Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 134 (2025) 508–513

www.elsevier.com/locate/procedia

Proceedings of the 58th CIRP Conference on Manufacturing Systems 2025

A Constraint Programming Approach for the Multi-Robot Multibay Unit Load Pre-marshalling Problem

Thomas Bömer[a,*], Max Disselnmeyer[b], Anne Meyer[b]

[a]TU Dortmund University, Leonhard-Euler-Straße 5, 44227 Dortmund, Germany
[b]Karlsruhe Institute of Technology, Zirkel 2, 76131 Karlsruhe, Germany

* Corresponding author. *E-mail address:* thomas.boemer@tu-dortmund.de

Abstract

Autonomous mobile robots are increasingly employed to automate intralogistics operations. This work addresses the multi-robot multibay unit load pre-marshalling problem, wherein robots optimize warehouse processes during peak times by proactively reshuffling inventory based on anticipated demand during idle periods. The problem involves reshuffling a warehouse according to priority classes using multiple robots, necessitating coordinated tour planning between reshuffling moves while respecting dependencies. A stage-based constraint programming model is introduced that determines reshuffling moves and plans robot tours while considering dependencies between moves. Our objective is to minimize the makespan for the minimum number of moves. The results demonstrate the limitations of the constraint programming model and offer valuable insights for future heuristic approaches.

Keywords: pre-marshalling; warehouse optimization; multi-robot

1. Introduction

Block stacking storage systems involve arranging unit loads (e.g. pallets) on the floor and possibly stacking them vertically. A shared storage strategy can be used to increase the storage density. However, a shared storage strategy can cause blockages in the system. A unit load is deemed *blocking* if a unit load of a lower priority class hinders access to a unit load of a higher priority class. Priority classes can be assigned based on factors like the expected retrieval time. To clear blockages, robots can perform pre-marshalling operations in idle times to prepare the warehouse for future demand. This ensures unit loads can be retrieved from the warehouse without the need for reshuffling during peak times, improving operational efficiency, reducing delays, and optimizing resource utilization.

[1] introduce this problem as the unit load pre-marshalling problem (UPMP) for a single bay, optimizing for the minimal number of reshuffling moves. [2] extend the approach of [1] to warehouses with multiple bays as the multibay unit load pre-marshalling problem (MUPMP). The authors propose a similar two-stage approach in both works. First, the authors fix the access direction and access point for each storage slot in the warehouse. *Access points* are situated in the aisle space in front of the bays. Storage slots accessed from the same access point are referred to as a *virtual lane*. Afterward, a tree-search procedure searches primarily for the minimal number of reshuffling moves. As tie-breaking criteria, the tree-search selects the move with the shortest loaded move time. We distinguish between the *loaded* and *unloaded move time*: The loaded travel time considers the travel time between the pick-up and drop-off access point of the same move, the unloaded move time describes the travel time between the drop-off access point move and pick-up access point of the next move.

[3] build upon the approach of [2] by introducing a sequential method that assigns reshuffling moves—determined by a tree-search heuristic—to robots based on move dependencies, using a mixed-integer programming formulation. A *dependency* arises when two moves involve the same virtual lane, as only one robot can access a virtual lane at a time. Consequently, pick-up and drop-off operations must be carefully scheduled to prevent conflicts.

However, the sequential assignment in [3] does not account for whether the moves identified by the tree-search heuristic are well-suited for efficient robot scheduling with the goal of minimizing makespan. In some cases, slightly longer loaded moves might be strategically preferable to reduce dependencies and improve overall efficiency. Additionally, selecting the shortest loaded moves at every step does not necessarily lead to the optimal solution, as a slightly longer move might better align with subsequent tasks and minimize idle time.

To address these limitations, our proposed approach integrates reshuffling move search and robot tour planning into a single, stage-based constraint programming model. Unlike the sequential approach, our approach jointly optimizes both aspects to ensure that move selection also considers its impact on makespan.

Our approach first applies access direction fixing, adapted from [1], to define virtual lanes within the warehouse. Then, using constraint programming, we simultaneously determine reshuffling moves and schedule robot tours while respecting move dependencies. The primary objective is to minimize the number of reshuffling moves, based on the assumption that pick-up and drop-off handling times account for a significant portion of total operational time. As a secondary objective, we minimize the makespan of the entire pre-marshalling operation, ensuring efficient execution in multi-robot settings.

Our main contributions with this paper are:

- We present a constraint programming approach for the multi-robot multibay unit load pre-marshalling problem (MR-MUPMP).
- We demonstrate the limitations of this approach when used as a standalone solution.
- We derive insights for future heuristic approaches.

The outline of this paper is as follows. Section 2 introduces the optimization problem. Section 3 provides an overview of related work. Section 4 describes the proposed solution approach. Section 5 evaluates the approach in computational experiments. Section 6 concludes the findings and points out further research directions.

