Taylor & Francis
Taylor & Francis Group

International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: www.tandfonline.com/journals/tprs20

The multi-AMR buffer storage, retrieval, and reshuffling problem: a mathematical model and hierarchical heuristic

Max Disselnmeyer, Thomas Bömer, Laura Dörr, Bastian Amberg & Anne Meyer

To cite this article: Max Disselnmeyer, Thomas Bömer, Laura Dörr, Bastian Amberg & Anne Meyer (15 Jul 2026): The multi-AMR buffer storage, retrieval, and reshuffling problem: a mathematical model and hierarchical heuristic, International Journal of Production Research, DOI: 10.1080/00207543.2026.2702050

To link to this article: https://doi.org/10.1080/00207543.2026.2702050

Published online: 15 Jul 2026.

INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH
https://doi.org/10.1080/00207543.2026.2702050

The multi-AMR buffer storage, retrieval, and reshuffling problem: a mathematical model and hierarchical heuristic

Max Disselnmeyer, Thomas Bömer, Laura Dörr, Bastian Amberg and Anne Meyer

Institute for Information Management in Engineering (IMI), Karlsruhe Institute of Technology, Karlsruhe, Germany

ABSTRACT

Buffer zones are essential in production systems to decouple sequential processes. In dense floor storage environments, such as space-constrained brownfield facilities, manual operation is increasingly challenged by severe labour shortages and rising operational costs. Automating these zones requires solving the Buffer Storage, Retrieval, and Reshuffling Problem (BSRRP). While previous work has addressed scenarios where the focus is limited to reshuffling and retrieving a fixed set of items, real-world manufacturing necessitates an adaptive approach that also incorporates arriving unit loads. This paper introduces the Multi-AMR BSRRP, coordinating a robot fleet to manage concurrent reshuffling, alongside time-windowed storage and retrieval tasks, within a shared floor area. We formulate a Binary Integer Programming (IP) model to obtain exact solutions for benchmarking purposes. As the problem is NP-hard, rendering exact methods computationally intractable for industrial scales, we propose a hierarchical heuristic. This approach decomposes the problem into an A^* search for task-level sequence planning of unit load placements, and a Constraint Programming (CP) approach for multi-robot coordination and scheduling. Experiments demonstrate orders-of-magnitude computation time reductions compared to the exact formulation. These results confirm the heuristic's viability as responsive control logic for high-density production environments.

ARTICLE HISTORY
Received 27 March 2026
Accepted 29 June 2026

KEYWORDS
Autonomous mobile robots (AMR); buffer management; integer programming; production logistics; hierarchical heuristic

1. Introduction

Rising labour shortages and the need for optimised material flow drive the demand for Autonomous Mobile Robots (AMRs) in production logistics – specifically for the management of buffer zones that decouple sequential production stages (Descartes Systems Group 2023; Pytel et al. 2021). In space-constrained brownfield facilities, legacy infrastructure necessitates dense, floor-level block storage where accessibility is restricted (Andulkar, Le, and Berger 2018). In manual operation, these zones often lead to significant operational delays due to time spent searching for materials. While AMRs offer superior flexibility and productivity compared to traditional vehicles (Fragapane et al. 2021), managing these zones requires solving the Buffer Storage, Retrieval, and Reshuffling Problem (BSRRP). Figure 1 illustrates a representative scenario from the surface coating industry, where automation must integrate into irregular residual spaces surrounding existing machinery.

However, previous research primarily optimised the retrieval of fixed item sets (Disselnmeyer et al. 2024). In contrast, industrial environments require an adaptive approach incorporating arriving unit loads. This paper addresses this gap by introducing a coordinated multi-robot approach for the integrated optimisation of time-windowed storage, retrieval, and reshuffling. Our contributions are as follows:

- *Exact Formulation (EF)* We develop a Binary Integer Programming (IP) model for the Multi-AMR BSRRP. It couples storage, retrieval and reshuffling decisions in dense storage, managing a fleet restricted to perimeter access like conventional forklifts.
- *Complexity Analysis* We establish the computational complexity of the BSRRP problem, referencing a formal proof of NP-hardness via reduction from the Block Relocation Problem (BRP).
- *Hierarchical Heuristic:* We propose a hybrid algorithm combining A^* for task sequencing with Constraint Programming (CP) for precise scheduling. This enables real-time fleet control capable of responding to dynamic production requirements.
- *Validation & Industrial Insights:* We benchmark the heuristic against the exact IP model to quantify optimality gaps. Based on these experiments, we derive Managerial Insights demonstrating the algorithm's ability to maintain a capacity-adaptive breathing topology. This emergent behaviour prioritises

CONTACT Max Disselnmeyer max.disselnmeyer@kit.edu Karlsruhe Institute of Technology, Zirkel 2, Karlsruhe 76131, Germany

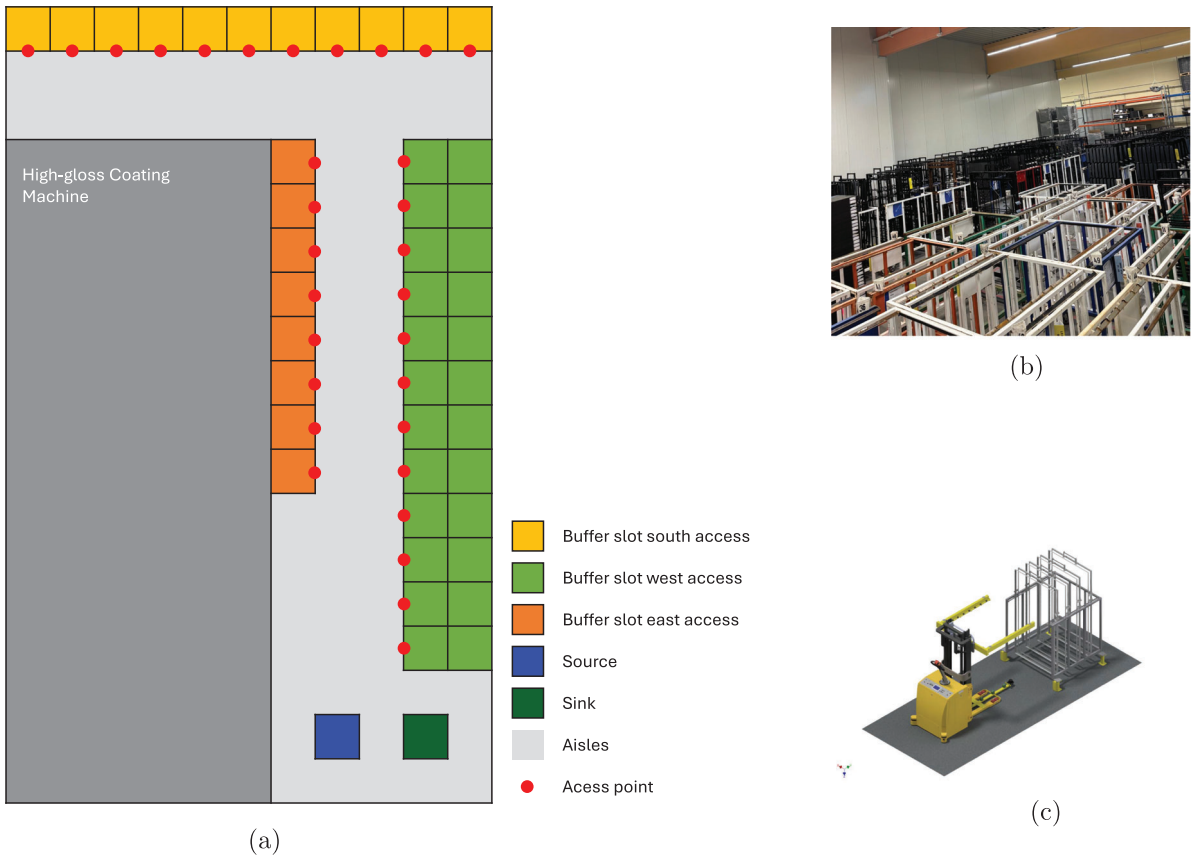

Figure 1. Real-world buffer scenario from the surface coating industry. Automating material flow in such space-constrained brownfield facilities requires solving the Buffer Storage, Retrieval, and Reshuffling Problem (BSRRP) to ensure continuous machine supply. (a) Schematic layout discretizing the residual space around a high-gloss coating machine into static LIFO storage lanes. (b) View from the source/sink area towards the coating machine (background wall on the left), highlighting the storage density. (c) blueAMR with a single unit load consisting of 4 carriers.

accessibility by utilising nearby slots during low demand and dynamically expanding into distant buffer areas during peak congestion. Additionally, we identify a 90% stability threshold as a critical operational limit, beyond which the lack of reshuffling space leads to system brittleness.

The remainder of this paper is structured as follows: Section 2 reviews the relevant literature on the Block Relocation Problem and multi-AMR coordination to situate the BSRRP within the current research landscape. In Section 3, we provide a detailed description of the Multi-AMR BSRRP, including the logical subdivision of the storage space and the underlying operational assumptions. Section 4 presents the formal Binary Integer Programming formulation and discusses the computational complexity of the problem. The proposed hierarchical heuristic, combining A^* with Constraint Programming, is detailed in Section 5. Section 6 evaluates the performance of both the exact and heuristic approaches through extensive computational experiments and discusses managerial insights derived from the results. Finally, Section 7 concludes the paper and outlines avenues for future research.

2. Related work

This section reviews the current state of research and defines the scientific context of the Multi-AMR BSRRP. First, we compare the problem to foundational approaches such as the Block Relocation Problem (BRP) and Multi-Agent Path Finding (MAPF). Second, we outline the specific modelling requirements for autonomous buffers operated by an AMR fleet, focussing on the transition from single AMR models to integrated multi-AMR systems. Third, we discuss existing heuristics for complex logistics tasks. Finally, we identify the research gaps regarding holistic, real-time control in fragmented environments, which forms the basis for the solutions presented in this paper.

2.1. Foundations and related problems

Container retrieval under strict LIFO constraints is formalised as the Block Relocation Problem (BRP) (Kim and Hong 2006; Lersteau and Shen 2022). It relates to the Pre-Marshalling Problem (PMP) and its warehousing variant, the Unit-Load Pre-Marshalling Problem (UPMP) (Bömer, Disselnmeyer, and Meyer 2025; Bömer, Pfrommer et al. 2026; Pfrommer, Meyer, and Tierney 2024). Similarly, the steel industry addresses the Slab Pre-Marshalling Problem (SPMP) with identical LIFO constraints (Ge et al. 2020).

While BRP provides the foundation for the Buffer Reshuffling and Retrieval Problem (BRR) (Disselnmeyer et al. 2024), the Multi-AMR BSRRP adds challenges: unit arrivals, real-time decisions, and fleet coordination. Unlike PMP/UPMP, BSRRP couples reshuffling with fleet navigation. While BRP research often assumes single-crane operations (Ji et al. 2015; Tang and Ren 2010), BSRRP leverages AMR flexibility for coordinated navigation without rigid segmentation.

BSRRP remains distinct from other optimisation problems. It avoids crane-specific non-crossing constraints of the Yard Crane Scheduling Problem (YCSP) (Kizilay and Eliiyi 2021). Unlike space-constrained AGV scheduling (Chen, Tiong, and Chen 2019), it incorporates active reshuffling for deep LIFO stacks. It differs from the Parallel Stack Loading Problem (PSLP) by managing continuous evolution (Boge and Knust 2020) and from the Storage Location Assignment Problem (SLAP) by handling ongoing reorganisation instead of static snapshots (Charris, Rojas-Reyes, and Montoya-Torres 2018). While the Pallet Retrieval and Processing Problem (PRPP) focuses on transport interfaces (Buckow, Goerigk, and Knust 2025) and Robotic Mobile Fulfillment Systems (RMFS) often address stochastic human picking (Teck, Dewil, and Vansteenwegen 2024), BSRRP optimises deterministic reshuffling to minimise travel distance.

Finally, BSRRP differs fundamentally from traditional warehouse order picking (van Gils et al. 2019). While the latter focuses on batching and routing in open aisles, BSRRP manages dense, multi-deep floor storage under LIFO constraints with single-capacity AMRs. The resulting cascading reshuffling logic necessitates fundamentally different modelling approaches than standard open-aisle routing.

2.2. Extensions for modelling the buffer storage, retrieval, and reshuffling problem

This study extends previous research on the BRR problem (Disselnmeyer et al. 2024) by incorporating storage operations into the buffer and multi-AMR management. We build upon the foundational BRP model by Borjian et al. (2015) for storing, retrieving and reshuffling container stacks in a single-crane yard, adapting it to address key requirements for autonomous buffer zones

(1) *Objective Function:* Prioritising the minimisation of total AMR travel distance rather than the number of moves, which is the traditional focus of the BRP literature.
(2) *Unrestricted Relocation:* Allowing the relocation of any blocking unit load, not just those directly blocking the target, which is a common approach for the BRP.
(3) *Retrieval Time Windows:* Incorporating strict time windows for retrieval to ensure process synchronisation.
(4) *Time-Based Modelling:* Utilising discrete time steps for precise multi-AMR coordination and collision avoidance.
(5) *Storage Decisions:* Accommodating new unit loads requiring placement during ongoing operations, rather than focussing solely on reshuffling and retrieval tasks.

2.3. Heuristic and learning-Based approaches

Given the NP-hard nature of BRP variants, exact methods are often intractable for real-time decision-making, prompting the widespread use of heuristics (Kim and Hong 2006; Lersteau and Shen 2022). However, the BSRRP extends beyond the purely combinatorial challenge of reshuffling: it necessitates the translation of moves into collision-free trajectories. Consequently, the problem encompasses both pathfinding, commonly solved using A∗ or Multi-Agent Path Finding (MAPF) techniques (Q. Wang et al. 2024), and task allocation, which relates to the Vehicle Routing Problem (VRP) (Archetti et al. 2025; Dantzig and Ramser 1959).

This complexity highlights the potential of hybrid decomposition approaches. For example, Bömer, Koltermann et al. (2024) successfully applied a sequential method combining A∗ search for reshuffling logic with a mixed-integer program for multi-AMR tour planning in the UPMP context. This demonstrates the viability of leveraging well-established heuristics to solve coupled subproblems sequentially. In the broader context of pre-marshalling, recent advances have also employed Monte Carlo Tree Search (MCTS) to effectively manage the cascading chain effects of reshuffling moves (Z. Wang et al. 2024).

Most recently, concurrent research has explicitly addressed the intersection of multi-agent pathfinding

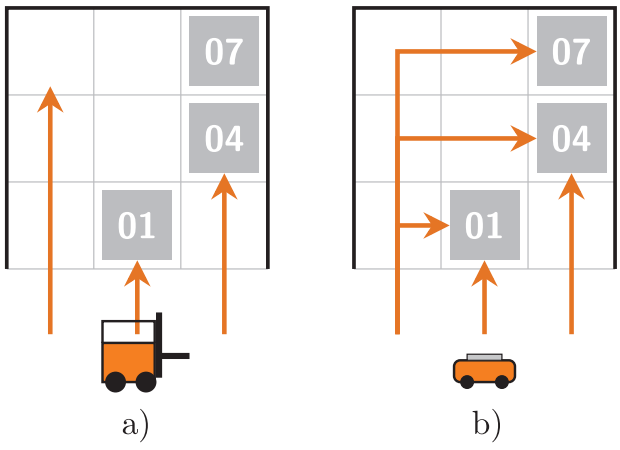

Figure 2. Conceptual comparison of buffer accessibility: (a) Access from the Perimeter, as defined in the BSRRP. (b) In-Grid Access, typical for Robotic Mobile Fulfillment Systems (RMFS) where agents navigate within the grid; In the former, AMRs are restricted to the exterior aisles, necessitating strategic reshuffling of obstructing unit loads to access deep LIFO slots.

and movable obstacles in dense environments: Hu, Zhao, and Ren (2025) introduced M-PAMO (MAPF Among Movable Obstacles), utilising Conflict-Based Search (CBS) to resolve dependencies between agents and movable blockers. Addressing extreme density, Makino and Ito (2025) proposed MAPF-HD (MAPF-for high density environments), utilising swapping heuristics to manage obstructing agents.

Finally, Fu et al. (2026) formalised the Block Rearrangement Problem (BRaP) for dense storage grids. Their approach models the system as a discrete sliding-tile puzzle where agents navigate within the grid to rearrange blocks. Complementing this, Geft et al. (2026) investigate the theoretical limits of relocation-free retrieval under uncertainty. They prove that relocations can be eliminated through robust storage arrangements if specific empty space ratios are maintained. While these works establish important foundations for dense storage, they assume internal grid accessibility or focus on static sequence optimisation. In contrast, the BSRRP addresses scenarios where AMRs are restricted to perimeter access (see Figure 2), due to the characteristics of the layout, unit loads, or AMR fleet, necessitating logic to handle LIFO access constraints from the outside. Furthermore, unlike the focus on minimising moves between static snapshots found in these approaches, our work addresses the continuous temporal synchronisation required for floor-handling AMRs to meet strict time windows.

2.4. Research gap

Within production logistics, the BSRRP serves as a framework for managing localised, high-density buffer zones that decouple sequential manufacturing processes in restricted brownfield environments. Despite the extensive literature on BRP and autonomous logistics, a significant research gap remains regarding the Multi-AMR BSRRP. As summarised in Table 1, existing research tends to isolate specific subproblems, failing to address the interdependent complexities of modern brownfield production logistics. Specifically, the table categorises state-of-the-art approaches across key methodological dimensions: handling operations, access mode (physical constraints), fleet configuration, temporal representation and the Method used for solving the corresponding problem in that paper.

Regarding the latter, we distinguish between three levels of abstraction that define when the system state is evaluated:

- *Moves* treat operations as a logical sequence; the system state is only updated between complete crane or robot movements, effectively ignoring durations.
- *Steps* partition time into fixed, synchronous intervals (ticks); the entire system state is recalculated at every interval (e.g. every 5 s), which is common in grid-based pathfinding but imposes rigid synchronisation.
- *Continuous* representations allow for variable task durations. While implemented on a discrete time grid for computational feasibility, the model treats travel times as distance-dependent parameters rather than fixed steps, enabling precise synchronisation with external deadlines.

Based on this comparison, we identify three specific gaps in the current body of work:

(1) *Lack of Integrated Models:* Most existing approaches address only fragments of the problem. For instance, the BRR model (Disselnmeyer et al. 2024) effectively optimises the reshuffling and retrieval of a fixed set of unit loads but neglects the storage of unit loads arriving from upstream processes. Conversely, literature on the dynamic BRP (e.g. Borjian et al. 2015) accounts for incoming containers but typically minimises the number of moves for a single crane. This metric is insufficient for AMR fleets, where minimising travel distance and execution time is critical to meeting strict retrieval deadlines in a spatially distributed buffer. Consequently, there is no model that integrates the conflicting objectives of storage, retrieval, and reshuffling into a single formulation.

