Complete, Scalable, and Robust Prioritized Planning for Multi-Robot Ordered Storage and Retrieval at Maximum Capacity

William Zhang, Tzvika Geft, Jingjin Yu, and Kostas Bekris

Department of Computer Science, Rutgers University, New Brunswick, NJ, USA

Abstract. Automated warehouses face a fundamental trade-off between maximizing storage density and achieving high retrieval throughput. While puzzle-based storage (PBS) architectures increase capacity by eliminating aisles, coordinating multiple robots in these high-density spaces is computationally challenging due to the potential for deadlocks. This paper introduces a novel multi-robot formulation for the "ordered storage and retrieval problem at maximum capacity". The focus is on rectangular grids accessible from a single boundary, where loads need to be first stored up to full capacity and then efficiently retrieved, given a planned departure sequence. This work bridges the gap between geometric feasibility and execution efficiency by leveraging the properties of relocation-free arrangements. These properties guide an online, prioritized multi-agent path-finding algorithm, which is the main contribution of this work. Unlike general centralized planners, the approach exploits the specific invariants of the storage layout to guarantee completeness and prevent deadlocks, enabling scalability. Experiments demonstrate that the method achieves near-linear improvement in makespan with respect to the number of robots, up to $m = C$, where C is the grid width. Crucially, the algorithmic overhead of supporting robustness is negligible; the system handles uncertainty in departure sequences using robust storage arrangements with no significant penalty in execution speed compared to the non-robust baseline.

Keywords: Multi-robot Coordination · Path Planning · Logistics.

1 Introduction

The rapid expansion of global e-commerce has placed unprecedented pressure on automated warehouses to maximize two competing, yet equally critical, metrics for return on investment: storage density and retrieval throughput. Modern facilities increasingly adopt high-density warehouse designs to reduce the physical footprint of inventory, especially as smaller facilities are being built in urban centers to expedite delivery times, where real estate costs are significant. Shelf-based Automated Storage and Retrieval Systems

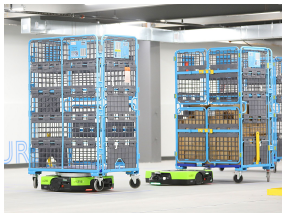


Fig. 1: Amazon's Proteus robots [1] carry uniform loads and are example platforms that can be used for multi-robot ordered storage and retrieval in warehouses.