2. Problem

The multi-robot multibay unit load pre-marshalling problem (MR-MUPMP) involves sorting unit loads in a block-stacking warehouse using multiple robots across multiple bays until all blockages are cleared. During the pre-marshalling, no unit load enters or leaves the warehouse. A unit load is deemed *blocking* if it hinders access to a unit load with a higher priority class. Priority classes can for example be assigned based on the expected retrieval time. The warehouse consists of a set of interconnected bays as shown in Figure 1. Each bay can be accessed from up to four access directions. Each bay has a grid of storage slots. A storage slot can be identified by its column (x-axis), row (y-axis), and tier (z-axis). Unit loads are placed inside a bay in a shared storage policy. No empty gaps inside the bay are allowed. Each unit load has a priority class assigned. The storage slots can be accessed from access points in the aisle in front of each bay. The pre-marshalling process is operated by robots. A robot can only access the outermost unit load in a bay, and a

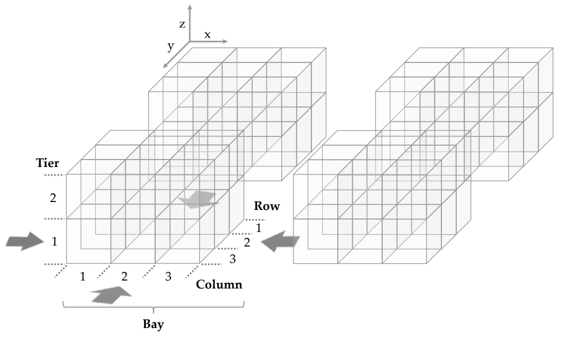

Fig. 1. Representation of the multibay block stacking warehouse [2].

robot can only carry a single unit load at a time. Moreover, only one robot can access an access point for a pick-up or drop-off process at a time. Each reshuffling move comprises: a loaded move, an unloaded move, pick-up/drop-off handling, and waiting time. The loaded move time is the time required to travel the distance between the pick-up access point and the drop-off access point; the unloaded move time is the time required to travel to the next move's pick-up access point; the handling time is the time it takes for the pick-up or drop-off process; and the waiting time is the time a robot has to wait for another robot to finish a process to respect dependencies between moves.

3. Related work

The pre-marshalling of unit loads is closely related to the container pre-marshalling problem (CPMP). Most studies minimize reshuffling moves using integer programming [4] or constraint programming [5, 6]. [6] find constraint programming superior in runtime over integer programming. While move count is the primary objective, makespan and travel time are often overlooked. [7] introduce an integer programming model minimizing crane travel time. [8] extend the CP5 model [6], proposing the MCT model for CPMP with a travel time objective, which outperforms state-of-the-art integer programming in runtime and solution quality. They also study CPMP under limited operation time. However, CPMP remains limited to single-crane, single-block settings, as container yards typically operate with one crane per block. The operation of more than one crane inside a block is studied in the remarshalling problem. For the remarshalling problem different problem variants exist.

The intra-block remarshalling problem is explored in [9], focusing on relocating containers without blockages while considering crane interdependencies. Both adopt a two-phase simulated annealing approach for target configuration and crane scheduling. A two-crane variation is studied in [10], where one loads vessels and the other moves containers from trucks. Their two-stage algorithm heuristically selects target stacks and optimizes movement order. [11] integrates preparatory and main jobs using a look-ahead scheduling approach, prioritizing impactful moves and adapting to real-time conditions.

4. Solving the MR-MUPMP

4.1. Step 1: Access direction fixing

Before reshuffling the warehouse, we define a fixed access direction for each storage slot. This procedure splits the bays into virtual lanes. A virtual lane includes all storage slots that are accessed from the same access point. A mixed integer model selects the access direction for each storage location so that there are initially as few blocking loads as possible. A virtual lane representation of the bay results from this step. Please refer to [1] and [2] for an in-depth explanation. Figure 2 shows the result of this step with four potential access directions. On the left, colors indicate the selected access direction for each storage slot of the bay. The right side shows the virtual lane representation of the same bay. The virtual lane representation serves as input for the model in Subsection 4.2. This step can be neglected if only a single access direction is allowed.

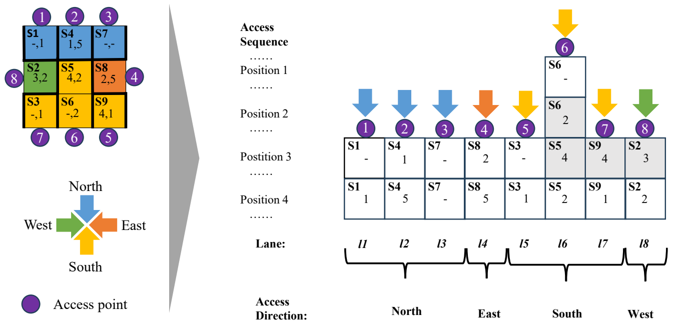

Fig. 2. Virtual lane representation. [2]

4.2. Step 2: Constraint Programming Model

To solve the MR-MUPMP, we extend a constraint programming approach proposed by [6] for solving the pre-marshalling problem while minimizing the number of reshuffling moves. First, we construct a constraint programming model that allows a limited number of reshuffling moves, starting from a lower bound. We use the lower bound proposed by [1]. If no solution is found after a complete search, an additional reshuffling move until a solution is identified. The authors introduce multiple model versions. We opt for CP5 as it proved to be the most efficient. We extend the CP5 by [6] to the MR-MUPMP-CP:

Each stack of containers in the original model equates to a virtual lane and each tier equates to a position in the access sequence of the virtual lane. To ensure comparability, the variable names used are matched to the original model. Let $\bar{p}$ denote the lowest priority class and $\mathcal{P} = \{1, ..., \bar{p}\}$ be the set of priority classes. Let m_p be the number of unit loads with the priority class $p \in \mathcal{P}$. Let $\bar{s}$ denote the number of virtual lanes and $\bar{t}$ represent the number of positions in each virtual lane s. Accordingly, $\mathcal{S} = \{1, ..., \bar{s}\}$ is the set of virtual lanes and $\mathcal{T}_s = \{1, ..., \bar{t}\}$ is the set of positions for virtual lane $s \in \mathcal{S}$. A slot is defined by $s \in \mathcal{S}$ and $t \in \mathcal{T}_s$. Let $d_{s,s'}$ be the time to travel between virtual lane $s \in \mathcal{S}$ and $s' \in \mathcal{S}$. Let d^h be the handling time to perform a pick-up or drop-off process. Let $\bar{k}$ be the permissible number of moves in the model. $\mathcal{K}$ is the set of stages, defined as $\mathcal{K} = \{1, ..., \bar{k}\}$. Stage 0 signifies the initial layout. $\mathcal{K}^0 = \{0, 1, ..., \bar{k}\}$ is the set of stages, including the initial layout. The parameter $f_{s,t}$ defines the initial layout. $f_{s,t}$ is set to the priority class of the unit load positioned in virtual lane $s \in \mathcal{S}$ and position $t \in \mathcal{T}_s$. Let $\bar{r}$ be the number of robots and $\mathcal{R} = \{1, ..., \bar{r}\}$ be the set of robots. The parameter $l^r_{s,1}$ is 1, if robot $r \in \mathcal{R}$ is initially placed in virtual lane $s \in \mathcal{S}$, else 0. The variables are introduced in Table 1.

Table 1. Variable Sets

Variable	Description
$x^k_{s,t}$	p, if a unit load with priority p is placed in slot (s,t) during stage k, else 0. $\forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K}^0$
$\delta^k_{s,t}$	1, if there is a unit load in slot (s,t) during stage k, else 0. $\forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K}^0$
$y^{k,r}_{s,t}$	If a unit load is moved to slot (s,t) by robot r during stage k, else 0. $\forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K}, \forall r \in \mathcal{R}$
$z^{k,r}_{s,t}$	1, if a unit load is removed from slot (s,t) by robot r during stage k, else 0. $\forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K}, \forall r \in \mathcal{R}$
$w^k_{s,t}$	1, if there is a blocking unit load in slot (s,t) during stage k, else 0. $\forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K}$
$\zeta^{k,r}$	1, if robot r moves a unit load in stage k, else 0. $\forall k \in \mathcal{K}, \forall r \in \mathcal{R}$
$h^{k,r}_{s,t}$	1, if robot r is in slot (s,t) at the start of stage k, else 0. $\forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K}^0, \forall r \in \mathcal{R}$
$g^{k,r}_{s,t}$	1, if robot r is in slot (s,t) at the end of stage k, else 0. $\forall s \in \mathcal{S}, \forall k \in \mathcal{K}^0, \forall r \in \mathcal{R}$
$v^{k,r}_s$	Start time of the pick-up process in virtual lane s by robot r during stage k. $\forall s \in \mathcal{S}, \forall k \in \mathcal{K}^0, \forall r \in \mathcal{R}$
$u^{k,r}_s$	End time of the drop-off process in virtual lane s by robot r during stage k. $\forall s \in \mathcal{S}, \forall k \in \mathcal{K}^0, \forall r \in \mathcal{R}$
$b^{k,r}$	Waiting time by robot r during stage k. $\forall k \in \mathcal{K}^0, \forall r \in \mathcal{R}$
u_{max}	End time of the last drop-off performed.
$\alpha^{k,k',r,r'}_s$	1, if robot r in stage k and robot r' in stage k' remove a unit load from the same virtual lane s, else 0. $\forall s \in \mathcal{S}, \forall k \in \mathcal{K}, \forall k' \in \{k, ..., \bar{k}\}, \forall r, r' \in \mathcal{R} : r \neq r'$
$\beta^{k,k',r,r'}_s$	1, if robot r in stage k and robot r' in stage k' move a unit load to the same virtual lane s, else 0. $\forall s \in \mathcal{S}, \forall k \in \mathcal{K}, \forall k' \in \{k, ..., \bar{k}\}, \forall r, r' \in \mathcal{R} : r \neq r'$
$\eta^{k,k',r,r'}_s$	1, if robot r in stage k moves a unit load to virtual lane s and robot r' in stage k' removes a unit load from the virtual lane s, else 0. $\forall s \in \mathcal{S}, \forall k \in \mathcal{K}, \forall k' \in \{k, ..., \bar{k}\}, \forall r, r' \in \mathcal{R} : r \neq r'$
$\theta^{k,k',r,r'}_s$	1, if robot r in stage k removes a unit load to virtual lane s and robot r' in stage k' moves a unit load to the virtual lane s, else 0. $\forall s \in \mathcal{S}, \forall k \in \mathcal{K}, \forall k' \in \{k, ..., \bar{k}\}, \forall r, r' \in \mathcal{R} : r \neq r'$