(2) *Insufficient Multi-AMR Coordination for Perimeter Access:* While the coordination of multiple agents is central to MAPF, scalability remains a

Table 1. Comparison of related literature in the field of block relocation and buffer reshuffling.

Reference	Handling operations	Access mode	Fleet	Temporal representation	Method
Caserta, Schwarze, and Voß (2012)	◐	**Perimeter**	1 Crane	Moves	IP
Borjian et al. (2015)	●	**Perimeter**	1 Crane	**Continuous**	IP
Ji et al. (2015)	◐	**Perimeter**	n Cranes	Moves	Genetic Algo.
Disselnmeyer et al. (2024)	◐	**Perimeter**	1 AMR	**Continuous**	IP
Hu, Zhao, and Ren (2025)	○	In-Grid	**n AMRs**	Steps	CBS Search
Bömer, Pfrommer et al. (2026)	○	**Perimeter**	**n AMRs**	Moves	CP / A∗
Fu et al. (2026)	◐	In-Grid	**n AMRs**	Steps	Symbolic Planning
This Paper	●	**Perimeter**	**n AMRs**	**Continuous**	**IP + Heuristic**

Note: **Operations:** ○ Reshuffling only, ◐ Reshuffling & Retrieval, ● Storage, Reshuffling & Retrieval
Temporal Rep.: *Moves* (abstract sequence), *Steps* (discrete synchronous), *Continuous* (variable durations).

hurdle in dense, interactive environments (Hua, Wang, and Ji 2024; Q. Wang et al. 2024). Standard MAPF ignores the manipulation of unit loads and focuses solely on computing collision-free paths for the agents. Conversely, multi-crane BRP literature addresses manipulation but relies on zoning strategies or rail-bound constraints (Ji et al. 2015) that do not apply to flexible AMRs. Furthermore, recent concurrent studies on Block Rearrangement or Movable Obstacles (Fu et al. 2026; Hu, Zhao, and Ren 2025) model the problem as a discrete sliding-tile puzzle where agents navigate within the grid (Grid-Based traffic control). This abstraction differs fundamentally from the BSRRP, where AMRs access the buffer from the perimeter. There is currently no model that couples the combinatorial complexity of BRP reshuffling with the coordinated traffic management of a multi-AMR fleet required to meet strict service time windows.

(3) *Absence of Fleet-Aware Heuristics:* Addressing the BSRRP requires tightly integrating storage assignment, reshuffling logic, retrieval sequencing, and vehicle routing. Solving these problems in isolation is known to yield suboptimal results, as the interdependencies between subproblems are lost (ElWakil, Eltawil, and Gheith 2022). While heuristics exist for different BRP variants (see the survey by Lersteau and Shen 2022), there is a lack of scalable approaches specifically designed to couple the combinatorial storage, retrieval and reshuffling decisions with the spatio-temporal constraints of multi-AMR fleet routing.

As demonstrated in Table 1, this work is the first to simultaneously integrate storage, retrieval, and reshuffling for multi-AMR fleets with perimeter access and continuous time. This provides a unified treatment of these interdependent constraints for a previously unformulated industrial problem.

3. The multi-AMR buffer storage, retrieval, and reshuffling problem

This section formally defines the Multi-AMR Buffer Storage, Retrieval, and Reshuffling problem (BSRRP). We describe the physical storage environment, introduce the concept of Static Lanes as a static graph overlay to manage accessibility, and detail the objective function used to coordinate the fleet.

3.1. Static lanes and graph overlay

The buffer consists of a continuous floor area where unit loads are stacked directly on the ground. To ensure accessibility and prevent deadlocks in this dense environment, we overlay a fixed graph structure onto the continuous space. Following the approach of Pfrommer, Meyer, and Tierney (2022), we partition the storage area into a set of Static Lanes, denoted as $\mathcal{I}$.

Figure 3 illustrates this concept using a 5×5 buffer layout. As shown in Figure 3(a), the floor is first discretised into a grid of storage slots. In a second step (Figure 3(b)), these slots are grouped into Static Lanes based on their accessibility from the perimeter aisles.

This partitioning serves three critical operational purposes:

(1) *LIFO Constraints:* Each static lane $i \in \mathcal{I}$ functions as a Last-In-First-Out (LIFO) stack. As depicted in Figure 3(b), a lane consists of a sequence of floor slots. To retrieve a unit load stored deep within a lane i, all blocking unit loads placed in front of it (i.e. closer to the aisle) must first be reshuffled to empty slots in other lanes $k \in \mathcal{I} \setminus \{i\}$.

(2) *Deadlock Prevention:* A major challenge in multi-AMR systems on dense grids is congestion. To prevent circular deadlocks, we enforce a strict resource locking mechanism: Only one AMR may enter a lane at any given time. If multiple AMRs were to enter a single lane i simultaneously, the leading robot would be blocked by the following one, causing a deadlock. Consequently, a lane $i \in \mathcal{I}$ is locked by an AMR during entry and exit operations.

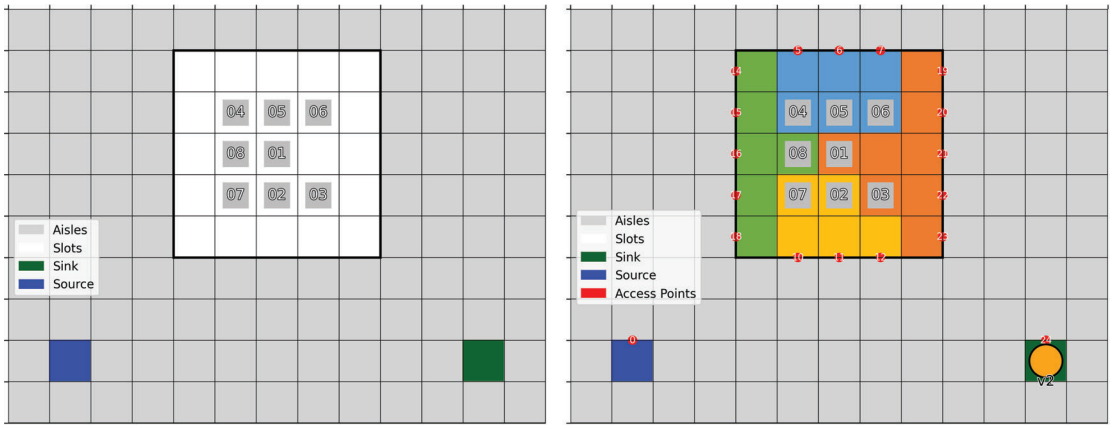

Figure 3. Logical decomposition of the storage area. (a) The continuous floor is discretised into slots. (b) These slots are grouped into Static Lanes (colored), where each lane acts as a LIFO stack accessible from the perimeter. The grey numbered rectangles represent the unit loads. (a) Physical Grid Layout (5×5). (b) Static Lane Overlay.

(3) *Static Lane Configuration:* While the allocation of unit loads to slots is dynamic, the lane layout itself (the set $\mathcal{I}$) is static. For the instances analysed in this study, the lane configuration is pre-computed (e.g. using a maximum-flow network formulation Pfrommer, Meyer, and Tierney 2022) to maximise storage density while ensuring connectivity. Once generated, this layout remains fixed throughout the runtime.

3.2. Model simplifications

To maintain computational tractability while capturing the core dynamics of the system, we apply the following abstractions:

- *Kinematics:* We assume constant AMR velocity and negligible acceleration/deceleration phases. This allows us to model travel time as a linear function of distance on the graph.
- *Handling Times:* Unit load handling (picking up and setting down) is modelled with constant duration, assuming standardised load carriers.
- *Idealized Environment:* We assume a deterministic environment without disruptions (e.g. human traffic or breakdowns), focussing the optimisation on the logical coordination of the fleet.

3.3. Objective function and model flexibility

The primary objective of the BSRRP is to minimise the total distance travelled by the AMR fleet while strictly satisfying all service time windows.

Given the set of unit loads $\mathcal{N}$ designated for retrieval, where each load $n \in \mathcal{N}$ has a release time r_n and a deadline d_n (derived from the retrieval window), the solver must determine a sequence of storage, reshuffling, and retrieval moves such that:

(1) Every retrieval job is completed within its time window $[r_n, d_n]$.
(2) No static lane constraints (LIFO, Capacity, Exclusive Access) are violated.
(3) The sum of distances for all loaded and empty AMR moves is minimised.

The cumulative distance serves as a robust proxy for efficiency. Within the production logistics literature, minimising travel distance is widely established as the primary objective for reducing energy consumption, hardware wear, and overall operational costs in automated guided vehicle systems (Vis 2006). Furthermore, by strictly enforcing time window constraints, the model acts as a validation tool for strategic planning: if the operational requirements exceed the fleet's capacity given the layout, the model returns an infeasibility status, signalling the need for layout adjustments or fleet expansion.

4. Exact problem formulation

We formulate the Multi-AMR Buffer Storage, Retrieval, and Reshuffling Problem (BSRRP) as a binary integer program, referred to as the Exact Formulation (EF). This model builds directly upon the Buffer Reshuffling and Retrieval formulation introduced in Disselnmeyer et al. (2024). We extend this model to a multi-AMR

Table 2. Sets, indices, and parameters.

Notation	Description
$\mathcal{N}, \mathcal{V}, \mathcal{T}$	Sets of unit loads, AMRs, and time steps.
$\mathcal{I}, \mathcal{I}'$	Sets of all lanes (incl. I/O), Set of all buffer lanes (excl. I/O).
$[i, j]$	Slot at lane i and position j ($j = 1$ is deepest in the lane).
d_{ijkl}	Distance between slots $[i, j]$ and $[k, l]$.
d_{01ij}	Distance from the source [0, 1] to a slot $[i, j]$
d_{ij1}	Distance from a slot $[i, j]$ to the sink $[I, 1]$.
τ_{ijkl}	Travel and handling time (cost) between slots $[i, j]$ and $[k, l]$.
h	Handling time for picking up or dropping a unit load.
$[a_n, a_n + \alpha_n]$	Arrival time window for unit load n.
$[r_n, r_n + \rho_n]$	Retrieval time window for unit load n.

setting by incorporating unit load arrivals–conceptually following the dynamic BRP formulation by Borjian et al. (2015)–while adding explicit constraints for collision-free fleet coordination and floor-level maneuvering.

4.1. Notation

Let $\mathcal{N} = \{1, \ldots, N\}$ be the set of unit loads, $\mathcal{I} = \{0, \ldots, I\}$ the set of lanes, and $\mathcal{T} = \{1, \ldots, T\}$ the set of time steps. The fleet of AMRs is denoted by $\mathcal{V} = \{1, \ldots, V\}$. A specific storage location is defined as $[i, j]$ for lane $i \in \mathcal{I}$ and depth position $j \in \mathcal{J}_i = \{1, \ldots, J_i\}$. Within this notation, specific index combinations are explicitly reserved for the input and output locations of the system: the slot $[0, 1]$ consistently denotes the system's source, and the slot $[I, 1]$ represents the sink. The planning horizon T is calculated based on the latest arrival (a_n) and retrieval (r_n) deadlines and the length of the arrival (α_n) and retrieval window (ρ_n):

$$T = \max_{n,m \in \mathcal{N}} \{a_n + \alpha_n, r_m + \rho_m\} \tag{1}$$

The parameter τ_{ijkl} represents the time cost to move between slots. It defaults to the distance d_{ijkl}, but includes handling time h for loaded moves: $\tau_{ijkl} = \max(1, d_{ijkl} + 2h)$. We enforce a lower bound of 1 to ensure that every action consumes time. This prevents the solver from scheduling instantaneous zero-cost idle loops, ensuring that waiting explicitly advances the system state.

Table 2 summarises the sets, indices, and parameters.

4.2. Decision variables

We classify the binary decision variables into AMR actions and system state indicators. All variables are defined as 1 if the condition holds and 0 otherwise.

The decision-making process is modelled using two categories of binary variables. First, AMR decision variables determine the specific tasks started by each vehicle v at time t. We define variables for reshufflings (x), retrievals (y), and the storage of new arrivals (z). Additionally, the variable e explicitly models empty travelling to accurately track the positions of the AMRs.

Second, state variables are required to track the physical configuration of the system. The variable b maps the inventory of unit loads to slots while c tracks the spatio-temporal position of every AMR to prevent collisions. The auxiliary variables s and g monitor the completion status of storage and retrieval requests, respectively.

AMR Decision Variables: Control the movement and handling tasks of each vehicle v.

$$x_{ijklntv} = 1 \text{ if AMR } v \text{ relocates } n \text{ from } [i,j] \text{ to } [k,l] \text{ at } t, \quad \forall\, i,k \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, l \in \mathcal{J}_k,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{2}$$

$$y_{ijntv} = 1 \text{ if AMR } v \text{ retrieves } n \text{ from } [i,j] \text{ at } t, \quad \forall\, i \in \mathcal{I} \setminus \{I\},\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{3}$$

$$z_{ijntv} = 1 \text{ if AMR } v \text{ stores } n \text{ into } [i,j] \text{ at } t, \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{4}$$

$$e_{ijkltv} = 1 \text{ if AMR } v \text{ performs an empty drive from } [i,j] \rightarrow [k,l] \text{ at } t, \quad \forall\, i,k \in \mathcal{I},\ \forall\, j \in \mathcal{J}_i,\ \forall\, l \in \mathcal{J}_k,\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{5}$$

State Variables: Track the location and completion status of loads and vehicles.

$$b_{ijnt} = 1 \text{ if unit load } n \text{ is in slot } [i,j] \text{ at time } t, \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{6}$$

$$c_{ijtv} = 1 \text{ if AMR } v \text{ is present at } [i,j] \text{ at time } t, \quad \forall\, i \in \mathcal{I},\ \forall\, j \in \mathcal{J}_i,\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{7}$$

$$s_{nt} = 1 \text{ if unit load } n \text{ is stored by time } t-1, \quad \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{8}$$

$$g_{nt} = 1 \text{ if unit load } n \text{ is retrieved by time } t-1, \quad \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{9}$$

4.3. Objective function

The objective is to minimise the total travel distance of the AMR fleet. Minimising distance serves as a proxy for energy efficiency and reduces hardware wear. Furthermore, reducing unnecessary travel alleviates congestion, which indirectly aids the throughput of the system. Note that service deadlines are treated as hard constraints to

guarantee service levels; therefore, the objective focuses purely on efficiency rather than tardiness penalties.

The function sums the distances for retrieval (y), storage (z), reshuffling (x), and empty travel (e):

$$\min \sum_{t\in\mathcal{T},v\in\mathcal{V}} \left[\sum_{n\in\mathcal{N}} \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}_i}} (y_{ijntv} d_{ijI1} + z_{ijntv} d_{01ij}) + \sum_{\substack{i,k\in\mathcal{I}\\ j\in\mathcal{J}_i\\ l\in\mathcal{J}_k}} d_{ijkl} (e_{ijkltv} + \sum_{n\in\mathcal{N}} x_{ijklntv}) \right] \tag{10}$$

4.4. Constraints

The feasible region is governed by flow conservation, physical consistency, time windows, and traffic rules.

Flow Conservation & State Updates. We explicitly model the system dynamics using four flow conservation constraints that link the discrete AMR actions to the state of the unit loads and vehicles. Constraints (11) define the storage status s_{nt}. A unit load n is considered stored at time t if it was stored initially (s_{n1}) or if a storage action z transporting it from the source has been completed at a past time step t'. This summation explicitly accounts for the travel time τ_{01ij}, ensuring the status only updates after the transport is finished. Constraints (12) analogously track the retrieval status g_{nt}. This variable is updated to 1 if a retrieval action y has been initiated for unit load n at any previous time step t'. Unlike storage, this updates at the start of the action to prevent the load from being accessed again. Constraints (13) govern the slot occupancy b_{ijnt} for every buffer slot $[i,j]$. The state at time t is determined by the state at $t-1$, plus any unit loads arriving via reshuffling (x) or new storage (z) after their respective travel times, minus any loads leaving the slot due to reshuffling (x) or retrieval (y). Finally, constraints (14) ensure spatio-temporal continuity for the mobile robots. The AMR position variable c_{ijtv} tracks whether the vehicle v is in the slot $[i,j]$ at time t. The constraint updates the position by adding vehicles arriving from storage (z), retrieval (y), reshuffling (x), or empty travel (e) tasks–accounting for the specific travel duration τ of each–and subtracting vehicles that depart to initiate these tasks.

$$s_{nt} = s_{n1} + \sum_{i\in\mathcal{I}'} \sum_{j\in\mathcal{J}_i} \sum_{t'=1}^{t-\tau_{01ij}} \sum_{v\in\mathcal{V}} z_{ijnt'v}$$
$$\forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{11}$$

$$g_{nt} = \sum_{i\in\mathcal{I}\setminus\{I\}} \sum_{j\in\mathcal{J}_i} \sum_{t'=1}^{t-1} \sum_{v\in\mathcal{V}} y_{ijnt'v} \quad \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{12}$$

$$b_{ijnt} = b_{ijn(t-1)} + \sum_{v\in\mathcal{V}} \left[\sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} \left(x_{klijn(t-\tau_{klij})v} - x_{ijkln(t-1)v} \right) - y_{ijn(t-1)v} + z_{ijn(t-\tau_{01ij})v} \right]$$
$$\forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \setminus \{1\} \tag{13}$$

$$c_{ijtv} = c_{ij(t-1)v} + \sum_{n\in\mathcal{N}} z_{ijn(t-\tau_{01ij})v} - \sum_{n\in\mathcal{N}} y_{ijn(t-1)v} + \sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} \sum_{n\in\mathcal{N}} x_{klijn(t-\tau_{klij})v} + \sum_{k\in\mathcal{I}} \sum_{l\in\mathcal{J}_k} e_{klij(t-\tau_{klij})v} - \sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} \sum_{n\in\mathcal{N}} x_{ijkln(t-1)v} - \sum_{k\in\mathcal{I}} \sum_{l\in\mathcal{J}_k} e_{ijkl(t-1)v}$$
$$\forall i \in \mathcal{I}', \forall j \in \mathcal{J}_i, \forall t \in \mathcal{T} \setminus \{1\}, \forall v \in \mathcal{V} \tag{14}$$

Note: Equation (14) defines the motion for storage lanes $\mathcal{I}'$; analogous conservation constraints apply for the Source (c_{01tv}) and Sink (c_{I1tv}) nodes.