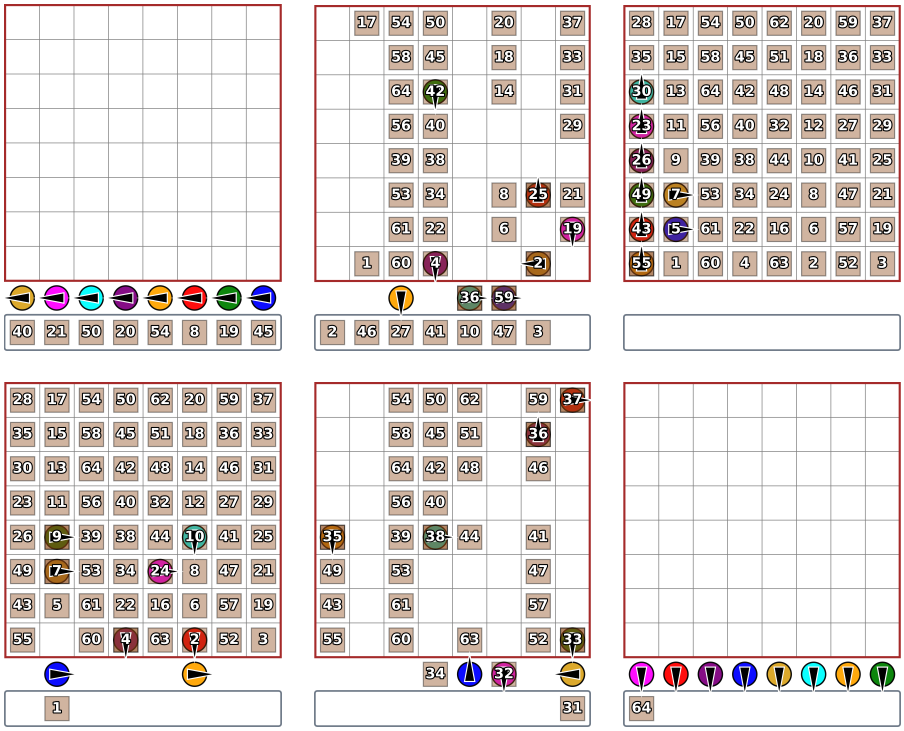


Fig. 2: Snapshots of a multi-robot storage and retrieval problem for a grid-based system with 8 robots. The storage area is depicted as an 8×8 square. The conveyor belt is depicted as a 1×8 rectangle below the storage area, where loads arrive and depart from left to right. Robots are depicted with colored circles with an arrow indicating direction. Loads are depicted with squares labeled per their departure order. Top left: an initially empty grid with an initial set of loads arriving via a conveyor belt. Top right: completion of the storage phase, with the warehouse at full capacity. Bottom left: first load retrieval. Bottom right: the final state where the last load is retrieved and moved to the conveyor belt.

(AS/RS), such as Kiva-style ones [22], use robots like the Amazon Proteus (Fig. 1). While operationally efficient, they depend on dedicated aisles and buffer zones in the warehouse to facilitate robot navigation. This layout sacrifices valuable space that could otherwise be used for storage.

To address these challenges, high-density storage systems have emerged, often modeled as Puzzle-Based Storage (PBS) [7]. This framework features a 2D rectangular grid, where each grid cell stores a uniform sized load. A load can be moved along cardinal directions via a collision-free path. In these environments, the storage grid acts like a dense sliding-tile puzzle, removing internal aisles, where loads are rearranged using a small number of empty cells to create paths for retrieval. Coordinating multiple robots in these dense environments, however, is computationally challenging due to the high risk of deadlocks and limited maneuvering space.

This work builds on a model for grid-based storage that splits operations into two distinct phases: first, storing incoming items up to full capacity, and later, quickly retrieving them for transport. Such operations occur in facilities where goods must be stored before changing transport mode (e.g., from a ship to trucks in a port, or from a semi-truck to delivery vans in last-mile distribution centers). In these setups, it is critical to make the best use of available space and quickly serve outbound transportation vehicles as they arrive.

Prior work, such as `StoRMR` (Storage and Retrieval with Minimum Relocations) [5], has established the geometric feasibility of relocation-free arrangements for such grid-based storage systems. This means it is possible to store loads up to full capacity in a grid-based storage so that loads can be retrieved according to a scheduled departure sequence without any relocation, i.e., a rearrangement of other stored loads, under weak assumptions about the grid's width. This model has been extended to accommodate controlled uncertainty in the departure sequence, as in `R-StoRMR` (Robust `StoRMR`) [6]. These prior works, however, assume a sequential movement model in which only one robot moves a load at a time. Consequently, the execution-time benefits of these dense arrangements in a parallelized, multi-robot setting remain largely unexplored.

This paper bridges the gap between relocation-minimizing storage arrangements and practical, efficient multi-robot execution as highlighted in Fig. 2. It introduces a novel multi-robot formulation for the Ordered Storage and Retrieval problem under maximum storage capacity. Then, this paper describes an online, prioritized Multi-Agent Path Finding (MAPF) algorithm to effectively coordinate multiple robots to execute the corresponding storage and retrieval sequences. Unlike coupled, centralized planners that often time out in congested spaces due to the curse of dimensionality, the approach exploits specific layout properties to achieve efficiency via the proposed decoupled, prioritized approach. At the same time, it also guarantees completeness and prevents deadlocks even at 100% load density, which is a rare case for prioritized MAPF planning methods, e.g., [21].

Contributions: Overall, the primary contributions of this work are:

- It introduces a multi-robot formulation for ordered storage and retrieval at maximum capacity.
- It proposes an asynchronous, online prioritized planning algorithm that enables continuous task execution for relatively large-scale instances. It provides theoretical arguments that the algorithm remains complete by utilizing the geometric properties of relocation-free storage layouts.
- Extensive experiments on grids of size up to 30×30 demonstrate the algorithm's *efficiency and scalability*, which achieves near-linear makespan reduction as the robot count increases. The planner maintains low computational runtime per task, making it suitable for online, real-time operation.
- The evaluation also shows that the cost of handling *departure uncertainty* is negligible; `R-StoRMR` placement layouts provide safety against controlled sequence perturbations without significant penalties to execution speed.