MR-MUPMP-CP

$$min \quad u_{max} \tag{4.1}$$

Subject to:

$$x^0_{s,t} = f_{s,t} \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s \tag{4.2}$$

$$|\{x^k_{s,t} : s \in \mathcal{S}, t \in \mathcal{T}_s, x^k_{s,t} = p\}| = m_p \quad p \in \mathcal{P}^0, k \in \mathcal{K} \tag{4.3}$$

$$\sum_{r\in\mathcal{R}} \sum_{s\in\mathcal{S}} \sum_{t\in\mathcal{T}_s} y^{k,r}_{s,t} = 1 \quad \forall k \in \mathcal{K} \tag{4.4}$$

$$x^k_{s,t} \leq \bar{p} \cdot \delta^k_{s,t} \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K}^0 \tag{4.5}$$

$$\delta^k_{s,t} \leq x^k_{s,t} \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K}^0 \tag{4.6}$$

$$x^{k-1}_{s,t} \leq x^k_{s,t} + \bar{p}\,(1 - \delta^k_{s,t}) \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K} \tag{4.7}$$

$$x^k_{s,t} \leq x^{k-1}_{s,t} + \bar{p}\,(1 - \delta^{k-1}_{s,t}) \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K} \tag{4.8}$$

$$\sum_{r\in\mathcal{R}} y^{k,r}_{s,t} + \delta^{k-1}_{s,1} \leq 1 \quad \forall s \in \mathcal{S}, \forall k \in \mathcal{K} \tag{4.9}$$

$$\sum_{r\in\mathcal{R}} y^{k,r}_{s,t} \leq \delta^{k+1}_{s,t} \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}, \forall k \in \mathcal{K} \setminus \{\bar{k}\} \tag{4.10}$$

$$\sum_{r\in\mathcal{R}} \sum_{s\in\mathcal{S}} \sum_{t\in\mathcal{T}_s} z^{k,r}_{s,t} = 1 \quad \forall k \in \mathcal{K} \tag{4.11}$$

$$\delta^k_{s,t} + \sum_{r\in\mathcal{R}} z^{k,r}_{s,t} = \sum_{r\in\mathcal{R}} y^{k,r}_{s,t} + \delta^{k-1}_{s,t} \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K} \tag{4.12}$$

$$\sum_{r\in\mathcal{R}} \left(y^{k,r}_{s,t+1} + z^{k,r}_{s,t}\right) + \delta^{k-1}_{s,t+1} \leq \delta^{k-1}_{s,t} \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s \setminus \{\bar{t}\}, \forall k \in \mathcal{K} \tag{4.13}$$

$$\sum_{r\in\mathcal{R}} \left(z^{k+1,r}_{s,\bar{t}} + y^{k,r}_{s,\bar{t}}\right) \leq \delta^k_{s,\bar{t}} \quad \forall s \in \mathcal{S}, \forall k \in \mathcal{K} \setminus \{\bar{k}\} \tag{4.14}$$

$$\sum_{r\in\mathcal{R}} z^{1,r}_{s,\bar{t}} \leq \delta^0_{s,\bar{t}} \quad \forall s \in \mathcal{S} \tag{4.15}$$

$$\sum_{t\in\mathcal{T}_s} \sum_{r\in\mathcal{R}} \left(y^{k,r}_{s,t} + z^{k+1,r}_{s,t}\right) \leq 1 \quad \forall s \in \mathcal{S}, \forall k \in \mathcal{K} \setminus \{\bar{k}\} \tag{4.16}$$

$$x^k_{s,t+1} \leq x^k_{s,t} + \bar{p} \cdot w^k_{s,t+1} \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s \setminus \{\bar{t}\}, \forall k \in \mathcal{K} \tag{4.17}$$

$$w^k_{s,t} + \delta^k_{s,t+1} \leq w^k_{s,t+1} + 1 \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s \setminus \{\bar{t}\}, \forall k \in \mathcal{K} \tag{4.18}$$

$$w^k_{s,t} \leq \delta^k_{s,t} \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K} \tag{4.19}$$

$$w^k_{s,1} = 0 \quad \forall s \in \mathcal{S}, \forall k \in \mathcal{K} \tag{4.20}$$

$$x^k_{s,t} + 1 \leq x^k_{s,t+1} + (\bar{p} + 1) \cdot (1 - w^k_{s,t+1} + w^k_{s,t})$$
$$\forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s \setminus \{\bar{t}\}, \forall k \in \mathcal{K} \tag{4.21}$$