Initialisation. To accurately simulate the system's evolution, we must define its starting state at $t = 1$ based on the input instance. Constraints (15) map the initial slot occupancy, setting the state variable b_{ijn1} to 1 if unit load n occupies slot $[i,j]$ at the start of the planning horizon. Constraints (16) distinguish between initially stored unit loads and future arrivals; the storage status s_{n1} is initialised to 1 for loads already present in the buffer, and 0 for those arriving at later time steps. Finally, Constraints (17) establish the initial AMR positions, assigning each AMR v to its designated starting coordinates $[i,j]$ by setting the position variable c_{ij1v} accordingly.

$$b_{ijn1} = \begin{cases} 1, & \text{if unit load } n \text{ starts in slot } [i,j] \\ 0, & \text{otherwise} \end{cases}$$
$$\forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N} \tag{15}$$

$$s_{n1} = \begin{cases} 1, & \text{if unit load } n \text{ is initially stored in buffer} \\ 0, & \text{otherwise} \end{cases}$$
$$\forall\, n \in \mathcal{N} \tag{16}$$

$$c_{ij1v} = \begin{cases} 1, & \text{if AMR } v \text{ starts in slot } [i,j] \\ 0, & \text{otherwise} \end{cases}$$
$$\forall\, i \in \mathcal{I},\ \forall\, j \in \mathcal{J}_i,\ \forall\, v \in \mathcal{V} \tag{17}$$

System Consistency. We spatial and operational integrity through a set of constraints governing capacity, lane structure, and object permanence. Constraints (18) limit the capacity of each storage slot $[i,j]$, ensuring it holds at most one unit load at any given time t. Constraints (19) mandate a dense storage policy to ensure gapless lane utilisation; a unit load may only occupy the outer position $j+1$ if the adjacent inner position j is also occupied. This ensures that the lanes are filled continuously, reflecting the physical constraints of floor block storage. Constraints (20) ensure that an AMR v can only initiate a retrieval (y) or reshuffling (x) for a unit load n from slot $[i,j]$ if that load is present ($b_{ijnt} = 1$). Finally, Constraints (21) link the vehicle's actions to its physical location. The sum of all tasks–reshuffling (x), empty travel (e), and retrieval (y)–initiated by AMR v at slot $[i,j]$ is bounded by the presence variable c_{ijtv}, ensuring a robot must be at a location to operate from it.

$$\sum_{n\in\mathcal{N}} b_{ijnt} \le 1 \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, t \in \mathcal{T} \tag{18}$$

$$\sum_{n\in\mathcal{N}} b_{ijnt} \ge \sum_{n\in\mathcal{N}} b_{i(j+1)nt} \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i \setminus \{J_i\},\ \forall\, t \in \mathcal{T} \tag{19}$$

$$\sum_{v\in\mathcal{V}} \left(y_{ijntv} + \sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} x_{ijklntv} \right) \le b_{ijnt} \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{20}$$

$$\sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} \sum_{n\in\mathcal{N}} x_{ijklntv} + \sum_{k\in\mathcal{I}} \sum_{l\in\mathcal{J}_k} e_{ijkltv} + \sum_{n\in\mathcal{N}} y_{ijntv} \le c_{ijtv} \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{21}$$

Note: Constraints (21) restrict actions in buffer lanes; separate presence constraints govern the Source (for z, y, e) and Sink (for e). For brevity, we omit the explicit formulation and refer to the complete model in Appendix 1.

Time Windows. We model time windows as hard constraints, rendering any solution that misses a deadline infeasible. Constraints (22) mandate that every unit load n is successfully retrieved exactly once within its designated window $[r_n, r_n + \rho_n]$. Since the deadline applies to the arrival at the Sink $[I, 1]$, the valid start time for a retrieval action y_{ijntv} from a specific slot $[i,j]$ is shifted earlier by the travel time τ_{ijI1}. Constraints (23) explicitly forbid retrieval actions outside this valid interval, preventing premature or tardy deliveries.

$$\sum_{i\in\mathcal{I}\setminus\{I\}} \sum_{j\in\mathcal{J}_i} \sum_{t=r_n-\tau_{ijI1}}^{r_n+\rho_n-\tau_{ijI1}} \sum_{v\in\mathcal{V}} y_{ijntv} = 1 \quad \forall\, n \in \mathcal{N} \tag{22}$$

$$\sum_{i\in\mathcal{I}\setminus\{I\}} \sum_{j\in\mathcal{J}_i} \left(\sum_{t=1}^{r_n-\tau_{ijI1}-1} \sum_{v\in\mathcal{V}} y_{ijntv} + \sum_{t=r_n+\rho_n-\tau_{ijI1}+1}^{T} \sum_{v\in\mathcal{V}} y_{ijntv} \right) = 0 \quad \forall\, n \in \mathcal{N} \tag{23}$$

Note: Constraints (22 and 23) specifically govern retrieval windows. For brevity, we omit the explicit storage constraints, as they are mathematically symmetric to the retrieval case: they restrict the storage actions z_{ijntv} at the Source to start in the arrival window $[a_n, a_n + \alpha_n]$. (See Appendix 1)

Constraint (24) governs the entry logic for incoming unit loads, enforcing that each load n is either stored in the buffer or directly cross-docked. The first term handles standard storage, ensuring the action z occurs during the arrival window $[a_n, a_n + \alpha_n]$. The second term enables direct retrieval from the Source, which must satisfy two simultaneous conditions: the load must be available ($t \le a_n + \alpha_n$) and the resulting delivery to the Sink must meet the retrieval deadline ($t \ge r_n - \tau_{01I1}$). The upper bound of the summation enforces the tighter of these two limiting factors.

$$\sum_{i\in\mathcal{I}'} \sum_{j\in\mathcal{J}_i} \sum_{t=a_n}^{a_n+\alpha_n} \sum_{v\in\mathcal{V}} z_{ijntv} + \sum_{t=r_n-\tau_{01I1}}^{\min(r_n+\rho_n-\tau_{01I1},\, a_n+\alpha_n)} \sum_{v\in\mathcal{V}} y_{01ntv} = 1 \quad \forall\, n \in \mathcal{N} \tag{24}$$

Traffic Control. We implement traffic rules by modelling each static lane i as a unary resource that can accommodate at most one vehicle at any given time t. To formally capture lane occupancy during multi-step transitions, we define the set of relevant start times $\Omega(t,\tau) = \{t' \in \mathcal{T} \mid t - \tau < t' \le t\}$. This set identifies all past time steps t' where an action of duration τ initiated at t' would still be active at the current time t. Furthermore, to accurately account for movement within these restricted spaces, we define $\tau_{lane}(j)$ as the travel time required for an AMR to navigate from the lane's perimeter access point to a target slot at depth j.

Constraints (25) enforce the unary capacity by aggregating three distinct modes of occupation. The first term accounts for *static presence*, where an AMR is waiting in the lane (c_{ijtv}). The second term captures *incoming actions* (storage z, incoming reshuffling/empty travel x, e). Since the lane is blocked from the moment an AMR enters, we sum over the full duration window $\Omega(t,\tau)$. The third term handles *outgoing actions* (retrieval y, outgoing reshuffling/empty travel x, e). To avoid double-counting the vehicle's presence at the instant of departure (which is already captured by c_{ijtv}), we sum over the strictly past interval $\Omega^*(t,\tau) = \Omega(t,\tau) \setminus \{t\}$. Collectively, the sum of these binary indicators must not exceed 1, ensuring collision-free operations.

$$\begin{aligned}
&\sum_{v\in\mathcal{V}}\Bigg[\sum_{j\in\mathcal{J}_i} c_{ijtv} + \sum_{j\in\mathcal{J}_i}\Bigg(\sum_{n\in\mathcal{N}}\sum_{t'\in\Omega(t,\tau_{lane}(j)+h)} z_{ijnt'v} \\
&\quad + \sum_{k\in\mathcal{I}'}\sum_{l\in\mathcal{J}_k}\sum_{n\in\mathcal{N}}\sum_{t'\in\Omega(t,\tau_{lane}(j)+h)} x_{klijnt'v} \\
&\quad + \sum_{k\in\mathcal{I}}\sum_{l\in\mathcal{J}_k}\sum_{t'\in\Omega(t,\tau_{lane}(j))} e_{klijt'v}\Bigg) \\
&\quad + \sum_{j\in\mathcal{J}_i}\Bigg(\sum_{n\in\mathcal{N}}\sum_{t'\in\Omega^*(t,\tau_{lane}(j)+h)} y_{ijnt'v} \\
&\quad + \sum_{k\in\mathcal{I}'}\sum_{l\in\mathcal{J}_k}\sum_{n\in\mathcal{N}}\sum_{t'\in\Omega^*(t,\tau_{lane}(j)+h)} x_{ijklnt'v} \\
&\quad + \sum_{k\in\mathcal{I}}\sum_{l\in\mathcal{J}_k}\sum_{t'\in\Omega^*(t,\tau_{lane}(j))} e_{ijklt'v}\Bigg)\Bigg] \\
&\leq 1 \quad \forall\, i\in\mathcal{I}',\ \forall\, t\in\mathcal{T}
\end{aligned} \tag{25}$$

Finally, Constraints (26) enforce the physical accessibility of the block storage. A slot $[i,j]$ is accessible only if the blocking slot $[i,j+1]$ is empty. Consequently, any action (reshuffling x, retrieval y, or empty travel e) targeting or originating from $[i,j]$ is forbidden if $b_{i(j+1)nt} = 1$.

$$\begin{aligned}
&\sum_{n\in\mathcal{N}}(x_{ijklntv} + y_{ijntv} + e_{ijkltv}) \\
&\quad \leq 1 - \sum_{n\in\mathcal{N}} b_{i(j+1)nt}\ \forall\, i\in\mathcal{I}',\ \forall\, j\in\mathcal{J}_i\setminus\{J_i\}, \\
&\quad \forall\, k\in\mathcal{I}',\ \forall\, l\in\mathcal{J}_k,\ \forall\, t\in\mathcal{T},\ \forall\, v\in\mathcal{V}
\end{aligned} \tag{26}$$

4.5. Computational complexity

The BSRRP generalises the Block Relocation Problem (BRP), which is known to be NP-hard (Caserta, Schwarze, and Voß 2012). Consequently, the BSRRP is also NP-hard. This relationship can be established via a polynomial-time reduction where a standard BRP instance is mapped to a restricted BSRRP instance by fixing the fleet size to one ($|\mathcal{V}| = 1$), eliminating arrivals ($a_n = 1$), setting empty travel costs to zero, and normalising all loaded travel distances to unity ($d_{ijkl} = 1$). Under these conditions, the BSRRP objective of minimising total travel distance becomes mathematically equivalent to the BRP objective of minimising the total number of relocations.

For a theoretical foundation, we provide a formal proof of this reduction in Appendix 2. The proof explicitly constructs the mapping from BRP to BSRRP, demonstrating that intractability persists even under the specific constraints of our proposed model. Given this complexity, the EF is computationally intractable for large-scale instances, necessitating the heuristic approach described in Section 5.

5. Heuristic approach

The Exact Formulation proposed in Section 4 provides an exact solution but becomes computationally intractable for large-scale instances due to the coupled complexity of multi-agent pathfinding, task scheduling, and the storing, retrieving and reshuffling of unit loads. To address this, we propose a hierarchical heuristic approach that decomposes the global optimisation problem into four sequential, tractable sub-problems. This approach builds upon the multi-bay sorting strategies proposed by Bömer, Koltermann et al. (2024), extending them to handle the storage and retrieval operations and the multi-AMR constraints of the BSRRP problem
.

As illustrated in the flowchart (Figure 4), the proposed methodology processes the orders through four sequential stages. First, *Priority Queue Generation* linearises the asynchronous orders to provide a strict execution sequence for the *Operation Sequencing via A∗ search*, to find a sequence of transportation tasks with storage, retrieval and reshuffling moves. Finally, *Multi-AMR Scheduling* maps these moves to the AMR fleet via Constraint Programming, while the subsequent *Trajectory Repair* resolves any remaining spatiotemporal conflicts of the AMR positions using a three-tier priority mechanism to ensure collision-free execution.

Table 3 summarises the notation used throughout the heuristic formulation. In the following each of the four components will be explained.

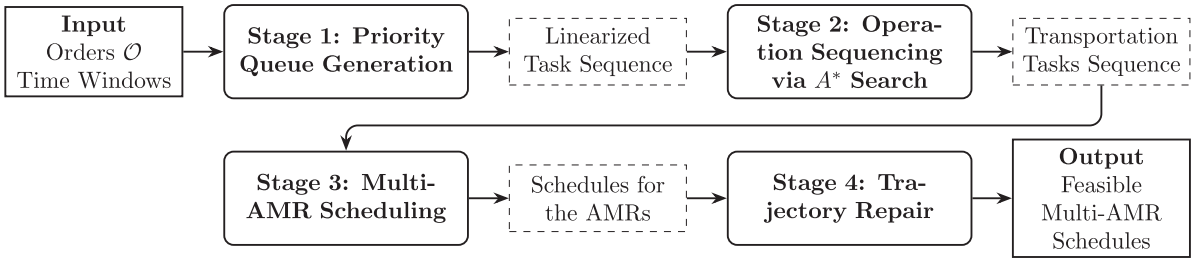

Figure 4. Overview of the hierarchical heuristic. The sequential stages transform the input from a linearised task sequence into strict precedence constraints, then into timed assignments, and finally into collision-free trajectories.

Table 3. Notation and parameters used in the heuristic approach.

Symbol	Description
Sets and Indices	
$\mathcal{O}$	Set of storage and retrieval orders ($\mathcal{O} = \mathcal{O}_S \cup \mathcal{O}_R$)
$\mathcal{O}_S, \mathcal{O}_R$	Subsets of storage and retrieval orders
o, p	Indices for specific orders ($o, p \in \mathcal{O}$)
$\mathcal{G}$	Set of dependency groups formed during priority assignment
$\mathcal{M}$	Set of moves generated by A∗ search
m_a, m_b	Indices for specific moves (storage, retrieval, reshuffling)
$\mathcal{V}$	Set of Autonomous Mobile Robots (AMRs)
u	Index for a specific unit load
Parameters and Variables	
$[a_o, d_o]$	Arrival and retrieval time window for order o
D_g	Effective deadline of a group ($\min_{o\in g} d_o$)
Q	Final prioritised execution queue
$P(u)$	Assigned priority value for unit load u (lower value indicates higher urgency)
$o \sim p$	Enclosure relationship (o is nested within p)
θ	Tabu tenure for cleared lanes
σ	Search state tuple ($B, \mathcal{U}_{\text{src}}, v_{\text{pos}}$) in A∗ search
Σ'	Set of candidate successor states generated during expansion
γ	Congestion scaling factor (set to 5)
W	Penalty weight for time window deviations (set to 10,000)
t_{handling}	Time required for load/unload operations
Functions and Variables	
$g_{\text{time}}(\sigma)$	Elapsed schedule time (makespan) at state σ
$h_{\text{est}}(\sigma)$	Total heuristic estimate cost
h_{ops}	Operational Cost component
h_{block}	Blocking Cost component
h_{prio}	Priority Penalty
$h_{\text{pre\_store}}$	Premature Storage Penalty
$\mathcal{A}_{m,v}$	Interval variable for move m assigned to vehicle v
$\text{start}_{m,v}, \text{end}_{m,v}$	Start and end times of move assignments
$\tau_{i,j}$	Travel time between location i and j

5.1. Priority queue generation

The first stage of the heuristic transforms the set of asynchronous storage and retrieval requests into a linear execution sequence. Let $\mathcal{O}$ denote the complete set of orders, comprising both storage requests $\mathcal{O}_S$ and retrieval requests $\mathcal{O}_R$. Each order $o \in \mathcal{O}$ is constrained by a time window $[a_o, d_o]$, where a_o represents the earliest release time and d_o the hard deadline.

Standard sorting strategies, such as Earliest Due Date (EDD), often fail in scenarios with asynchronous release times, where a task with a late deadline might have a very late release time (a tight window), rendering it more critical than a task with an earlier deadline but a wide window. Executing the task with the wider window first may irreversibly consume temporal resources required by the tighter task immediately upon its release.

Algorithm 1: Enhanced Earliest Due Date Strategy

Input: Set of orders $\mathcal{O}$
Output: Priority Queue Q
$\mathcal{G} \leftarrow$ Partition $\mathcal{O}$ into groups using enclosure relation $o \sim p$ (Equation (27))
Sort $\mathcal{G}$ ascending by group deadline $D_g = \min_{o\in g}(d_o)$
$Q \leftarrow$ Concatenate orders from sorted groups, locally sorted by d_o
return Q

To resolve this, we employ an Enhanced Earliest Due Date strategy, summarised in Algorithm 1, which is based on the concept of enclosed windows. We define an enclosure relationship $o \sim p$ between two orders $o, p \in \mathcal{O}$ if the time window of one is entirely contained within the other:

$$o \sim p \iff (a_o \le a_p \wedge d_o \ge d_p) \vee (a_p \le a_o \wedge d_p \ge d_o) \quad (27)$$

This binary relationship identifies pairs of tasks that compete for the same time interval. We utilise this relationship to partition the set of orders $\mathcal{O}$ into disjoint groups, effectively treating the orders as nodes in a graph where an edge exists between any pair o, p if $o \sim p$. The resulting connected components form the groups, clustering temporally dependent tasks.

The heuristic then generates the final priority queue by sorting these groups to ensure resource availability for critical tasks:

(1) *Inter-Group Sorting:* The groups are sorted by the minimum deadline of their constituent orders

($\min_{o \in \text{group}} d_o$). This ensures that a cluster containing even a single urgent order is prioritised over a cluster of flexible tasks.

(2) *Intra-Group Sorting:* Within each group, orders are sorted by their individual deadlines d_o.

This two-level sorting yields a strict priority ordering that explicitly protects orders with nested, tight windows from being preempted by non-critical tasks. Based on their final position in the queue Q, each unit load u is assigned an integer priority value $P(u)$, where a lower numerical value indicates a higher urgency. For example, a sequence of four unit loads might receive the priority assignments $P(u_1) = 1, P(u_3) = 1$ (if they share the same critical deadline), $P(u_4) = 2$, and $P(u_2) = 3$. During the subsequent A search, these priority values are used to calculate penalties for invalid or non-optimal placement sequences.

5.2. Operation sequencing via A∗ search

The second stage converts the prioritised sequence of orders $\mathcal{O}$ into a dependency graph of moves $\mathcal{M}$, which explicitly defines the locations for all storage, reshuffling, and retrieval operations. This process employs a two-phase approach: first, a pre-processing step prunes trivial direct transfers from the source to the sink to reduce the search space, followed by an A∗ search that enforces the priority scheme established in Section 5.1 and determines storage locations for the orders by using a virtual AMR as a substitute for the robot fleet.

