1
			34	31	0	32	30	1	16	15	1
			35	33	1	34	32	1	18	17	1
			32	31	1	31	30	1	16	15	1
			30	30	1	29	28	1	16	16	2
			40	39	1	39	38	1	22	20	1
			40	40	2	39	39	2	25	21	1
			41	40	1	40	39	1	22	20	1
			40	40	2	39	39	2	23	20	1
			40	40	1	39	39	1	23	20	1
			36.8	36.1	1.2	35.7	35.0	1.2	20.2	18.3	1.1
2	60	60	25	24	2	25	24	1	0	0	0
			20	19	1	20	20	2	0	0	0
			20	19	1	20	19	1	0	0	0
			19	19	1	19	19	1	0	0	1
			20	20	2	20	20	1	0	0	0
			22	21	1	22	21	1	0	0	0
			24	24	1	24	24	1	0	0	0
			24	24	2	24	24	2	0	0	0
			24	24	1	24	24	1	0	0	0
			24	24	1	24	24	1	0	0	0
			24	24	1	24	24	2	0	0	1
			22.4	22.0	1.3	22.4	22.1	1.3	0.0	0.0	0.2

Table 9 The tuning parameters used in the ILS

$l1$	30	P_I	3
$l2$	50	P_C	1
$g1$	50	μ_1	0.25
$g2$	90	μ_2	0.5
$g3$	100	h_1	3
$g4$	200	h_2	0
N_{LS}	50	$tabu\_penalty$	150
N_{ILS}	300	$tabu\_length$	5
N_{S1}	30	th_{max}	400
N_{S2}	50		

References

Ahmadi, R. H., & Kouvelis, P. (1994). Staging problem of a dual delivery pick-and-place machine in printed circuit card assembly. *Operations Research, 42*(1), 81–91.

Bozma, H. I., & Kalalıoğlu, M. (2012). Multirobot coordination in pick-and-place tasks on a moving conveyor. *Robotics and Computer-Integrated Manufacturing, 28*(4), 530–538.

Daoud, S., Yalaoui, F., Amodeo, L., & Chehade, H. (2010). Ant colony algorithms for robotic systems optimization. In *International conference on industrial engineering and manufacturing*.

Ferrari, G., Ferrarini, L., Petretti, A., & Pizzi, E. (2015). Modeling and design of an optimal line manager of a packaging system with MILP. In *Industrial electronics society, IECON 2015-41st annual conference of the IEEE* (pp. 005050–005056). IEEE.

Ho, W., & Ji, P. (2009). An integrated scheduling problem of PCB components on sequential pick-and-place machines: Mathematical models and heuristic solutions. *Expert Systems with Applications, 36*(3), 7002–7010.

Huang, Y., Chiba, R., Arai, T., Ueyama, T., & Ota, J. (2012). Part dispatching rule-based multi-robot coordination in pick-and-place task. In *2012 IEEE international conference on robotics and biomimetics (ROBIO)* (pp. 1887–1892). IEEE.

Huang, Y., Chiba, R., Arai, T., Ueyama, T., & Ota, J. (2015). Robust multi-robot coordination in pick-and-place tasks based on part-dispatching rules. *Robotics and Autonomous Systems, 64*, 70–83.

Humbert, G., Pham, M. T., Brun, X., Guillemot, M., & Noterman, D. (2015). Comparative analysis of pick & place strategies for a multi-robot application. In *2015 IEEE 20th conference on emerging technologies & factory automation (ETFA)* (pp. 1–8). IEEE.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2019). Iterated local search: Framework and applications. In *Handbook of metaheuristics* (pp. 129–168). Springer.

Mattone, R., Adduci, L., & Wolf, A. (1998). Online scheduling algorithms for improving performance of pick-and-place operations on a moving conveyor belt. In *Proceedings of the 1998 IEEE international conference on robotics and automation, 1998* (Vol. 3, pp. 2099–2105). IEEE.

Mattone, R., Divona, M., & Wolf, A. (2000). Sorting of items on a moving conveyor belt. Part 2: Performance evaluation and optimization of pick-and-place operations. *Robotics and Computer-Integrated Manufacturing, 16*(2), 81–90.

Pellicciari, M., Berselli, G., Leali, F., & Vergnano, A. (2013). A method for reducing the energy consumption of pick-and-place industrial robots. *Mechatronics, 23*(3), 326–334.

Pizzi, E., Bouchrit, A., Petretti, A., & Ferrarini, L. (2016). Performance improvement for online schedulers for packaging systems. In *2016 IEEE international conference on automation science and engineering (CASE)* (pp. 1243–1248). IEEE.

Schettini, T. (2017). Optimizing the scheduling of a pick and place robotic system. Master's thesis, Politecnico di Milano.