$$\sum_{s\in\mathcal{S}} \sum_{t\in\mathcal{T}_s} w^k_{s,t} + k \leq \bar{k} \quad \forall k \in \mathcal{K} \tag{4.22}$$

$$\sum_{s\in\mathcal{S}} \sum_{t\in\mathcal{T}_s} z^{k,r}_{s,t} = \sum_{s\in\mathcal{S}} \sum_{t\in\mathcal{T}_s} y^{k,r}_{s,t} \quad \forall k \in \mathcal{K}, \forall r \in \mathcal{R} \tag{4.23}$$

$$\zeta^{k,r} = \sum_{s\in\mathcal{S}} \sum_{t\in\mathcal{T}_s} z^{k,r}_{s,t} \quad \forall k \in \mathcal{K}, \forall r \in \mathcal{R} \tag{4.24}$$

$$h^{0,r}_{s,1} = l^r_{s,1} \quad \forall r \in \mathcal{R} \tag{4.25}$$

$$h^{k,r}_{s,t} = z^{k,r}_{s,t} \cdot \zeta^{k,r} + h^{k-1,r}_{s,t}(1 - \zeta^{k,r}) \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K}, \forall r \in \mathcal{R} \tag{4.26}$$

$$\sum_{s\in\mathcal{S}} \sum_{t\in\mathcal{T}_s} h^{k,r}_{s,t} = 1 \quad \forall k \in \mathcal{K}, \forall r \in \mathcal{R} \tag{4.27}$$

$$g^{0,r}_{s,1} = l^r_{s,1} \quad \forall r \in \mathcal{R} \tag{4.28}$$

$$g^{k,r}_{s,t} = y^{k,r}_{s,t} \cdot \zeta^{k,r} + g^{k-1,r}_{s,t}(1 - \zeta^{k,r}) \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T}_s, \forall k \in \mathcal{K}, \forall r \in \mathcal{R} \tag{4.29}$$

$$\sum_{s\in\mathcal{S}} \sum_{t\in\mathcal{T}_s} g^{k,r}_{s,t} = 1 \quad \forall k \in \mathcal{K}, \forall r \in \mathcal{R} \tag{4.30}$$

$$v^{k,r} = u^{k-1,r} + b^{k,r} + \sum_{s,s'\in\mathcal{S}} \sum_{t,t'\in\mathcal{T}_s} \left(g^{k-1,r}_{s,t} \cdot h^{k,r}_{s',t'} \cdot d_{s,s'}\right) \forall k \in \mathcal{K}, \forall r \in \mathcal{R} \tag{4.31}$$

$$u^{k,r} = v^{k,r} + 2 \cdot d^h + \sum_{s,s'\in\mathcal{S}} \sum_{t,t'\in\mathcal{T}_s} \left(g^{k,r}_{s,t} \cdot h^{k,r}_{s',t'} \cdot d_{s,s'}\right) \quad \forall k \in \mathcal{K}, \forall r \in \mathcal{R} \tag{4.32}$$

$$u^{k,r} \leq u_{max} \quad \forall k \in \mathcal{K}, \forall r \in \mathcal{R} \tag{4.33}$$

$$\alpha^{k,k'r,r'}_s = \sum_{t\in\mathcal{T}_s} \left(z^{k,r}_{s,t} \cdot z^{k',r'}_{s,t}\right)$$
$$\forall s \in \mathcal{S}, \forall k \in \mathcal{K}, \forall k' \in \{k, ..., \bar{k}\}, \forall r, r' \in \mathcal{R} : r \neq r' \tag{4.34}$$

$$\beta^{k,k'r,r'}_s = \sum_{t\in\mathcal{T}_s} \left(y^{k,r}_{s,t} \cdot y^{k',r'}_{s,t}\right)$$
$$\forall s \in \mathcal{S}, \forall k \in \mathcal{K}, \forall k' \in \{k, ..., \bar{k}\}, \forall r, r' \in \mathcal{R} : r \neq r' \tag{4.35}$$

$$\eta^{k,k'r,r'}_s = \sum_{t\in\mathcal{T}_s} \left(y^{k,r}_{s,t} \cdot z^{k',r'}_{s,t}\right)$$
$$\forall s \in \mathcal{S}, \forall k \in \mathcal{K}, \forall k' \in \{k, ..., \bar{k}\}, \forall r, r' \in \mathcal{R} : r \neq r' \tag{4.36}$$

$$\theta^{k,k'r,r'}_s = \sum_{t\in\mathcal{T}_s} \left(z^{k,r}_{s,t} \cdot y^{k',r'}_{s,t}\right)$$
$$\forall s \in \mathcal{S}, \forall k \in \mathcal{K}, \forall k' \in \{k, ..., \bar{k}\}, \forall r, r' \in \mathcal{R} : r \neq r' \tag{4.37}$$

$$\alpha^{k,k'r,r'}_s (v^{k,r} + d^h) \leq v^{k',r'} \forall s \in \mathcal{S}, \forall k, k' \in \mathcal{K}, \forall r, r' \in \mathcal{R} \tag{4.38}$$