5.2.1. Direct retrieval pruning

Before initialising the search, we perform a *Direct Retrieval Analysis* to identify unit loads that can bypass the buffer entirely. A unit load u is extracted for direct retrieval if its release time a_u and deadline d_u allow for a direct transfer from Source to Sink. Formally, if $a_u + \tau_{\text{source,sink}} \leq d_u$, the item is removed from the search space and assigned a direct move. This reduces the branching factor for the subsequent pathfinding.

5.2.2. State space and transitions

For the remaining orders, we search for an optimal sequence of operations. The state space is defined by the tuple $\sigma = (B, \mathcal{U}_{src}, v_{pos})$, where B represents the current buffer configuration (mapping unit loads to lanes and slots), $\mathcal{U}_{src}$ is the set of pending unit loads at the source, and v_{pos} tracks the position of a single virtual AMR to estimate and minimise the empty travel distances between consecutive transport tasks. Representing the fleet as a single virtual AMR is a necessary methodological simplification for fleet sizes $|\mathcal{V}| > 1$. Tracking the individual positions of multiple AMRs within the A∗ state space would lead to a combinatorial explosion, rendering the search intractable. By optimising the sequence for a single virtual AMR, the heuristic inherently groups spatially proximate tasks and minimises total empty travel. This produces a highly cohesive operation sequence that the subsequent CP-SAT scheduling stage can efficiently distribute and parallelise across the actual AMR fleet. Transitions between states correspond to three move types:

- Store: Places an arriving unit at the source into a valid empty slot in the buffer.
- Retrieve: Retrieves an accessible unit load from the front of a lane to the Sink.
- Reshuffle: Relocates a blocking unit load to a temporary position.

A state σ is identified as a goal state (*IsGoal*) when all storage and retrieval orders from the priority queue Q have been successfully executed and the buffer state is consistent. Once a goal is reached, the algorithm uses *ReconstructPath* to backtrack through the state transitions, yielding the final move sequence $\mathcal{M}$ for the subsequent scheduling phase.

To ensure scalability while maintaining the solution quality of standard A^*, we employ three key algorithmic optimizations. First, to bound the branching factor in high-density scenarios, we implement a Beam Search strategy (Lowerre 1976). At each expansion step, the generated successors are ranked by their f-cost, and only the top k most promising candidates (with $k = 8$ in our experiments) are added to the open set. Second, we enforce Open Set Pruning as a memory protection mechanism. If the priority queue exceeds a safety threshold (set to 5,000 nodes), the worst-performing 50% of the nodes are discarded to prevent memory exhaustion and search stagnation. Third, we utilise a lazy evaluation strategy for the heuristic cost. Upon node generation, only the computationally inexpensive operational costs are calculated. The expensive penalty components (blocking and priority violations) are deferred and computed only when the node is extracted from the priority queue, preventing wasted computation on unexplored nodes. The complete search procedure, integrating the beam search and memory protection strategies, is summarised in Algorithm 2.

To prevent cyclic behaviour (e.g. immediate refilling of a cleared lane), the search maintains a short-term Tabu list. When a unit load is retrieved or reshuffled from a lane l, that lane becomes tabu for incoming storage or reshuffling moves. To balance fleet coordination with unrestricted lane access, we set a fixed short-term tenure of $\theta = 1$ state transition. This tenure effectively prevents

Algorithm 2: A* search

Input: Initial State σ_{init}, Priority Queue Q
Output: Move Sequence $\mathcal{M}$
$OPEN \leftarrow \{(\sigma_{init}, 0)\}$
while $OPEN \neq \emptyset$ **do**
  Select state σ with lowest $f(\sigma)$ from $OPEN$
  if *IsGoal*(σ) **then**
    return ReconstructPath(σ)
  Step 1: Expansion & Tabu Validation
    Generate successors Σ' by applying actions {*Store*, *Retrieve*, *Reshuffle*}
    Filter Σ': Remove actions violating Tabu tenure θ (Cycle Prevention)
  Step 2: Evaluation & Beam Pruning
    Calculate cost $f(\sigma') = g_{\text{time}} + h_{\text{est}}$ for all $\sigma' \in \Sigma'$
    Sort Σ' by f-score and add only top k nodes to *OPEN* (Beam Width)
  Step 3: Memory Protection
    if $|OPEN| > Limit$ **then** Discard worst 50% of nodes from *OPEN*
return Failure

immediate inverse operations (such as placing a unit load back into the position it just vacated) without restricting the solution space or locking down buffer capacity for extended periods, which proved more robust than dynamic, fleet-dependent tenures in testing.

5.2.3. Cost function

The search is guided by a composite cost function $f(\sigma) = g_{\text{time}}(\sigma) + h_{\text{est}}(\sigma)$.

Total Schedule Makespan. The term $g_{\text{time}}(\sigma)$ represents the total elapsed time of the operation sequence up to state σ. Unlike standard pathfinding which might only sum loaded distances, our cost function accounts for the empty travel time required to travel between task locations, as well as any waiting time incurred if a vehicle arrives before a unit load's release window a_o opens. This is achieved by employing a virtual AMR that sequentially performs the tasks, calculating the unloaded travel from the end position of the preceding move to the start location of the next. This allows penalising inefficient sequencing (e.g. generating excessive empty travel for the virtual AMR between distant tasks) and optimise for the true operational makespan.

Heuristic Cost Function. The heuristic $h_{\text{est}}(\sigma)$ is a weighted sum of four components designed to minimise future effort while enforcing the priority ordering. To maintain search tractability and ensure robust performance in high-density scenarios, the heuristic employs soft constraints within this cost function. Rather than strictly forbidding undesirable states–which could lead to search stagnation–it penalises violations (such as lane blockages or sequence inversions) through weighted penalty terms.

$$h_{\text{est}}(\sigma) = h_{\text{ops}}(\sigma) + h_{\text{block}}(\sigma) + h_{\text{prio}}(\sigma) + h_{\text{pre\_store}}(\sigma) \tag{28}$$

(1) *Operational Costs (h_{ops}):* Estimates the travel time τ required to complete all pending tasks.
- *Storage:* We employ a *Greedy Best-Match* heuristic. The cost for storing pending unit loads is estimated by assigning each item to the nearest available slot that minimises the total sequence: Source → Slot → Sink.
- *Retrieval:* Sums the travel time $\tau_{i,\text{sink}}$ from each stored item's current position to the Sink.

(2) *Blocking Costs (h_{block}):* Anticipates the immediate reshuffling effort required for currently blocked targets. For every blocked target u, the heuristic identifies the set of blockers and calculates the cost to relocate them to the nearest empty lane. If no empty lane is available, the algorithm applies two times the average reshuffle cost to reflect the operational delay of waiting for future retrievals to free up buffer capacity. This cost is scaled by a fixed penalty multiplier $\gamma = 5.0$. We found this fixed value to be effective for consistent congestion avoidance, as it ensures high-priority reshuffling is penalised uniformly regardless of the fleet size.

(3) *Priority Penalty (h_{prio}):* Enforces the task sequence derived in Stage 1 by penalising two types of structural violations based on the assigned priority values $P(u)$:
- *Sequence Inversion:* Penalises the retrieval of a lower-priority unit load while a higher-priority one is still pending in the buffer.
  Example: Retrieving an item with $P = 3$ while an item with $P = 1$ is not retrieved yet.
- *Stacking Violation:* Penalises any LIFO lane configuration where a lower-priority item is placed closer to the aisle than a higher-priority item.
  Example: Placing an item with $P = 2$ in front of an item with $P = 1$ in the same lane. Since this placement guarantees a future reshuffle operation, it is penalised immediately to prune the search branch.

(4) *Premature Storage Penalty ($h_{\text{pre\_store}}$):* Penalises the storage of a low-priority unit load from the source

while a high-priority unit load is waiting to be retrieved from the buffer. This guides the search to clear urgent retrievals first, freeing up buffer space before bringing in less urgent items.

The output of this stage is a sequence of moves $\mathcal{M}$. If move m_a reshuffles a blocker for a retrieval move m_b, a strict precedence constraint $m_a \prec m_b$ is generated for the subsequent scheduling phase.

5.3. Multi-AMR scheduling via CP-SAT

The third stage maps the sequence of moves $\mathcal{M}$ generated by the A∗ search onto the fleet of Autonomous Mobile Robots (AMRs) $\mathcal{V}$. While standard Vehicle Routing Problems (VRP) focus primarily on spatial routing, the BSRRP is dominated by complex temporal constraints and precedences between moves. Consequently, we model this stage as a *Job Shop Scheduling Problem with Sequence-Dependent Setup Times*, solved using the CP-SAT solver from Google OR-Tools.

5.3.1. Model formulation

To emphasise the scheduling nature of the problem, the vehicles of the AMR fleet are modelled as a set of identical parallel machines $\mathcal{V} = \{1, \dots, V\}$. Each generated move $m \in \mathcal{M}$ is treated as a task that must be assigned to exactly one machine. Following standard Constraint Programming formulation, we define optional interval variables $\mathcal{A}_{m,v}$ to represent the potential execution of move m by vehicle v. If active, an interval $\mathcal{A}_{m,v}$ is characterised by a start time $\text{start}_{m,v}$, an end time $\text{end}_{m,v}$, and a fixed processing duration τ_m. This duration combines the loaded travel time (as defined in Section 4) and the constant handling time:

$$\tau_m = \tau_{\text{origin}(m),\text{dest}(m)} + t_{\text{handling}} \tag{29}$$

The model enforces the following constraints:

(1) *Assignment Completeness:* Every move must be assigned to exactly one vehicle. This is enforced by constraining the sum of active assignment booleans to 1 for each move:

$$\sum_{v \in \mathcal{V}} \mathbb{I}(\mathcal{A}_{m,v}) = 1 \quad \forall\, m \in \mathcal{M} \tag{30}$$

(2) *Precedence Constraints:* We enforce three types of hard dependencies to guarantee consistency and stack integrity:
- *Unit Load Flow:* Sequential operations acting on the same unit load (e.g. reshuffle blocker → retrieve target) must maintain the order defined by the logical flow.
- *LIFO Inter-Slot Dependencies:* Derived from the buffer geometry; if unit load A is stored in a slot strictly in front of unit load B within the same lane, the retrieval of A must complete before the retrieval of B can commence.
- *Lane Sequencing:* To preserve the validity of the buffer states determined by the heuristic decomposition, we enforce strict sequencing for all moves accessing the same lane. If the A∗ search schedules move m_a before m_b on lane l, the scheduler is constrained to respect this order ($m_a \prec m_b$), preventing the optimiser from creating invalid lane configurations.

For any such dependency pair (m_a, m_b), we enforce $\text{end}_{m_a} \leq \text{start}_{m_b}$.

(3) *Lane Capacity*: We model storage locations as resource constraints.
- *Buffer Lanes (Unary Resource):* Standard buffer lanes are modelled as unary resources. We apply a global *NoOverlap* constraint on the set of intervals assigned to any specific lane l:

$$\text{NoOverlap}(\{\mathcal{A}_{m,v} \mid \text{dest}(m) = l \vee \text{origin}(m) = l\}) \tag{31}$$

This constraint ensures that at any point in time t, at most one vehicle can execute a move involving lane l.
- *Source and Sink Queues:* Modelled as infinite-capacity resources to track vehicle availability without restricting the number of concurrent AMRs at these locations.

(4) *Sequence-Dependent Transition Times:* The setup time between two moves depends on the robot's location. If vehicle v performs move m_a immediately before m_b, a transition constraint ensures the gap covers the empty travel time:

$$\text{start}_{m_b,v} \geq \text{end}_{m_a,v} + \tau_{\text{dest}(m_a),\text{origin}(m_b)} \tag{32}$$

(5) *Time Windows:* Time windows are modelled using a hybrid approach to ensure feasibility.
- *Hard Constraints:* Physical constraints are strictly enforced (e.g. a storage action cannot start before the unit load's arrival time a_o; a retrieval cannot end after the deadline d_o).
- *Soft Constraints:* Operational targets (latest start, earliest finish) are treated as soft constraints. Violations are permitted to maintain feasibility but incur a weighted tardiness penalty in the objective function to prioritise service-level agreements.

Algorithm 3: Trajectory Repair Strategy

Input: Initial Schedule $\mathcal{S}$
Output: Feasible Schedule $\mathcal{S}^*$
Step 1: Deadlock Resolution
if *Symmetric deadlock detected between* v_1, v_2 **then** Swap schedules of v_1, v_2
Step 2: Conflict Resolution Loop
while *Collision detected between* v_1 *(parking AMR) and* v_2 *(incoming AMR)* **do**
  if v_1 *is waiting empty* ***and*** *early shift feasible* **then** `// Priority 1: Reschedule`
    Shift v_1 departure to $t < t_{arrival}(v_2)$
  else if *valid eviction strategy for* v_1 *exists* **then** `// Priority 2: Evict`
    Insert best eviction move (Smart or Standard) for v_1
  else `// Priority 3: Delay`
    Delay v_2 until v_1 departs (propagate downstream)
Step 3: Dependency Check
foreach *unit load* u *with* $end_{store} > start_{retrieve}$ **do**
  Shift retrieval forward to resolve violation
return $\mathcal{S}^*$

5.3.2. Objective function

The optimisation objective is hierarchical, implemented using a weighted sum method. The primary goal is to minimise total tardiness, reflecting the strict service level requirements. The secondary goal is to minimise the sum of completion times (Flow Time), which implicitly minimises unproductive empty travel and waiting times.

The objective is formulated as:

$$\min \sum_{m \in \mathcal{M}} \left(\underbrace{W \cdot \max(0, \text{end}_m - d_m)}_{\text{Weighted Tardiness}} + \underbrace{\text{end}_m}_{\text{Flow Time}} \right) \quad (33)$$

Here, the term $\max(0, end_m - d_m)$ calculates the strictly positive delay relative to the soft deadline d_m, which corresponds to the order deadline d_o of the unit load manipulated in move m. W is a large weighting constant (set to 10,000) ensuring that meeting deadlines strictly dominates operational efficiency.

5.4. Trajectory repair

While the CP-SAT schedule ensures temporal validity, it ignores the presence of idle AMRs, which may park after performing a storage or reshuffling move and occupy the buffer lanes. To generate collision-free trajectories, the final stage post-processes the timeline following the structure of Algorithm 3:

5.4.1. Step 1: deadlock resolution

The algorithm resolves symmetric head-on collisions (e.g. v_1 moving $A \to B$ while v_2 moves $B \to A$) by swapping the AMRs' future task schedules. Since the AMRs are homogenous, this resolves the deadlock instantly without additional travel time.

5.4.2. Step 2: conflict resolution loop

Remaining spatial overlaps caused by parked AMRs are resolved using a hierarchical strategy prioritising efficiency:

(1) *Priority 1 Reschedule:* Exploits schedule slack to shift the parked AMR's next departure to an earlier time, clearing the lane before the incoming AMR arrives.
(2) *Priority 2 Evict:* Inserts an explicit eviction move. The algorithm prioritises a *Smart Eviction* (moving the AMR directly to the start location of its next assigned task) over a *Standard Eviction* (relocating it to an available neutral position, such as a free buffer lane or the sink).
(3) *Priority 3 Delay:* As a fallback, the incoming AMR is delayed, and this shift is propagated downstream to all dependent tasks.

5.4.3. Step 3: dependency check

Delays introduced in Step 2 may violate precedence constraints (e.g. pushing a storage task to complete after its retrieval was scheduled to begin). This step enforces the $\text{end}_{\text{store}} \leq \text{start}_{\text{retrieve}}$ constraint for all unit loads, shifting retrieval tasks forward if necessary to guarantee consistency.

6. Computational experiments

This section evaluates the performance of the proposed solution approaches for the Multi-AMR Buffer Storage, Retrieval, and Reshuffling Problem (BSRRP). The experimental study is designed to answer three primary research questions:

(1) *Benchmarking:* What are the computational limits of the EF when establishing a ground truth of optimal solutions?
(2) *Heuristic Validation:* How does the proposed hierarchical heuristic perform in terms of feasibility rates and solution quality compared to the optimal baselines?
(3) *Managerial Insights:* How do operational parameters–specifically fleet size, access flexibility, and congestion levels–impact the stability and efficiency of space-constrained buffers?

6.1. Experimental design and dataset

To ensure a robust evaluation, we developed a comprehensive dataset reflecting the constraints of brownfield manufacturing facilities, such as limited floor space, high storage density, and complex traffic dynamics. To guarantee full reproducibility and facilitate future research, the complete source code of the proposed heuristic and the discrete-event simulator, as well as all generated benchmark instances and detailed solutions, are made publicly available (see the Data and Code Availability Statement in the Acknowledgements). Our experimental design employs a two-tiered validation to address different evaluation objectives: solution quality benchmarking on small-scale instances and scalability analysis on large-scale instances.

6.1.1. Instance generation and parameters

Small-Scale Instances. For 3×3 and 4×4 grids, instances were generated stochastically to quantify the heuristic's optimality gap against the EF. We systematically varied the following parameters:

- *Grid and Topology:* 3×3 (9 slots) and 4×4 (16 slots) with access points distributed across 1, 2, or 4 sides.
- *Fleet Size ($|\mathcal{V}|$):* 1 to 3 AMRs.
- *Load-to-Slot Ratio:* Ranged from 0.4 to 1.3 to test capacity limits.
- *Temporal Constraints:* Arrival and retrieval windows were generated with varying overlap (using random seeds for reproducibility) to enforce asynchronous operations. Negative arrival time windows were utilised to initialise instances with a set of already randomly placed pre-stored unit loads at $t = 0$.

Large-Scale Instances. Instance scale is defined by combinatorial complexity, determined by slots, loads, and planning horizon, relative to EF tractability. We classify layouts under 20 slots ($3 \times 3, 4 \times 4$) as small-scale, as they allow the EF to provide baselines for optimality gaps. Larger instances are large-scale, where state-space explosion prevents EF convergence within 3600 s. These are evaluated using a discrete-event simulator that models reshuffling sequences to identify saturation points and maximum load-to-slot ratios for the heuristic.

- *Topology:* Square blocks (5×5, 6×6), rectangular layouts (8×3), and industrial brownfield layouts.
- *Task Generation:* The simulator adaptively alternates storage and retrieval requests to maintain a target fill level (e.g. 80%), utilising a back-to-front filling strategy.
- *Idealized Time Estimation:* Task durations are estimated using scaled Manhattan distances ($t_{op} \approx 2.0 \times d_{\text{manhattan}}$) plus fixed handling times, and reshuffling operations incur explicitly simulated time penalties.
- *Time Window Derivation:* Task start times are greedily assigned to the earliest available robot in a simulated fleet with two AMRs. The arrival ($[a_n, a_n + \alpha_n]$) and retrieval time windows ($[r_n, r_n + \rho_n]$) are then generated by applying a fixed temporal slack (e.g. ± 15 time steps) around these simulated task start and end times.

6.1.2. Computational environment

All experiments were conducted on a workstation equipped with an AMD Ryzen 9 5950X processor. The EF was solved using Gurobi Optimiser version 13.0.0 with a strict time limit of 3600 s (1 h) per instance.

The hierarchical heuristic was implemented in Python, utilising Google OR-Tools (CP-SAT) for the scheduling stage. To ensure rapid convergence and stability during the experiments, the solver was configured with aggressive search parameters (linearisation level 2, probing level 2) and utilised 8 parallel search workers. To simulate a realistic production control environment, we imposed a realistic time limit of 300 s for both the A^* Search (Stage 2) and the CP-SAT solver (Stage 3). While typical runtimes are fractions of a second, this cap prevents indefinite stalling in degenerate cases. Additionally, the algorithmic parameters of the heuristic (specifically the congestion factor $\gamma = 5$, penalty weight $W = 10,000$ and tabu tenure $\theta = 1$) were calibrated based on preliminary experiments using a representative subset

Table 4. Composition of the Benchmark Reference Set: 810 high-quality instances filtered from a pool of 6,903 generated small-scale scenarios.

Parameter	Category	Small (3 × 3)	Large (4 × 4)
Fleet Size ($\|\mathcal{V}\|$)	1 AMR	151	0
	2 AMRs	288	48
	3 AMRs	277	46
Access Directions	1 Side	72	1
	2 Sides	373	54
	4 Sides	271	39
Total	**All Instances**	**716**	**94**

of small-scale instances to balance solution quality and computational speed.

6.1.3. Exact formulation experiments and benchmark set

To evaluate the EF and establish a ground truth, we generated a pool of 6,903 small-scale instances (3 × 3 and 4 × 4) across varying layout complexities and parameters. We subjected these instances to a two-stage process to test the computational limits of the EF and to build a benchmark set.

First, the EF was tasked with finding feasible solutions within a 3600-s time limit. This step identified 949 instances as solvable. We assume this step effectively separates operationally feasible scenarios from those rendered structurally impossible by tight temporal and spatial constraints. This assumption is based on the observation that when the EF found a feasible solution, it typically did so relatively quickly, which we verified by examining individual instances.

Second, to ensure a strict baseline for optimality gap calculations, we filtered this pool to instances where the EF achieved a solution with a MIP gap of $\leq 5\%$. This process yielded a final benchmark set of 810 instances. The composition of this benchmark, categorised by fleet size and access topology, is detailed in Table 4. The distribution highlights the computational boundaries of exact methods: the EF solved 716 instances in the 3 × 3 layout to the required gap, but only 94 instances in the 4 × 4 layout. Notably, the EF failed to solve any 4 × 4 instances with a single AMR, suggesting that the used parameter combination exceeds the capacity of a single robot.

We utilise this filtered benchmark set of 810 optimally or near-optimally solved instances ($\leq 5\%$ MIP gap) during the subsequent evaluation to validate the solution quality and feasibility rates of the proposed heuristic.

6.2. Quantitative benchmarking against exact formulation

We first analyse the quantitative performance of the heuristic against the benchmark set, followed by a qualitative assessment of its conflict resolution capabilities.

6.2.1. Quantitative analysis

Table 5 summarises the comparative performance of the exact formulation (EF) and the proposed heuristic. The results are first categorised by the physical layout and the fleet size ($|\mathcal{V}|$). Under the section *Solve Rate Heuristic*, the first column gives the total number of benchmark instances, the second column lists how many were successfully solved by the heuristic, and the third column provides the resulting success rate. In the *Computational Efficiency (Median)* section, the table compares the median EF runtime against the median heuristic runtime, followed by the relative speedup factor. Finally, the *Median Opt. Gap (%)* column quantifies the solution quality by measuring the objective value deviation of the heuristic solution from the exact lower bound.

Solvability and Structural Limits. The proposed heuristic demonstrated high consistency in finding feasible solutions when evaluated against the benchmark. As detailed under the *Solve Rate Heuristic* section of Table 5, the heuristic achieved success rates comparable to the EF across all parameter combinations where exact solutions were obtainable.

Specifically, the heuristic solved 698 out of 716 instances (97.5%) in the 3 × 3 layout. In the more complex 4 × 4 layout, it successfully solved 89 out of the 94 reference instances (94.7%). Notably, the EF was unable to solve any 4 × 4 instances restricted to a single AMR. As outlined in Section 6.1, this is a direct consequence of the instance generation parameters: the extensive reshuffling required to resolve deep blockages in a 4 × 4 grid simply consumes more time than the tight retrieval windows allow for a single AMR, rendering these scenarios operationally infeasible.

Runtime Performance. While feasibility is the primary prerequisite for deployment, operational viability depends on computational speed. The runtime disparity, visualised in the top panels of Figure 5, highlights the impact of the problem's NP-hard complexity. While the EF frequently approaches the timeout of 3,600 s (e.g. median of 2,927 s for 4 × 4 with 3 AMRs), the heuristic maintains stable sub-second to low-second runtimes. For the 3 × 3 instances, the median computation time was consistently under 0.1 s, achieving speedup factors exceeding 3,000x. Even in the complex 4 × 4 instances, the median was approximately 3 s–well below the 300-s time limit.

Table 5. Performance assessment: Feasibility, Runtime, and Optimality Gap comparison between the EF and the Heuristic approach across the reference set.

		Solve rate heuristic			Computational efficiency (Median)			Median Opt.
Layout	Fleet ($\|\mathcal{V}\|$)	Number of instances	Solved	Rate	EF Runtime	Heur. Runtime	Speedup	Gap (%)
3 × 3	1 AMR	151	139	92.1%	515.5 *s*	0.05 s	10,310x	2.56%
	2 AMRs	288	284	98.6%	253.0 *s*	0.08 s	3,163x	10.20%
	3 AMRs	277	275	99.3%	307.2 *s*	0.09 s	3,413x	14.58%
4 × 4	2 AMRs	48	44	91.7%	1,524.3 *s*	3.37 s	452x	4.70%
	3 AMRs	46	45	97.8%	2,927.9 *s*	2.52 s	1,162x	10.63%

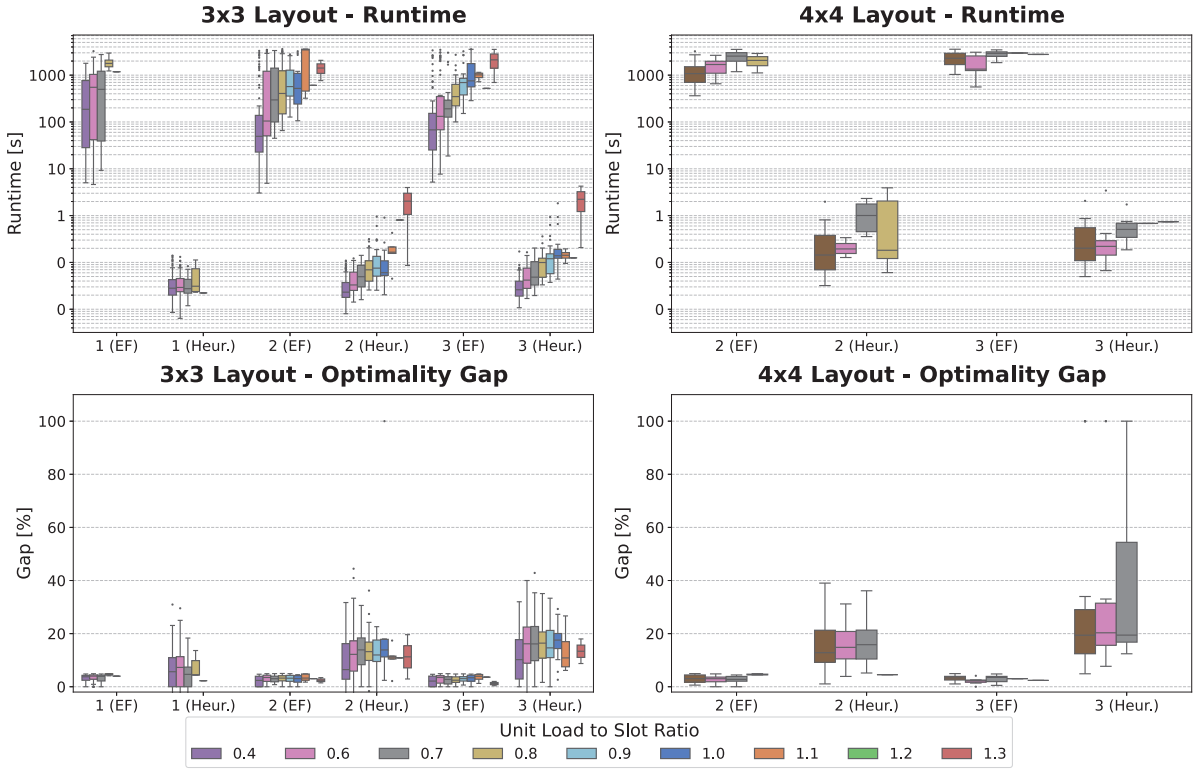

Figure 5. Performance comparison between the EF and the proposed Heuristic. Top panels: Runtime distributions (logarithmic scale) across 3 × 3 and 4 × 4 layouts. Bottom panels: Optimality gaps relative to the exact lower bound, categorised by unit-load-to-slot-ratio.

Solution Quality and Optimality Gap. To assess the trade-off for this massive speedup, we evaluate the optimality gap (visualised in the bottom panels of Figure 5). To provide a rigorous assessment of solution quality, we calculate the True Optimality Gap. Unlike relative deviations between two feasible solutions, this metric compares the heuristic solution (Z_{Heur}) against the theoretical lower bound (Z_{LB}) established by the exact solver:

$$Gap = \frac{Z_{Heur} - Z_{LB}}{Z_{Heur}} \times 100\% \tag{34}$$

The heuristic maintains high solution quality, with a median gap of just 4.70% for the challenging 4 × 4 layout with 2 AMRs. As expected, the gap increases with fleet size and exhibits a higher sensitivity to the unit-load-to-slot-ratio. Medians typically range between 10% and 20% for larger configurations, reflecting the heuristic's tendency to prioritise rapid conflict resolution over finding the mathematically perfect spatial sequence.

Constraint Handling and Temporal Flexibility. A key distinction in this assessment lies in the treatment of time windows. The EF strictly enforces the start times at the storage slots – which are derived from the external deadlines in (22) – as hard constraints, classifying any temporal deviation as an infeasible solution. In contrast, the heuristic is designed to maintain operational flow by selectively relaxing these internal bounds. Specifically, it permits internal storage tasks to complete later and retrieval tasks to commence earlier than originally scheduled. This approach ensures that the handovers at the Source and Sink remain perfectly synchronised with external processes, while providing the internal

buffer operations with the temporal flexibility necessary to resolve spatial conflicts. Among the generated solutions, approximately 9.5% utilised this flexibility. While strictly speaking suboptimal compared to the rigid EF lower bound, these solutions are operationally superior in a production context, as they preserve the punctuality of external processes while preventing internal system lockups.

6.3. Qualitative analysis of solution behaviour

To validate the performance of the proposed approach, we analyse the operational behaviour of the generated solutions in two distinct scenarios: reactive conflict resolution in high-density confined spaces and proactive capacity management in large-scale brownfield layouts. This analysis examines solutions for 8 × 3, 5 × 5, and 6 × 6 layouts, as well as a real-world brownfield case from the large-scale instance set.

6.3.1. Reactive conflict resolution (8 × 3 layout)

First, we examine a highly constrained environment serviced by a fleet of 4 AMRs (Utilisation ≈ 90%). Figure 6 visualises a complex interleaved reshuffling and storage sequence spanning from time step $t = 345$ to $t = 446$. The system's objective is to retrieve the target unit load (UL) 04, which is currently blocked.

(1) *Reshuffling ($t = 345–349$):* The heuristic identifies UL 18 as blocking the retrieval of the target UL 04. AMR V2 is assigned to reshuffle UL 18 to a temporary position.
(2) *Storage ($t = 375–401$):* Concurrently, new storage requests for UL 20 and UL 21 arrive. Rather than blocking the now-accessible lane for UL 04, the heuristic utilises the space in front of the reshuffled UL 18. It directs the fleet to stack UL 20 ($t = 375$) and UL 21 ($t = 401$) in front of UL 18. This choice avoids creating new blockages, as UL 20 and UL 21 are scheduled for retrieval earlier than UL 18.
(3) *Retrieval ($t = 440–446$):* With the blockage removed and incoming traffic diverted to non-critical lanes, AMR V1 navigates to the target lane and retrieves UL 04 within its designated retrieval window.

Overall, this scenario exemplifies how the heuristic resolves conflicts by leveraging temporal slack. It stacks UL 20/21 in front of UL 18 without creating blockages; this is permissible due to the specific retrieval order and maximizes the utility of the constrained floor space.

6.3.2. Spatial strategies in brownfield layouts

To demonstrate the algorithm's capability to operate within complex layouts, we applied the heuristic to a real-world layout from the surface coating industry (Figure 7).

This scenario uses the irregular floor area surrounding a fixed high-gloss coating machine (grey obstacle). The layout is characterised by the adjacent positioning of the Source and Sink (bottom right), requiring all AMR movements to be sequenced through the central aisle.

Figure 7(b) illustrates the system state at $t = 012$. The heuristic utilises this layout through two key mechanisms:

(1) *Static Lanes:* The irregular floor plan is manually decomposed into a set of static Lanes. Specifically, the Orange and Yellow zones are mapped to single-deep lanes (depth 1), while the Green zone forms double-deep lanes (depth 2). Static lanes allow the algorithm to apply standard stack-based logic to handle LIFO constraints, regardless of the specific physical arrangement.
(2) *Look-Ahead Placement:* The heuristic optimises storage locations based on retrieval deadlines. For instance, although both UL 19 and UL 20 are retrieved late in the horizon, the heuristic distinguishes between them. UL 20, having the later deadline, is moved to a distant location (lane 8) during initial reshuffling, whereas UL 19 is kept in a closer slot (lane 16). This prioritisation ensures that accessible buffer capacity is preserved for unit loads with earlier retrieval times.

These results show exemplarly that the proposed logic enables robust operations in layout-constrained environments, preventing deadlocks through the temporal sequencing of lane access.

6.3.3. Spatial decision policy

To analyse the spatial allocation strategies of the heuristic, we evaluate the occupancy intensity and traffic density within the brownfield scenario. As illustrated in Figure 8, occupancy intensity is measured by the average number of timesteps a storage slot holds a unit load, whereas traffic density quantifies the cumulative frequency of AMR visits at each lane access point. This evaluation reveals that the algorithm adopts a highly strategic utilisation of the irregular space without explicit pre-programming. Rather than treating the available floor as a rigid grid, the system behaves organically, dynamically adapting its spatial footprint to the current operational load.

The heatmaps reveal three key behaviours. Notably, these advanced spatial strategies emerge despite the

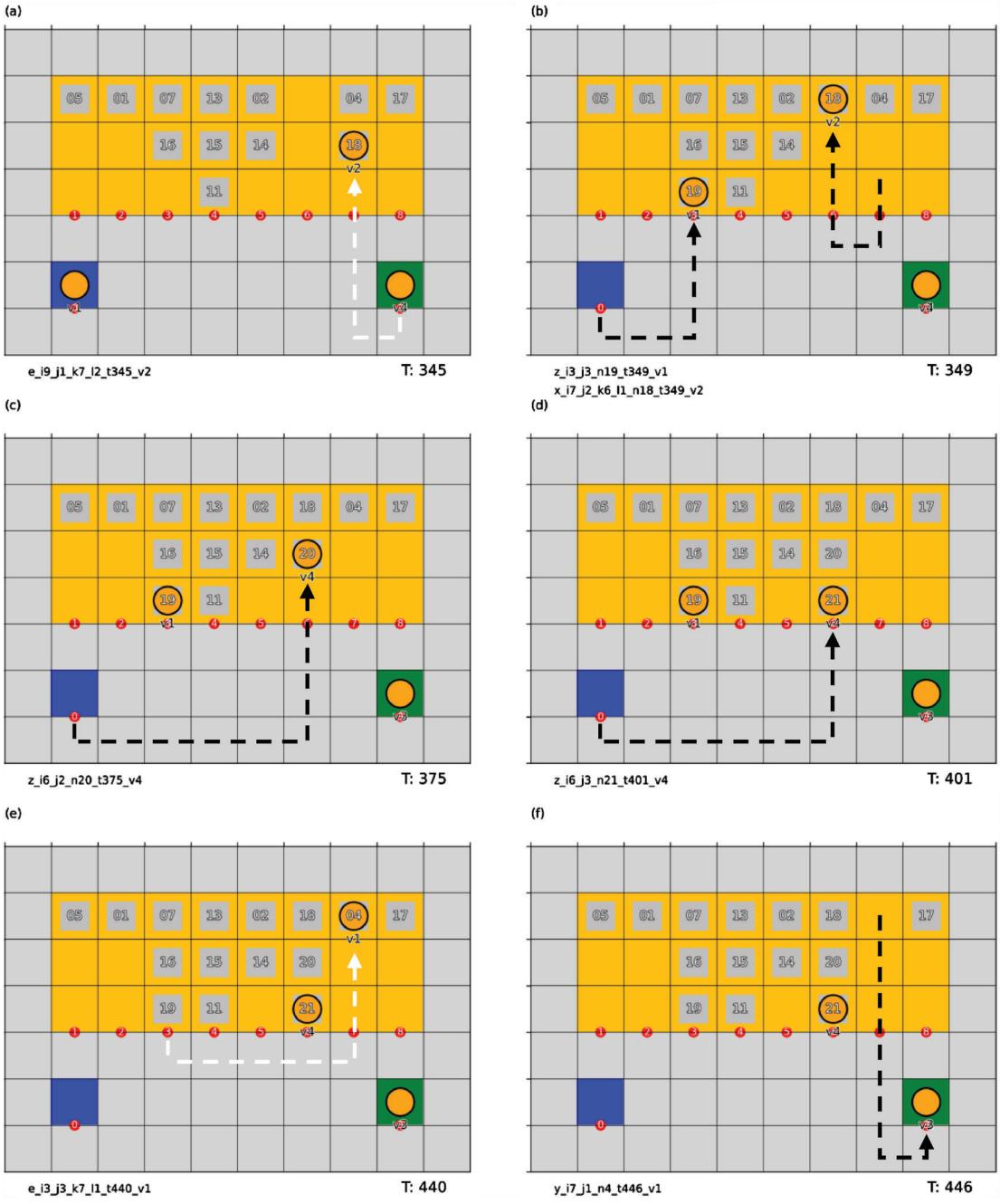

Figure 6. Visualization of a solution in a high-density 8 × 3 layout (21 unit loads) with 4 AMRs. The bottom-left of each sub-figure indicates the corresponding decision variable from the EF (also visualised with the dotted arrows; white for empty drives; black for transportation of an UL), while T denotes the discrete time step of the event.

methodological simplifications of the approach. Specifically, the heuristic nature of the A^* search and the hierarchical decomposition of the problem. This demonstrates that the designed cost function captures global system dynamics, leading to the following emergent behaviours:

(1) *Lane-Blocking Avoidance:* A feature in Figure 8(a) is the low utilisation of the front positions in double-deep lanes, despite their proximity to the access point. The heuristic demonstrates a preference for utilising distant single-deep slots rather than blocking an occupied rear slot in a closer lane. This confirms that the algorithm prioritises accessibility over physical proximity, accepting longer travel distances to prevent future reshuffling operations.