$$\beta^{k,k'r,r'}_s (u^{k,r} + d^h) \leq u^{k',r'} \forall s \in \mathcal{S}, \forall k, k' \in \mathcal{K}, \forall r, r' \in \mathcal{R} \tag{4.39}$$

$$\eta^{k,k'r,r'}_s \cdot u^{k,r} \leq v^{k',r'} \forall s \in \mathcal{S}, \forall k, k' \in \mathcal{K}, \forall r, r' \in \mathcal{R} \tag{4.40}$$

$$\theta^{k,k'r,r'}_s (v^{k,r} + 2 \cdot d^h) \leq u^{k',r'} \forall s \in \mathcal{S}, \forall k, k' \in \mathcal{K}, \forall r, r' \in \mathcal{R} \tag{4.41}$$

The objective function (4.1) minimizes the makespan. Constraints (4.2) assign the initial layout to the virtual lanes of the warehouse. Constraints (4.3) ensure that the number of unit loads per priority class stays consistent for each stage. The number of moves per stage is restricted to one by Constraints (4.4). Constraints (4.5) and (4.6) link the variables $x^k_{s,t}$ and $\delta^k_{s,t}$. $\delta^k_{s,t}$ is 0 if $x^k_{s,t}$ is 0, else $\delta^k_{s,t}$ is 1. Unit loads that are not moved are fixed to their position by Constraints (4.7) and (4.8). Constraints (4.9) impose that a unit load can only be moved to a slot that was empty in the previous stage. Constraints (4.10) prevent the unit loads from being moved in two consecutive stages because the two moves could have been performed in one move. This rule is also called transitive move avoidance [12]. Constraints (4.11) sets the number of unity loads removed from the virtual lanes to 1 for each stage. Constrains (4.12) make sure that if a unit load is in a slot. It has been there in the previous stage or was moved there in the current stage. (4.13) impose that a unit load can be removed from a certain slot in a stage k only if the respective slot was occupied and the slot in front $(s, t + 1)$ was empty during the previous stage. Constraints (4.14) impose that a unit load can only be moved to a slot (s, t) that was empty during the previous stage. It also requires that the slot behind $(s, t - 1)$ was occupied. Constraints (4.15) do the same for the first stage. Constraints (4.16) impose that when a unit load is moved to a certain virtual lane, no unit load can be removed from that virtual lane in the next stage. Constraints (4.17) to (4.21) determine unit loads as blocking. Constraints (4.17) express that a unit load that is in front of another unit load with a higher priority class is a blocking unit load. Constraints (4.18) require that if there is a blocking unit load on slot (s, t) and slot $(s, t + 1)$ is occupied, there is also a blocking unit load on slot $(s, t + 1)$. Constraints (4.19) impose that empty slots can not be blocking. Constraints (4.20) define that the first position of a virtual lane is never blocking because access is always granted. Constraints (4.21) determine that if there is no blocking unit load in slot (s, t) and there is no higher priority unit load in the slot in front of it, then there is no blocking unit load in slot $(s, t+1)$. Constraints (4.22) impose that the number of blocking unit loads at each stage cannot be greater than the number of following stages. Constraints (4.23) enforce that the same robot must perform the pick-up and drop-off of the same move. Constraints (4.24) sets $\zeta^{k,r}$ to 1 if a move is performed in stage k by robot r. Constraints (4.25) define the initial start position of each robot in the first stage. Constraints (4.26) impose the start position of a robot to be the same position as in the previous stage if the robot was not used in the previous stage. Constraints (4.27) define that each robot has to be at a defined access point at the start of each stage. Constraints (4.28) define the initial end position of each robot in the first stage. Constraints (4.29) impose the end position of a robot to be the same position as in the previous stage if the robot was not used in the previous stage. Constraints (4.30) define that each robot has to be at a defined access point at the end of each stage. Constraints (4.31) set the start time of the pick-up process and Constraints (4.32) set the end time of the drop-off process. Constraints (4.33) set u_{max} to the latest drop-off end time in the model. Constraints (4.34) to (4.37) track the dependencies between moves. Constraints (4.38) to (4.41) enforce dependencies between moves when two robots access the same virtual lane. Start-start dependencies ensure that a later move's pick-up starts no earlier than the prior move's pick-up plus handling time d^h. End-end dependencies require a later drop-off to finish no earlier than the prior drop-off plus d^h. End-start dependencies ensure a later move's drop-off starts only after the prior move's pick-up ends. Start-end dependencies enforce that a later move's pick-up ends no earlier than the prior move's drop-off start plus $2d^h$.

5. Experiments

All experiments were executed on the Linux-HPC-Cluster (LiDO3) at TU Dortmund on an Intel Xeon E5-4640v4 node with 256 GB RAM. We use Google OR-Tools to implement the network flow model for the access direction fixing and the MR-MUPMP-CP model. The OR-Tools CP-SAT solver solves both models. The maximum overall runtime for the model building and solving time per instance time is limited to 47 hours.