(2) *Buffer Duration-Based Inventory Stratification:* The system exhibits a distinct separation of inventory based on buffer duration in the buffer. Deep slots in

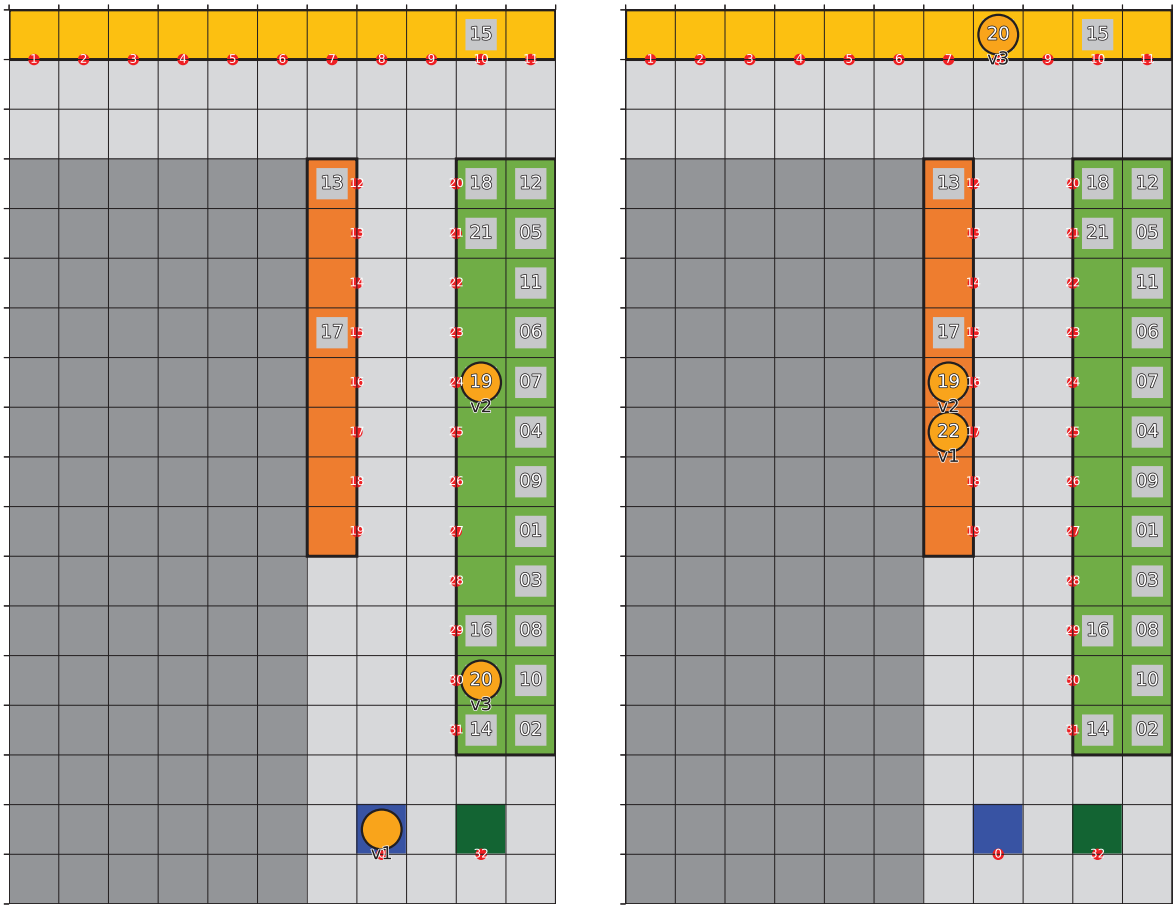

Figure 7. Real-world use case (surface coating). This brownfield scenario demonstrates the conversion of dead floor space into an autonomous buffer. Unlike open grids, the layout is dictated by the footprint of the machinery. The visualisation highlights the heuristic's capability to assign inventory to separated zones (depth 1 vs. depth 2) to optimise storage utilisation within the irregular boundaries. (a) *Topology ($t = 001$):* The layout utilises the residual space around a high-gloss coating machine (grey) with adjacent Source (blue) and Sink (green). The niches are manually abstracted into static lanes with varying depths: orange/yellow zones (depth 1) and green zones (depth 2). (b) *Operational State ($t = 012$):* Early optimisation. The snapshot captures a reshuffling operation where AMR V2 relocates unit load UL 19 to the single-deep yellow zone (lane 16). In contrast, the lower-priority UL 20 is moved to the distant yellow zone (lane 8), preserving closer slots for urgent unit loads.

deep lanes show the highest cumulative occupancy (Figure 8(a)), indicating they are used to buffer unit loads with late deadlines. Conversely, slots adjacent to the Source and Sink exhibit the highest traffic density (Figure 8(b)) but comparatively low occupancy intensity. The heuristic adaptively treats these prime locations as a staging area with short buffer durations for immediate retrieval tasks.

(3) *Capacity-Adaptive Breathing Topology:* The algorithm automatically adjusts its active storage area based on current inventory levels. During periods of low utilisation, it allocates unit loads to nearby slots, minimising AMR travel distances and leaving distant areas empty. As storage density increases, the algorithm dynamically expands the active storage area into the further distant slots. This ensures strict travel time optimisation under normal conditions, while unlocking maximum capacity during peak congestion.

These behaviours suggest that the heuristic is capable of highly adaptive, context-aware decision making on complex floor plans.

6.4. Layout sensitivity and saturation points

To isolate the impact of layout topology from pure capacity, we expanded the evaluation to compare rectangular configurations (8×3) against dense square blocks (5×5, 6×6) and irregular real-world layouts (e.g. Figure 7). These scenarios were evaluated across fleet sizes ranging from $|\mathcal{V}| = 2$ to 6 AMRs. Since feasibility rates

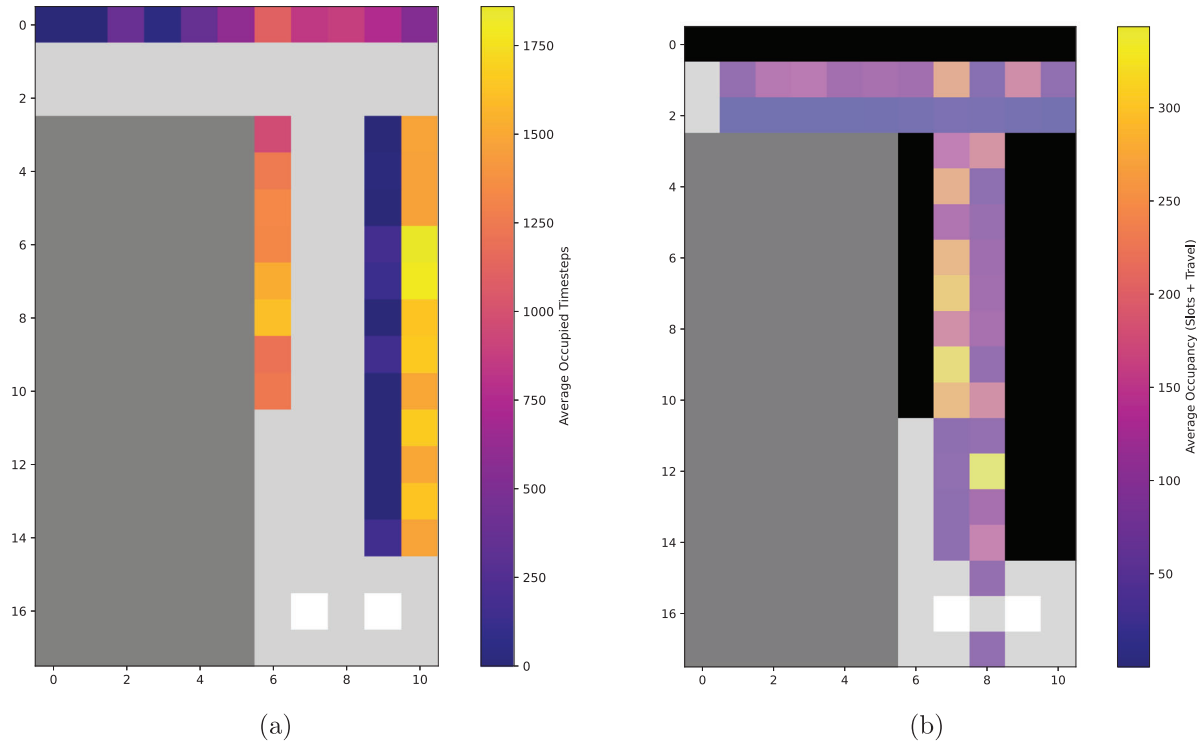

Figure 8. Algorithmic storage location assignment in the brownfield scenario. The heatmaps contrast buffer duration (a) with traffic flow (b), highlighting the differentiation between high-density storage and transfer zones. (a) *Occupancy Intensity:* Average duration a slot is occupied. Lighter colors indicate the slot is utilised most of the time. (b) *Traffic Density:* Frequency of AMR visits at access points. Lighter colors indicate the access point is utilised frequently.

remained consistent regardless of fleet density – confirming the heuristic's deadlock avoidance – we aggregate these results to focus on the more decisive structural factors: number of access points and lane-to-depth ratio.

Table 6 presents a sensitivity analysis, breaking down feasibility by the unit-load-to-slot-ratio and access constraints. The table lists the absolute number of solved instances versus the total number of generated instances for each configuration. The results highlight three critical findings regarding system stability:

Most notably, the coating industry layouts with high slot counts demonstrate exceptional stability compared to the theoretical block models. As shown in Table 6, feasibility remains at or near 100% up to a unit load-to-slot ratio of 0.9. Even at full capacity, the system successfully solves between 70% and 80% of instances. The decline in solvability for these instances is attributable to two converging factors: primarily, the time required for deep reshuffling exceeds the theoretical baselines derived from simplified travel estimates, rendering some instances structurally infeasible. Additionally, for theoretically solvable but complex instances, the A∗ search (Stage 2) becomes a computational bottleneck, hitting time limits due to the search space explosion.

The analysis of the 8 × 3 layout confirms that the topology dictates performance. Even at a saturation ratio of 1.0, this configuration maintains 90% feasibility provided it allows for 2-sided access (45/50 solved). This evidence suggests that a high number of independent access points facilitates parallel operations, effectively mitigating the congestion effects typically associated with high storage density.

In sharp contrast, deep square layouts (e.g. 6 × 6) exhibit significantly earlier saturation points. With limited access (1–2 sides), performance degrades rapidly even at moderate ratios (0.7). This confirms that for deep storage, optimisation cannot fully compensate for a lack of reshuffling space; consequently, either lower storage densities or full perimeter access (4 sides) are requisite for reliable solvability.

6.5. Managerial insights

The computational experiments provide guidelines for the design and operation of autonomous buffer zones. By synthesising the results, we derive four principles for managing floor storage.

6.5.1. Layout fragmentation and access efficiency

In brownfield facilities, contiguous areas for buffering are rarely available. The results on layout topology show that

Table 6. Sensitivity analysis: feasibility by unit load to slot ratio and access directions.

Layout topology	Ratio	Access directions				*Solverate*
		1	2	3	4	
0. Rectangle (8 × 3)	0.7	30/50	50/50	–	–	80.0%
	0.8	15/50	50/50	–	–	65.0%
	0.9	5/50	50/50	–	–	55.0%
	1.0	0/50	45/50	–	–	45.0%
Block (5 × 5)	0.7	0/50	85/100	–	50/50	67.5%
	0.8	0/50	70/100	–	50/50	60.0%
	0.9	0/50	20/100	–	45/50	32.5%
	1.0	0/50	10/100	–	40/50	25.0%
Block (6 × 6)	0.7	0/50	15/100	–	50/50	32.5%
	0.8	0/50	0/100	–	50/50	25.0%
	0.9	0/50	5/100	–	25/50	15.0%
	1.0	0/50	0/100	–	25/50	12.5%
Real-World (39 and 43 slots)	0.5–0.8	–	–	450/450	–	100.0%
	0.9	–	–	98/100	–	98.0%
	1.0	–	–	75/100	–	75.0%

Note: The table displays the success rate as a fraction (Solved/Generated) for various layouts.

spatial fragmentation is not a disadvantage. Small, distributed rectangular buffers (e.g. 8 × 3) outperform large, symmetrical block layouts with deeper lanes (e.g. 6 × 6), even if these distributed zones are located further away from the source and sink.

Strategic Insight: Buffering does not require consolidating space into a single zone. Utilising decentralised floor space near production lines is effective, provided these pockets maintain high access availability relative to storage slots. Distributed buffers with multiple access directions prevent bottlenecks more effectively than deep storage blocks.

6.5.2. Capacity-adaptive breathing topology

While traditional material handling often relies on static zoning or manual ABC-classification, our heuristic facilitates the emergence of a capacity-adaptive breathing topology. Heatmap analysis confirms that storage location assignment occurs autonomously, driven by the underlying cost function rather than pre-defined spatial rules.

Strategic Insight: The interplay between accessibility penalties (h_{block}) and sequence integrity (h_{prio}) forces a dynamic spatial stratification: urgent unit loads are concentrated at the periphery for immediate extraction, while less critical items are *inhaled* into deeper buffer zones to preserve access. This results in a *breathing* system footprint that expands or contracts its active storage area in real-time response to the production schedule. To the best of our knowledge, this represents one of the first integrated models to achieve such emergent spatial adaptability in a multi-robot environment governed by the simultaneous constraints of deep LIFO stacking and perimeter-only access.

6.5.3. The 90% stability threshold

The sensitivity analysis shows a stability threshold across all tested layouts. Solvability and system reliability decline when the unit-load-to-slot ratio exceeds 0.9. At higher ratios, the system lacks the unoccupied slots required for reshuffling.

Strategic Insight: Planners must distinguish between storage capacity and operational throughput capacity. A production buffer requires approximately 10% of slots as slack to resolve deadlocks. Operating above this 90% threshold transforms the buffer into a static storage yard, rendering it brittle to stochastic production requests.

6.5.4. Operational robustness and fleet scalability

The algorithm maintains robustness across varying fleet sizes from 2 to 6 robots without parameter retuning. The system utilises additional robots to reduce makespan until the maximum throughput capacity is reached.

Strategic Insight: This ensures operational robustness through fleet scalability. If a robot is removed for maintenance, the algorithm redistributes tasks and adapts the throughput capacity without causing an operational standstill. Process continuity is decoupled from the specific fleet count, providing investment security.

7. Conclusion and future work

This paper addressed the Multi-AMR Buffer Storage, Retrieval, and Reshuffling Problem (BSRRP) by establishing an Exact Formulation (EF) for benchmarking and proposing a hierarchical heuristic. Our research demonstrates that automating dense floor storage in brownfield environments requires the integration of reshuffling logic and multi-AMR coordination, as implemented in our two-stage approach.

The computational experiments reveal a disparity in tractability between exact and heuristic methods. While exact approaches often fail to converge for dense industrial scenarios, the hierarchical heuristic achieves speedup factors ranging from 450x to over 3,000x compared to the exact solver.

Regarding operational robustness, our analysis identified a saturation threshold at a unit load to slot ratio of 0.9. Beyond this ratio, the lack of reshuffling space renders the system brittle, confirming that 10% of capacity must be treated as slack to resolve deadlocks. Furthermore, strict time windows were identified as a source of infeasibility; treating deadlines as soft constraints allowed the heuristic to resolve high-traffic scenarios where the exact solver failed, confirming that flow continuity must take precedence over precision in time-critical environments.

Finally, the physical layout topology and the heuristic's capacity-adaptive behaviour play a decisive role. Our experiments demonstrate that rectangular layouts with high access availability outperform deep square configurations by enabling parallel access. The heuristic adaptively adjusts the storage footprint based on current inventory levels: it utilises nearby slots during low demand to minimise travel distances and expands into distant slots only when capacity requirements increase. This allows the system to optimise storage depth based on buffer duration without explicit pre-configuration, adapting autonomously to irregular brownfield constraints. This confirms that a decoupled approach, combining A^* search for task sequencing with Constraint Programming for scheduling, is a viable path for industrial control. While the Exact Formulation serves as a baseline, its complexity makes exact solvers impractical for real-time deployment. Future research may explore more compact formulations, though the high-frequency requirements of industrial buffers likely necessitate heuristic approaches.

Operationally, the current model is limited by deterministic environmental assumptions and idealised kinematics. Future work will focus on integrating kinematic constraints (e.g. acceleration profiles, turning radii) and energy management models into the scheduling logic. Additionally, implementing post-processing trajectory smoothing could further enhance path efficiency.

Algorithmically, profiling identifies the A^* search for task sequencing as the primary bottleneck in dense scenarios and highlights the trade-offs of using a virtual AMR substitute during sequencing. Future optimisation of this component – through pre-computed pattern databases or Multi-Agent Path Finding (MAPF) refinements – would eliminate remaining latency. Furthermore, this modular architecture establishes a data-driven infrastructure for online operations to enhance applicability in uncertain industrial settings. The decoupled solver can serve as a baseline for real-time decision-making, where the system must continuously adapt to dynamic production streams and stochastic arrival rates in a data-rich environment.

Acknowledgements

This research was presented as a work-in-progress at the VeRoLog conference 2025. The authors acknowledge the use of artificial intelligence (AI) tools during the preparation of this manuscript. Specifically, Gemini (versions 1.5 to 3.0; Google) and GitHub Copilot (Claude Sonnet 4.0 and 4.5) were used to help with aspects of software development, language improvement and editing, and the classification of the literature relevant to this study. The authors have reviewed and edited all content and assume full responsibility for the accuracy, integrity, and originality of the work presented.

Author contributions

CRediT: **Max Disselnmeyer:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft; **Thomas Bömer:** Conceptualization, Funding acquisition, Methodology, Writing – review & editing; **Laura Dörr:** Formal analysis, Validation, Writing – review & editing; **Bastian Amberg:** Formal analysis, Validation, Writing – review & editing; **Anne Meyer:** Conceptualization, Funding acquisition, Methodology, Resources, Supervision, Writing – review & editing

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the Federal Ministry for Economic Affairs and Climate Action (BMWK) [grant number 13IK0321]; and the European Union – NextGenerationEU.

Notes on contributors

Max Disselnmeyer is a Research Associate and doctoral candidate at the Institute for Information Management in Engineering (IMI), Karlsruhe Institute of Technology (KIT), Germany. His research interests include data science in production logistics and combinatorial optimisation. Specifically, he focuses on the design and operation of autonomous buffer zones using data science methods. His work combines theoretical scheduling approaches with the practical implementation of autonomous mobile robot (AMR) systems.

Thomas Bömer is a Research Associate and doctoral candidate at the Institute for Information Management in Engineering (IMI), Karlsruhe Institute of Technology (KIT), Germany. His research interests lie in combinatorial optimisation and automated heuristic design using large language models (LLMs), with a particular focus on logistics-oriented optimisation problems.

Laura Dörr is a postdoctoral researcher at the Institute for Information Management in Engineering (IMI), Karlsruhe Institute of Technology (KIT), Germany. She holds degrees in mathematics and computer science and completed her PhD on deep-learning-based vision systems for logistics applications. Her current research interests include generative and multimodal AI for engineering applications.

Bastian Amberg is a researcher at the Institute for Information Management in Engineering (IMI), Karlsruhe Institute of Technology (KIT), Germany. Prior to joining Prof. Anne Meyer's research group, he served as Assistant Professor of Information Systems with a focus on Advanced Decision Analytics at Freie Universität Berlin. His research centers on decision support systems for robust and efficient planning and control in operational and logistics contexts, combining mathematical optimisation and modern data science techniques.

Anne Meyer is a Professor of Data Science in Mechanical Engineering at the Institute for Information Management in Engineering (IMI), Karlsruhe Institute of Technology (KIT), Germany. Her research focuses on developing adaptive methods to optimise production and logistics systems by combining mathematical optimisation and machine learning.

ORCID

Max Disselnmeyer http://orcid.org/0009-0008-5689-2235
Thomas Bömer http://orcid.org/0000-0003-4979-7455
Laura Dörr http://orcid.org/0000-0002-8007-1815
Bastian Amberg http://orcid.org/0000-0001-6715-3819
Anne Meyer http://orcid.org/0000-0001-6380-1348

Data availability statement

The source code is available under https://github.com/mdisse/BSRRP_IP_Heuristic. Instances and solutions are available under https://doi.org/10.5281/zenodo.19222950.

References

Andulkar, M., D. T. Le, and U. Berger. 2018. "A Multi-case Study on Industry 4.0 for SME's in Brandenburg, Germany." In *Proceedings of the 51st Hawaii International Conference on System Sciences*. https://doi.org/10.24251/HICSS.2018.574.

Archetti, C., L. Coelho, M. Speranza, and P. Vansteenwegen. 2025. "Beyond Fifty Years of Vehicle Routing: Insights into the History and the Future." *European Journal of Operational Research* 330:355–372. https://doi.org/10.1016/j.ejor.2025.06.014.

Boge, S., and S. Knust. 2020. "The Parallel Stack Loading Problem Minimizing the Number of Reshuffles in the Retrieval Stage." *European Journal of Operational Research* 280 (3): 940–952. https://doi.org/10.1016/j.ejor.2019.08.005.

Bömer, T., M. Disselnmeyer, and A. Meyer. 2025. "A Constraint Programming Approach for the Multi-robot Multi-bay Unit Load Pre-marshalling Problem." *Procedia CIRP* 134:508–513. https://doi.org/10.1016/j.procir.2025.02.151.

Bömer, T., N. Koltermann, J. Pfrommer, and A. Meyer. 2024. "Sorting Multibay Block Stacking Storage Systems with Multiple Robots." In *International Conference on Computational Logistics*, 34–48. https://doi.org/10.1007/978-3-031-71993-6.

Bömer, T., J. Pfrommer, D. Akizhanov, and A. Meyer. 2026. "Sorting Multi–bay Block Stacking Storage Systems." *Computers & Operations Research* 188:107359. https://doi.org/10.1016/j.cor.2025.107359.

Borjian, S., V. H. Manshadi, C. Barnhart, and P. Jaillet. 2015. "Managing Relocation and Delay in Container Terminals with Flexible Service Policies." arXiv.1503.01535.

Buckow, J. N.., M. Goerigk, and S. Knust. 2025. "Integrated Pallet Retrieval and Processing in Warehouses under Uncertainty." *OR Spectrum* 47 (3): 817–855. https://doi.org/10.1007/s00291-024-00806-7.

Caserta, M., S. Schwarze, and S. Voß. 2012. "A Mathematical Formulation and Complexity Considerations for the Blocks Relocation Problem." *European Journal of Operational Research* 219 (1): 96–104. https://doi.org/10.1016/j.ejor.2011.12.039.

Charris, E., J. Rojas-Reyes, and J. Montoya-Torres. July, 2018. "The Storage Location Assignment Problem: A Literature Review." *International Journal of Industrial Engineering Computations* 10: 1–26.

Chen, C., L. K. Tiong, and I. M. Chen. 2019. "Using a Genetic Algorithm to Schedule the Space-Constrained AGV-based Prefabricated Bathroom Units Manufacturing System." *International Journal of Production Research* 57 (10): 3003–3019. https://doi.org/10.1080/00207543.2018.1535205.

Dantzig, G. B., and J. H. Ramser. 1959. "The Truck Dispatching Problem." *Management Science* 6 (1): 80–91. https://doi.org/10.1287/mnsc.6.1.80.

Descartes Systems Group. May, 2023. "How Bad is the Supply Chain and Logistics Workforce Challenge?" Accessed January 28, 2026. https://www.descartes.com/resources/news/descartes-study-reveals-76-supply-chain-and-logistics-operations-are-experiencing.

Disselnmeyer, M., T. Bömer, J. Pfrommer, and A. Meyer. 2024. "The Static Buffer Reshuffling and Retrieval Problem for Autonomous Mobile Robots." In *International Conference on Computational Logistics*, 18–33. Springer. https://doi.org/10.1007/978-3-031-71993-6_2.

ElWakil, M., A. Eltawil, and M. Gheith. 2022. "On the Integration of the Parallel Stack Loading Problem with the Block Relocation Problem." *Computers & Operations Research* 138:105609. https://doi.org/10.1016/j.cor.2021.105609.

Fragapane, G., R. de Koster, F. Sgarbossa, and J. O. Strandhagen. 2021. "Planning and Control of Autonomous Mobile Robots for Intralogistics: Literature Review and Research Agenda." *European Journal of Operational Research* 294 (2): 405–426. https://doi.org/10.1016/j.ejor.2021.01.019.

Fu, B., Z. Chen, R. Chandan, A. Barbosa, M. Caldara, J. Durham, and F. Pecora. 2026. "Symbolic Planning and Multi-agent Path Finding in Extremely Dense Environments with Unassigned Agents." arXiv.2509.01022.

Ge, P., Y. Meng, J. Liu, L. Tang, and R. Zhao. 2020. "Logistics Optimisation of Slab pre-marshalling Problem in Steel Industry." *International Journal of Production Research* 58 (13): 4050–4070. https://doi.org/10.1080/00207543.2019.1641238.

Geft, T., W. Zhang, J. Yu, and K. Bekris. 2026. "Robust Out-of-order Retrieval for Grid-Based Storage at Maximum Capacity." arXiv.2601.19144.

Hu, S., S. Zhao, and Z. Ren. 2025. "Conflict-Based Search and Prioritized Planning for Multi-agent Path Finding among Movable Obstacles." arXiv.2509.26050.

Hua, Y., Y. Wang, and Z. Ji. June, 2024. "Adaptive Lifelong Multi-agent Path Finding with Multiple Priorities." *IEEE Robotics and Automation Letters* 9 (6): 5607–5614. https://doi.org/10.1109/LRA.2024.3392084.

Ji, M., W. Guo, H. Zhu, and Y. Yang. 2015. "Optimization of Loading Sequence and Rehandling Strategy for Multi-quay Crane Operations in Container Terminals." *Transportation Research Part E: Logistics and Transportation Review* 80:1–19. https://doi.org/10.1016/j.tre.2015.05.004.

Kim, K. H., and G. P. Hong. 2006. "A Heuristic Rule for Relocating Blocks." *Computers& Operations Research* 33 (4): 940–954. https://doi.org/10.1016/j.cor.2004.08.005.

Kizilay, D., and D. T. Eliiyi. 2021. "A Comprehensive Review of Quay Crane Scheduling, Yard Operations and Integrations Thereof in Container Terminals." *Flexible Services and Manufacturing Journal* 33 (1): 1–42. https://doi.org/10.1007/s10696-020-09385-5.

Lersteau, C., and W. Shen. 2022. "A Survey of Optimization Methods for Block Relocation and PreMarshalling Problems." *Computers& Industrial Engineering* 172:108529. https://doi.org/10.1016/j.cie.2022.108529.

Lowerre, B. T. 1976. *The Harpy Speech Recognition System*. Carnegie Mellon University.

Makino, H., and S. Ito. 2025. "Mapf-hd: Multi-agent Path Finding in High-Density Environments." arXiv.2509.06374.

Pfrommer, J., A. Meyer, and K. Tierney. 2022. "Solving the Unit-Load Pre-Marshalling Problem in Block Stacking Storage Systems with Multiple Access Directions." Preprint. arXiv:2207.09118.

Pfrommer, J., A. Meyer, and K. Tierney. 2024. "Solving the Unit-Load pre-marshalling Problem in Block Stacking Storage Systems with Multiple Access Directions." *European Journal of Operational Research* 313 (3): 1054–1071. https://doi.org/10.1016/j.ejor.2023.08.044.

Pytel, S., S. Sitek, M. Chmielewska, E. Zuzańska-Żyśko, A. Runge, and J. Markiewicz-Patkowska. 2021. "Transformation Directions of Brownfields: The Case of the Górnośląsko-Zagłębiowska Metropolis." *Sustainability* 13 (4): 2075. https://doi.org/10.3390/su13042075.

Tang, L., and H. Ren. 2010. "Modelling and a Segmented Dynamic Programming-Based Heuristic Approach for the Slab Stack Shuffling Problem." *Computers& Operations Research* 37 (2): 368–375. https://doi.org/10.1016/j.cor.2009.05.011.

Teck, S., R. Dewil, and P. Vansteenwegen. 2024. "A Simulation-Based Genetic Algorithm for a Semi-automated Warehouse Scheduling Problem with Processing Time Variability." *Applied Soft Computing* 160:111713. https://doi.org/10.1016/j.asoc.2024.111713.

van Gils, T., A. Caris, K. Ramaekers, and K. Braekers. 2019. "Formulating and Solving the Integrated Batching, Routing, and Picker Scheduling Problem in a Real-Life Spare Parts Warehouse." *European Journal of Operational Research* 277 (3): 814–830. https://doi.org/10.1016/j.ejor.2019.03.012.

Vis, I. F. 2006. "Survey of Research in the Design and Control of Automated Guided Vehicle Systems." *European Journal of Operational Research* 170 (3): 677–709. https://doi.org/10.1016/j.ejor.2004.09.020.

Wang, Q., R. Veerapaneni, Y. Wu, J. Li, and M. Likhachev. May, 2024. "MAPF in 3D Warehouses: Dataset and Analysis." *Proceedings of the International Conference on Automated Planning and Scheduling* 34:623–632. https://doi.org/10.1609/icaps.v34i1.31525.

Wang, Z., C. Zhou, A. Che, and J. Gao. 2024. "A Policy-Based Monte Carlo Tree Search Method for Container pre-marshalling." *International Journal of Production Research* 62 (13): 4776–4792. https://doi.org/10.1080/00207543.2023.2279130.

Appendices

Appendix 1. Multi-AMR buffer storage, retrieval, and reshuffling problem – complete model

$$b_{ijnt} = \begin{cases} 1 & \text{if unit load } n \text{ is in } [i,j] \text{ at time } t, \\ 0 & \text{otherwise;} \end{cases}$$
$$\forall\, i \in \mathcal{I},\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{A1}$$

$$x_{ijklntv} = \begin{cases} 1 & \text{if unit load } n \text{ is relocated from } [i,j] \text{ to } [k,l] \\ & \quad \text{at time } t \text{ by AMR } v, \\ 0 & \text{otherwise;} \end{cases}$$
$$\forall\, i,k \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, l \in \mathcal{J}_k,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{A2}$$

$$y_{ijntv} = \begin{cases} 1 & \text{if unit load } n \text{ is retrieved from } [i,j] \\ & \quad \text{at time } t \text{ by AMR } v, \\ 0 & \text{otherwise;} \end{cases}$$
$$\forall\, i \in \mathcal{I} \setminus \{I\},\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{A3}$$

$$g_{nt} = \begin{cases} 1 & \text{if unit load } n \text{ has been retrieved at time} \\ & \quad t' \in \{1,\ldots,t-1\}, \\ 0 & \text{otherwise;} \end{cases}$$
$$\forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{A4}$$

$$z_{ijntv} = \begin{cases} 1 & \text{if unit load } n \text{ is stored in } [i,j] \\ & \quad \text{at time } t \text{ by AMR } v, \\ 0 & \text{otherwise;} \end{cases}$$

$$\forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{A5}$$

$$s_{nt} = \begin{cases} 1 & \text{if unit load } n \text{ has been stored at time} \\ & \quad t' \in \{1, \ldots, t-1\}, \\ 0 & \text{otherwise;} \end{cases}$$

$$\forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{A6}$$

$$e_{ijkltv} = \begin{cases} 1 & \text{if the AMR drives from slot } [i,j] \\ & \quad \text{to slot } [k,l] \text{ at time } t, \\ & \text{without transporting a unit load} \\ 0 & \text{otherwise;} \end{cases}$$

$$\forall\, i,k \in \mathcal{I},\ \forall\, j \in \mathcal{J}_i,\ \forall\, l \in \mathcal{J}_k,\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{A7}$$

$$c_{ijtv} = \begin{cases} 1 & \text{if the AMR } v \text{ is at slot } [i,j] \text{ at time } t, \\ 0 & \text{otherwise;} \end{cases}$$

$$\forall\, i \in \mathcal{I},\ \forall\, j \in \mathcal{J}_i,\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{A8}$$

$$T = \max_{n,m \in \mathcal{N}} \{a_n + \alpha_n, r_m + \rho_m\} \tag{A9}$$

$$\tau_{ijkl} = \begin{cases} \max(1, d_{ijkl}) & \text{if performing an empty} \\ & \quad \text{drive } e_{ijklt}, \\ \max(1, d_{ijkl} + 2h) & \text{otherwise;} \end{cases} \tag{A10}$$

Starting Constraints

$$b_{ijn1} = \begin{cases} 1, & \text{if unitload } n \text{ starts in slot } [i,j] \\ 0, & \text{otherwise} \end{cases}$$

$$\forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N} \tag{A11}$$

$$s_{n1} = \begin{cases} 1, & \text{if unitload is initially stored in buffer zone} \\ 0, & \text{otherwise} \end{cases}$$

$$\forall\, n \in \mathcal{N} \tag{A12}$$

$$c_{ij1v} = \begin{cases} 1, & \text{if vehicle } v \text{ starts in slot } [i,j] \\ 0, & \text{otherwise} \end{cases}$$

$$\forall\, i \in \mathcal{I},\ \forall\, j \in \mathcal{J}_i,\ \forall\, v \in \mathcal{V} \tag{A13}$$

Objective function

$$\min \quad \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}_i} \sum_{t\in\mathcal{T}} \sum_{v\in\mathcal{V}} \Bigg(\sum_{n\in\mathcal{N}} \left(y_{ijntv} * d_{ijI1} + z_{ijntv} * d_{01ij} \right) + \sum_{k\in\mathcal{I}} \sum_{l\in\mathcal{J}_k} \left(\sum_{n\in\mathcal{N}} x_{ijklntv} + e_{ijkltv} \right) * d_{ijkl} \Bigg) \tag{A14}$$