Parameters. In sum, we conducted 240 experiments. The instances were taken from the dataset of [13]. We select ten instances with the seeds zero to nine for each of the following layout configurations: one-tier; bay layout (b) sizes of 2x2 and 3x3; warehouse layout (wh) sizes of 2x2 and 3x3; fill percentages (f %) of 60 %, 80 %, and 90 %; and ten priority classes. A 4x4 bay layout describes bays with four rows and four columns - 16 slots for the one-tier case. A 2x2 warehouse layout denotes a warehouse that consists of 4 bays that are arranged in a quadratic pattern. We refer to a combination of a 4x4 bay layout and a 2x2 warehouse layout as 4x4_2x2. A fill percentage of 80 % indicates that 80 % of the slots per bay are filled with unit loads. Further, we consider two access directions (north and south) and 2 or 3 robots (r). The robots travel at a constant speed of one distance unit per second and operate with a handling time of 30 seconds per pick-up/ drop-off.

Solved instances. The Table 2 shows the number of solved instances (sol.), the number of instances solved with proven optimality (opt.), and the number of reshuffling moves for each instance configuration. The solvability decreases as the number of reshuffling moves increases. This is attributed to the rapid growth of the model size with increasing reshuffling moves. Consequently, optimal solutions can typically be proven only for instances with fewer moves. The maximum number of moves for which a proven optimal solution was found is 12.

Solver and build time. Table 3 shows a significant increase in both mean solver and build times as the number of robots increases, indicating a substantial rise in solution complexity. Notably, the mean time to find the first solution is considerably higher for three robots compared to two. The model build time is another indicator of the increasing solution effort. For a 5x5_2x2 layout and 80 % fill percentage, the mean build time is 275 s for two robots and 435 s for three robots.

Objective over solver time. The mean solver time of the best solution in Table 3 shows that improvements are still found after hours. However, Figure 3 shows that the most significant improvement in the objective function occurs relatively quickly after the first solution is identified.

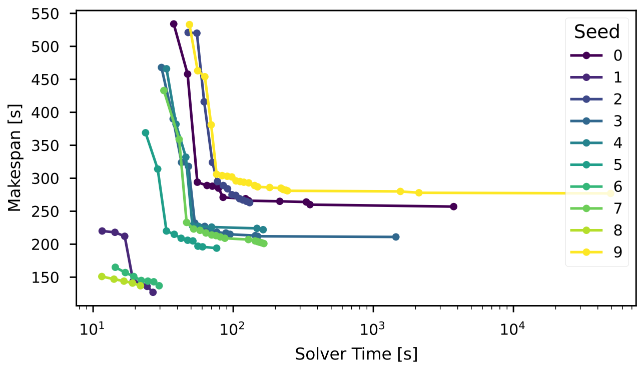

Fig. 3. Objective function value over solver time for each seed of a configuration with 4x4_2x2 layout, 80 % fill percentage, and two robots.

6. Conclusion

In this work, we presented a two-stage approach to solve the multi-robot unit load pre-marshalling problem. The first stage decomposes the warehouse in virtual lanes. The second stage employs a novel constraint programming model to integrate the search for reshuffling moves with tour planning. Our experiments revealed that this integrated approach is computationally challenging. However, the model can improve relatively quickly after the first solution. Therefore, this approach could be valuable for assessing the quality of heuristic solutions.

Future work should investigate approaches to reduce the solution space without reducing the solution quality. Additionally, the impact of access direction fixing on solution quality warrants further study. Adapting this approach to related warehouse problems, such as the buffer reshuffling and retrieval problem [14], should also be considered.

Table 2. Overview of solved instances.

r	b	wh	f %	sol.	opt.	moves mean	moves min	moves max
2	4x4	2x2	60	10	10	5.6	2	10
			80	10	7	10.8	6	16
			90	10	2	18.6	10	26
		3x3	60	10	3	11.6	6	20
			80	8	0	27	18	34
			90	1	0	28	28	28
	5x5	2x2	60	10	1	14.2	8	20
			80	7	0	28.6	24	34
			90	1	0	30	30	30
		3x3	60	10	0	32.8	24	38
			80	0	0	-	-	-
			90	0	0	-	-	-
3	4x4	2x2	60	10	10	5.6	2	10
			80	10	7	10.8	6	16
			90	10	2	18.6	10	26
		3x3	60	10	5	11.6	6	20
			80	8	0	27	18	34
			90	1	0	28	28	28
	5x5	2x2	60	10	4	14.2	8	20
			80	6	0	28.3	24	34
			90	0	0	-	-	-
		3x3	60	4	0	29.5	24	32
			80	0	0	-	-	-
			90	0	0	-	-	-

Table 3. Overview of the mean solver times and mean build times.