Subject to:

$$\sum_{i\in\mathcal{I}'} \sum_{j\in\mathcal{J}_i} b_{ijnt} \le s_{nt}, \quad \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{A15}$$

$$g_{nt} \ge s_{nt} - \sum_{i\in\mathcal{I}'} \sum_{j\in\mathcal{J}_i} b_{ijnt}, \quad \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{A16}$$

$$\sum_{n\in\mathcal{N}} b_{ijnt} \le 1, \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, t \in \mathcal{T} \tag{A17}$$

$$\sum_{n\in\mathcal{N}} b_{ijnt} \ge \sum_{n\in\mathcal{N}} b_{ij+1nt}, \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i \setminus J_i,\ \forall\, t \in \mathcal{T} \tag{A18}$$

$$\sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} \sum_{n\in\mathcal{N}} x_{ijklntv} + \sum_{k\in\mathcal{I}} \sum_{l\in\mathcal{J}_k} e_{ijkltv} + \sum_{n\in\mathcal{N}} y_{ijntv} \le c_{ijtv}, \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{A19}$$

$$\sum_{k\in\mathcal{I}} \sum_{l\in\mathcal{J}_k} e_{I1kltv} \le c_{I1tv}, \quad \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{A20}$$

$$\sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} z_{klntv} + \sum_{k\in\mathcal{I}} \sum_{l\in\mathcal{J}_k} e_{01kltv} + y_{01ntv} \le c_{01tv}, \quad \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{A21}$$

$$\sum_{v\in\mathcal{V}} y_{01ntv} + s_{nt} \le 1, \quad \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{A22}$$

$$\sum_{v\in\mathcal{V}} \left(y_{ijntv} + \sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} x_{ijklntv} \right) \le b_{ijnt}, \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{A23}$$

$$b_{ijnt} = b_{ijn(t-1)} + \sum_{v\in\mathcal{V}} \Bigg[\sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} \left(x_{klijn(t-\tau_{klij})v} - x_{ijkln(t-1)v} \right) - y_{ijn(t-1)v} + z_{ijn(t-\tau_{01ij})v} \Bigg], \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \setminus 1 \tag{A24}$$

$$g_{nt} = \sum_{i\in\mathcal{I}\setminus I} \sum_{j\in\mathcal{J}_i} \sum_{t'=1}^{t-1} \sum_{v\in\mathcal{V}} y_{ijnt'v}, \quad \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{A25}$$

$$s_{nt} = s_{n1} + \sum_{i\in\mathcal{I}'} \sum_{j\in\mathcal{J}_i} \sum_{t'=1}^{t-\tau_{01ij}} \sum_{v\in\mathcal{V}} z_{ijnt'v}, \quad \forall\, n \in \mathcal{N},\ \forall\, t \in \mathcal{T} \tag{A26}$$

$$c_{ijtv} = c_{ij(t-1)v} + \sum_{n\in\mathcal{N}} z_{ijn(t-\tau_{01ij})v} - \sum_{n\in\mathcal{N}} y_{ijn(t-1)v} + \sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} \sum_{n\in\mathcal{N}} x_{klijn(t-\tau_{klij})v} + \sum_{k\in\mathcal{I}} \sum_{l\in\mathcal{J}_k} e_{klij(t-\tau_{klij})v} - \sum_{k\in\mathcal{I}'} \sum_{l\in\mathcal{J}_k} \sum_{n\in\mathcal{N}} x_{ijkln(t-1)v} - \sum_{k\in\mathcal{I}} \sum_{l\in\mathcal{J}_k} e_{ijkl(t-1)v}, \quad \forall\, i \in \mathcal{I}',\ \forall\, j \in \mathcal{J}_i,\ \forall\, t \in \mathcal{T} \setminus 1,\ \forall\, v \in \mathcal{V} \tag{A27}$$

$$c_{I1tv} = c_{I1(t-1)v} + \sum_{i\in\mathcal{I}\setminus I} \sum_{j\in\mathcal{J}_i} \sum_{n\in\mathcal{N}} y_{ijn(t-\tau_{ijI1})v} + \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}_i} \left(e_{ijI1(t-\tau_{ijI1})v} - e_{I1ij(t-1)v} \right),$$

$$\forall\, t \in \mathcal{T} \setminus 1,\ \forall\, v \in \mathcal{V} \tag{A28}$$

$$c_{01tv} = c_{01(t-1)v} - \sum_{i\in\mathcal{I}'}\sum_{j\in\mathcal{J}_i}\sum_{n\in\mathcal{N}} z_{ijn(t-1)v} + \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}_i}\left(e_{ij01(t-\tau_{ijIJ})v} - e_{01ij(t-1)v}\right) - \sum_{n\in\mathcal{N}} y_{01n(t-1)v}, \quad \forall\, t \in \mathcal{T} \setminus 1,\ \forall\, v \in \mathcal{V} \tag{A29}$$

$$\sum_{i\in\mathcal{I}\setminus I}\sum_{j\in\mathcal{J}_i}\sum_{t=1}^{r_n-\tau_{ijI1}-1}\sum_{v\in\mathcal{V}} y_{ijntv} = 0, \quad \forall\, n \in \mathcal{N} \tag{A30}$$

$$\sum_{i\in\mathcal{I}\setminus I}\sum_{j\in\mathcal{J}_\rangle}\sum_{t=r_n-\tau_{ijI1}}^{r_n+\rho_n-\tau_{ijI1}}\sum_{v\in\mathcal{V}} y_{ijntv} = 1, \quad \forall\, n \in \mathcal{N} \tag{A31}$$

$$\sum_{i\in\mathcal{I}\setminus I}\sum_{j\in\mathcal{J}_i}\sum_{t=r_n+\rho_n-\tau_{ijI1}+1}^{T}\sum_{v\in\mathcal{V}} y_{ijntv} = 0, \quad \forall\, n \in \mathcal{N} \tag{A32}$$

$$\sum_{i\in\mathcal{I}'}\sum_{j\in\mathcal{J}_i}\sum_{t=1}^{a_n-1}\sum_{v\in\mathcal{V}} z_{ijntv} = 0, \quad \forall\, n \in \mathcal{N} \tag{A33}$$

$$\sum_{i\in\mathcal{I}'}\sum_{j\in\mathcal{J}_i}\sum_{t=a_n}^{a_n+\alpha_n}\sum_{v\in\mathcal{V}} z_{ijntv} + \sum_{t=r_n-\tau_{01I1}}^{\min(r_n+\rho_n-\tau_{01I1},a_n+\alpha_n)}\sum_{v\in\mathcal{V}} y_{01ntv} = 1, \quad \forall\, n \in \mathcal{N} \tag{A34}$$

$$\sum_{i\in\mathcal{I}'}\sum_{j\in\mathcal{J}_i}\sum_{t=a_n+\alpha_n+1}^{T}\sum_{v\in\mathcal{V}} z_{ijntv} = 0, \quad \forall\, n \in \mathcal{N} \tag{A35}$$

$$\sum_{v\in\mathcal{V}}\left[\sum_{j\in\mathcal{J}_i} c_{ijtv} + \sum_{j\in\mathcal{J}_i}\left(\sum_{n\in\mathcal{N}}\sum_{t'\in\Omega(t,\tau_{lane}(j)+h)} z_{ijnt'v} + \sum_{k\in\mathcal{I}'}\sum_{l\in\mathcal{J}_k}\sum_{n\in\mathcal{N}}\sum_{t'\in\Omega(t,\tau_{lane}(j)+h)} x_{klijnt'v} + \sum_{k\in\mathcal{I}}\sum_{l\in\mathcal{J}_k}\sum_{t'\in\Omega(t,\tau_{lane}(j))} e_{klijt'v}\right) + \sum_{j\in\mathcal{J}_i}\left(\sum_{n\in\mathcal{N}}\sum_{t'\in\Omega^*(t,\tau_{lane}(j)+h)} y_{ijnt'v} + \sum_{k\in\mathcal{I}'}\sum_{l\in\mathcal{J}_k}\sum_{n\in\mathcal{N}}\sum_{t'\in\Omega^*(t,\tau_{lane}(j)+h)} x_{ijklnt'v} + \sum_{k\in\mathcal{I}}\sum_{l\in\mathcal{J}_k}\sum_{t'\in\Omega^*(t,\tau_{lane}(j))} e_{ijklt'v}\right)\right] \leq 1 \quad \forall\, i \in \mathcal{I}',\ \forall\, t \in \mathcal{T} \tag{A36}$$

$$\sum_{n\in\mathcal{N}}(x_{ijklntv} + y_{ijntv} + e_{ijkltv}) \leq 1 - \sum_{n\in\mathcal{N}} b_{i(j+1)nt}, \quad \forall\, i \in \mathcal{I}',$$

Table A1. Sets, indices, and general notation for the Integer Programming model.

Element	Notation	Description
Sets and Indices		
Set of unit loads	$\mathcal{N}, n$	Range: $\{1, 2, \ldots, N\}$
Set of time steps	$\mathcal{T}, t$	Range: $\{1, 2, \ldots, T\}$
Set of all lanes	$\mathcal{I}, i, k$	Range: $\{0, 1, \ldots, I\}$
Set of buffer lanes	$\mathcal{I}'$	Range: $\{1, 2, \ldots, I-1\}$ (excluding sink and source)
Set of slots	$\mathcal{J}_{i,k}, j, l$	Range: $\{1, 2, \ldots, J\}$
Set of vehicles	$\mathcal{V}, v$	Range: $\{1, 2, \ldots, V\}$
General Notation and Parameters		
Unit load	(u_n, r_n, a_n)	A unit load with label u_n. Retrieval window opens at r_n, arrival window at a_n.
Slot	$[i, j]$	Slot in the ith lane and jth slot (deepest $j = 1$, at the perimeter $j = J$).
Outermost slot	$[i, J_i]$	The outermost slot J_i in lane i.
Sink	$[I, 1] = [I, J_I]$	Modelled as a slot in the lane with the highest value for i.
Source	$[0, 1] = [0, J_I]$	Modelled as a slot in the lane with the lowest value for i.
Relocation move	$[i, j] \rightarrow [k, l]$	A relocation move from slot $[i, j]$ to $[k, l]$.
Retrieval move	$[i, j] \rightarrow [I, J]$	Retrieving a unit load from slot $[i, j]$.
Empty drive	$[i, j] \hookrightarrow [k, l]$	Driving the AMR from slot $[i, j]$ to $[k, l]$ without transporting a load.
Storage move	$[0, 1] \rightarrow [i, j]$	Storing a unit load to slot $[i, j]$.
Retrieval window	$[r_n, r_n + \rho_n]$	Time window for retrieval of u_n; ρ_n is the maximum delay.
Arrival window	$[a_n, a_n + \alpha_n]$	Time window for arrival of u_n; α_n is the maximum delay.
Distance	d_{ijkl}	Distance between the slots $[i, j]$ and $[k, l]$.
Travel time	τ_{ijkl}	Travel time (incl. handling time) between the slots $[i, j]$ and $[k, l]$.

$$\forall\, j \in \mathcal{J}_i \setminus \{1\},\ \forall\, k \in \mathcal{I}',\ \forall\, l \in \mathcal{J}_k,\ \forall\, t \in \mathcal{T},\ \forall\, v \in \mathcal{V} \tag{A37}$$

Appendix 2. NP-hardness of the BSRRP problem

The Block Relocation Problem is known to be NP-hard (Caserta, Schwarze, and Voß 2012). We establish the NP-hardness of the Buffer Storage, Retrieval, and Reshuffling Problem by providing a polynomial-time reduction from any instance of BRP to BSRRP.

A.1 Problem definitions

Definition A.1 (BRP – Block Relocation Problem): Given:

- A set of containers $C = \{c_1, \ldots, c_N\}$
- A set of stacks $S = \{s_1, \ldots, s_M\}$ with maximum height H
- A priority function $p : C \rightarrow \{1, \ldots, N\}$ assigning unique priorities to containers
- An initial configuration function $f : C \rightarrow S \times \{1, \ldots, H\}$ mapping containers to positions

Find a sequence of relocations minimising the total number of moves k while retrieving containers in ascending priority order.

Definition A.2 (BSRRP – Buffer Storage, Retrieval, and Reshuffling Problem): Given:

- A set of unit loads $U = \{u_1, \ldots, u_N\}$
- A buffer zone with lanes $L = \{l_1, \ldots, l_M\}$ of maximum depth H
- Time windows $[e_i, l_i]$ for each unit load u_i
- An initial configuration function $g : U \to L \times \{1, \ldots, H\}$

Find a sequence of relocations minimising total travel distance D while retrieving unit loads within their time windows.

A.2 Polynomial-time reduction

We present a polynomial-time transformation T that maps any instance I_{BRP} of BRP to an instance $I_{BSRRP} = T(I_{BRP})$ of BSRRP.

Definition A.3 (Transformation T): Let $I_{BRP} = (C, S, p, f, H)$ be any instance of BRP. We construct $I_{BSRRP} = (U, L, W, g, H)$ as follows:

(1) *Structure Preservation:*
   - Create a set of unit loads U such that $|U| = |C|$ (same number of elements)
   - Create a set of Lanes L such that $|L| = |S|$ (same number of stacks/lanes)
   - For each container $c_i \in C$, create a corresponding unit load $u_i \in U$
   - For each container position $(s, h) = f(c_i)$, set the position of the corresponding unit load $g(u_i) = (l, h)$ where l is the lane corresponding to stack s

(2) *Time Window Construction:* Let $k_{\max}(N) = \frac{N(N-1)}{2}$ be the maximum possible relocations for $N = |U| = |C|$ unit loads.

   For each u_i corresponding to container c_i with priority $p(c_i) = i$:
   - $e_i = (i-1) * (k_{\max}(N) + 1) + 1$
   - $l_i = i * (k_{\max}(N) + 1)$

(3) *Distance Metric:* Define the distance function $d(x, y)$ between any two positions x, y in the BSRRP instance as follows:

$$d(x,y) = \begin{cases} 1, & \text{if the move from } x \text{ to } y \text{ involves carrying a unit load (loaded move)} \\ 0, & \text{if the move from } x \text{ to } y \text{ does not involve carrying a unit load (unloaded move)} \end{cases}$$

This distance metric directly counts the number of loaded moves (relocations and retrievals), since unloaded moves contribute to zero distance.

A.3 Proof of correctness

Lemma A.1 (Correctness of Mapping T): *The mapping of containers c_i to unit loads u_i and stacks s_j to lanes l_j defined by the transformation T preserves all accessibility relationships between the elements.*

Proof: As described in the definition of transformation T, the following holds:

- For each $c_i \in C$, there exists a corresponding $u_i \in U$.
- For each $s_j \in S$, there exists a corresponding $l_j \in L$.
- The position of each element u_i in I_{BSRRP} corresponds to the position of the corresponding element c_i in I_{BRP} (i.e. if $f(c_i) = (s, h)$, then $g(u_i) = (l, h)$, where l is the lane corresponding to s).

Therefore, for any containers $c_i, c_j \in C$:

- If c_i blocks c_j in I_{BRP}, then u_i blocks u_j in I_{BSRRP}.
- If c_i is accessible in I_{BRP}, then u_i is accessible in I_{BSRRP}.
- The mapping g preserves the relative positions of all elements.

Thus, any valid access and relocation sequence in one problem has a corresponding valid sequence in the other problem. ■

Lemma A.2 (Time Window Correctness): *The construction of the time window enforces the priority ordering of BRP while allowing for all necessary relocations for each unit load in I_{BSRRP}.*

Proof: The time windows $[e_i, l_i]$ for each unit load u_i (corresponding to container c_i with BRP priority i) are defined as $e_i = (i-1)(k_{\max}(N) + 1) + 1$ and $l_i = i(k_{\max}(N) + 1)$. This construction creates sequential and strictly nonoverlapping time windows, such as $e_i = l_{i-1} + 1$. Consequently, operations related to u_i can only begin after the time window for u_{i-1} has closed, thus directly enforcing the ascending priority order of the BRP.

The duration of each time window for u_i is $l_i - e_i + 1 = k_{\max}(N) + 1$ time units. In the BSRRP instance, each unit of time corresponds to a loaded move (either a relocation or a retrieval), according to the defined distance metric. The value $k_{\max}(N) = \frac{N(N-1)}{2}$ represents an upper bound on the total number of relocations required for the entire BRP instance. Allocating $k_{\max}(N) + 1$ time units (i.e. potential loaded moves) for each individual unit load u_i is therefore amply sufficient to accommodate:

- Any relocations necessary to access u_i after $u_1, \ldots, u_{i-1}$ have been retrieved. The number of such relocations for u_i alone will not exceed $N-1$, which is less than or equal to $k_{max}(N)$ for $N \geq 2$.
- The single loaded move required for the retrieval of u_i itself.

Thus, any valid sequence of BRP operations can be mapped to the BSRRP instance within the constructed time windows, respecting the priority order and allowing for all required relocations and retrievals. ■

Theorem A.1 (Solution Equivalence): *An optimal solution to I_{BRP} with k relocations exists if and only if an optimal solution to I_{BSRRP} with total distance $D = k + N$ exists.*

Proof: Let Sol_{BRP} be a feasible sequence of operations for I_{BRP} involving k relocations and N mandatory retrievals. Let Sol_{BSRRP} be the corresponding sequence of operations for I_{BSRRP}.

Under the defined distance metric, unloaded moves in Sol_{BSRRP} have a distance of 0. Loaded moves have a distance

of 1. Each of the k relocations in Sol_{BRP} corresponds to exactly one loaded move in Sol_{BSRRP} (moving the relocated item). Each of the N retrievals in Sol_{BRP} corresponds to exactly one loaded move in Sol_{BSRRP} (moving the retrieved item out).

Therefore, the total distance D for Sol_{BSRRP} is precisely the sum of the distances of the loaded moves: $D = (k \times 1) + (N \times 1) = k + N$.

($\Rightarrow$) Optimal BRP Solution implies Optimal BSRRP Solution: Assume Sol^*_{BRP} is an optimal solution to I_{BRP} with k^* relocations. The corresponding Sol^*_{BSRRP} has total distance $D^* = k^* + N$. Suppose, for contradiction, that Sol^*_{BSRRP} is not optimal for I_{BSRRP}. Then there must exist a different feasible solution Sol'_{BSRRP} with total distance $D' < D^*$. Since distance only counts loaded moves, D' must correspond to some number of relocations k' and the N retrievals, such that $D' = k' + N$. Given $D' < D^*$, we have $k' + N < k^* + N$, which implies $k' < k^*$. The sequence Sol'_{BSRRP} corresponds to a valid sequence Sol'_{BRP} with k' relocations (due to Lemmas 4.1 and 4.2). This contradicts the assumption that Sol^*_{BRP} with k^* relocations was optimal for I_{BRP}. Therefore, Sol^*_{BSRRP} must be optimal for I_{BSRRP}.

($\Leftarrow$) Optimal BSRRP Solution implies Optimal BRP Solution: Assume Sol^*_{BSRRP} is an optimal solution to I_{BSRRP} with total distance D^*. As shown above, D^* must be equal to the number of loaded moves, which corresponds to k^* relocations and N retrievals, so $D^* = k^* + N$. This solution Sol^*_{BSRRP} corresponds to a valid BRP solution Sol^*_{BRP} with $k^* = D^* - N$ relocations. Suppose, for contradiction, that Sol^*_{BRP} is not optimal for I_{BRP}. Then there must exist a different feasible solution Sol'_{BRP} with k' relocations such that $k' < k^*$. From the ($\Rightarrow$) direction, this Sol'_{BRP} corresponds to a BSRRP solution Sol'_{BSRRP} with total distance $D' = k' + N$. Since $k' < k^*$, it follows that $D' < D^*$. This contradicts the assumption that Sol^*_{BSRRP} with distance D^* was optimal for I_{BSRRP}. Therefore, Sol^*_{BRP} must be optimal for I_{BRP}.

Thus, an optimal solution with k relocations exists for I_{BRP} if and only if an optimal solution with total distance $D = k + N$ exists for I_{BSRRP}. ■

A.4 Polynomial-time reduction complexity

The transformation T operates in polynomial time with respect to the input size (primarily N):

- Configuration mapping: $O(N)$
- Time window computation: The loop runs N times with constant time arithmetic operations inside. $O(N)$.
- Distance metric definition: $O(1)$

Thus, the transformation T is polynomial-time.

A.5 Conclusion

Since the reduction is polynomial-time, while it preserves solution feasibility and optimality for all instances, and the BRP is proven NP-hard, the BSRRP is also NP-hard.