r	b	wh	f %	solver time [s] first	solver time [s] best	build time [s] solved	build time [s] all
2	4x4	2x2	60	12	30	15	15
			80	29	5521	32	32
			90	3589	50413	66	66
		3x3	60	120	26656	172	172
			80	20352	84230	527	541
			90	32726	162356	539	947
	5x5	2x2	60	91	17810	106	106
			80	49202	123758	272	275
			90	131264	146517	309	517
		3x3	60	13846	23888	1679	1679
			80	-	-	-	4914
			90	-	-	-	9740
3	4x4	2x2	60	17	46	23	23
			80	44	6376	50	50
			90	13236	41125	105	105
		3x3	60	269	24103	263	263
			80	11807	82376	815	838
			90	91157	94697	845	1485
	5x5	2x2	60	182	13414	162	162
			80	50840	96788	429	435
			90	-	-	-	826
		3x3	60	15889	16599	2142	2142
			80	-	-	-	7798
			90	-	-	-	17899

Acknowledgements

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - project number 276879186 and the Federal Ministry of Economics and Climate Protection of the Federal Republic of Germany and the European Union as part of NextGenerationEU (funding code: 13IK032F and 13IK032I).

References

[1] J. Pfrommer, A. Meyer, and K. Tierney, "Solving the unit-load pre-marshalling problem in block stacking storage systems with multiple access directions," *European Journal of Operational Research*, 2023. [Online]. Available: https://doi.org/10.1016/j.ejor.2023.08.044

[2] J. Pfrommer, T. Bömer, D. Akizhanov, and A. Meyer, "Sorting multibay block stacking storage systems," *arXiv preprint arXiv:2405.04847*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.04847

[3] T. Bömer, N. Koltermann, J. Pfrommer, and A. Meyer, "Sorting multibay block stacking storage systems with multiple robots," in *International Conference on Computational Logistics*. Springer, 2024, pp. 34–48. [Online]. Available: https://doi.org/10.1007/978-3-031-16579-5_27

[4] C. Parreño-Torres, R. Alvarez-Valdes, and R. Ruiz, "Integer programming models for the pre-marshalling problem," *European Journal of Operational Research*, vol. 274, no. 1, pp. 142–154, 2019. [Online]. Available: https://doi.org/10.1016/j.ejor.2018.09.048

[5] A. Rendl and M. Prandtstetter, "Constraint models for the container pre-marshaling problem," in *ModRef 2013: The Twelfth International Workshop on Constraint Modelling and Reformulation*, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:43049743

[6] C. Jiménez-Piqueras, R. Ruiz, C. Parreño-Torres, and R. Alvarez-Valdes, "A constraint programming approach for the premarshalling problem," *European Journal of Operational Research*, vol. 306, no. 2, pp. 668–678, 2023. [Online]. Available: https://doi.org/10.1016/j.ejor.2022.07.042

[7] C. Parreño-Torres, R. Alvarez-Valdes, R. Ruiz, and K. Tierney, "Minimizing crane times in pre-marshalling problems," *Transportation Research Part E: Logistics and Transportation Review*, vol. 137, 2020. [Online]. Available: https://doi.org/10.1016/j.tre.2020.101917

[8] C. Jiménez-Piqueras, C. Parreño-Torres, R. Alvarez-Valdes, and R. Ruiz, "The container premarshalling problem under limited crane time: A constraint programming approach," *Computers & Operations Research*, vol. 166, p. 106635, 2024. [Online]. Available: https://doi.org/10.1016/j.cor.2024.106635

[9] R. Choe, T. Park, M.-S. Oh, J. Kang, and K. R. Ryu, "Generating a rehandling-free intra-block remarshaling plan for an automated container yard," *Journal of Intelligent Manufacturing*, vol. 22, pp. 201–217, 2011. [Online]. Available: https://doi.org/10.1007/s10845-009-0273-y

[10] K. Park, T. Park, and K. R. Ryu, "Planning for remarshaling in an automated container terminal using cooperative coevolutionary algorithms," in *Proceedings of the 2009 ACM symposium on Applied Computing*, 2009, pp. 1098–1105. [Online]. Available: https://doi.org/1529282.1529523

[11] R. Choe, T. S. Kim, T. Kim, and K. R. Ryu, "Crane scheduling for opportunistic remarshaling of containers in an automated stacking yard," *Flexible Services and Manufacturing Journal*, vol. 27, no. 2, pp. 331–349, 2015. [Online]. Available: https://doi.org/10.1007/s10696-013-9186-3

[12] K. Tierney, D. Pacino, and S. Voß, "Solving the pre-marshalling problem to optimality with a* and ida*," *Flexible Services and Manufacturing Journal*, vol. 29, no. 2, pp. 223–259, Jun 2017. [Online]. Available: https://doi.org/10.1007/s10696-016-9246-6

[13] J. Pfrommer and T. Bömer, "Multibay unit-load pre-marshalling problem instances," TU Dortmund, Tech. Rep., 2024. [Online]. Available: https://doi.org/10.5281/zenodo.11093870

[14] M. Disselnmeyer, T. Bömer, J. Pfrommer, and A. Meyer, "The static buffer reshuffling and retrieval problem for autonomous mobile robots," in *International Conference on Computational Logistics*. Springer, 2024, pp. 18–33. [Online]. Available: https://doi.org/10.1007/978-3-031-71993-6_2