– In terms of *solution quality*, the algorithm exhibits low makespan suboptimality when compared to a theoretically optimal (but non-scalable) centralized, coupled planner.

2 Related Work

Significant progress has been made in Multi-Agent Path Finding (MAPF) and Multi-Agent Warehouse Rearrangement (MAWR). The combination, however, of efficient storage arrangements in Puzzle-Based Storage (PBS) and MAPF techniques has received less attention. In other words, the intersection of full density, strict arrival and departure sequencing, and multi-robot coordination remains a largely unexplored topic.

High-Density Storage and Retrieval. High-density storage systems, often formalized as Puzzle-Based Storage (PBS), were introduced to maximize space utilization by eliminating permanent aisles. Work has established the framework for PBS by modeling the storage grid on the classic "15-puzzle" game [7]. In this model, inventory items are treated as tiles that must navigate a dense grid using a limited number of empty cells, known as "escorts," to reach an input-output point. Building on this foundation, subsequent efforts extended the analytical framework by deriving closed-form expressions for retrieval times in systems with multiple, randomly distributed escorts [10]. While PBS systems often allow freedom in retrieval order [16, 8], this flexibility does not address scenarios requiring strict sequencing.

To address sequencing constraints, the Block Relocation Problem (BRP) models stack-based storage where items must be retrieved in a predefined order. Research has proven that the BRP is NP-hard and developed algorithms to minimize the relocation of blocking items [3]. Under uncertainty, variants such as the Parallel Stack Loading Problem focus on loading phases where the precise retrieval sequence is unknown [2].

Multi-Agent Path Finding. A popular solution for coordinating multiple robots is Conflict-Based Search (CBS) [19], which speeds up standard A* by coupling agents only when conflicts arise. While CBS provides strong theoretical foundations, it often suffers from scalability issues in congested environments. To mitigate this, Priority Inheritance with Backtracking (PIBT) [18] introduces reactive, priority-driven mechanisms. PIBT excels in dense settings by allowing blocked agents to inherit priority from their neighbors, effectively resolving local deadlocks. Previous approaches like Push and Swap [13] guaranteed completeness in dense settings but lacked solution quality. More recently, Large Neighborhood Search (LNS) algorithms [12] have emerged as an effective approach to rapidly repair suboptimal paths in congested grids.

The coordination strategy proposed in this work aligns with the principles of Lifelong Multi-Agent Path Finding [14], which enables continuous task execution by iteratively updating paths. While traditional prioritized solvers rely on environments with sufficient buffer space to resolve conflicts, the proposed

approach adapts these methodologies to the unique constraints of 100% density, a domain theoretically grounded in pebble motion on graphs [9].

Multi-Agent Warehouse Rearrangement. The Block Rearrangement Problem (BRaP) [4] investigates symbolic planning for moving blocks in extremely dense grids, treating loads as active agents to simplify coordination. The setting in this work differs from BRaP by utilizing a two-level height model, a standard feature in Kiva-style AMR systems [22]. In the two-level architecture, autonomous robots (AMRs) navigate beneath storage loads. This introduces unique constraints where pickup and drop-off actions must be explicitly modeled to effectuate load movement, unlike the sliding-tile abstraction used in BRaP.

Recent developments have led to solvers specifically for this two-height model. The Multi-Agent Warehouse Rearrangement (MAWR) problem incorporates an initial layout, movable obstacles, and agents. The DD-MAPD framework for MAWR achieves scalability by decomposing shelf trajectories from robot tasks, though it requires "well-formed" environments with sufficient buffer space to guarantee completeness [11].

The MARPF approach employs an ILP-based solution to actively relocate non-target racks as movable obstacles, but it primarily focuses on path-clearing for specific targets rather than global arrangement [15]. NAT-CBS introduces a makespan-optimal, coupled algorithm for MAWR by integrating a high-level obstacle planner with a low-level anonymous MAPF flow network, yet it remains computationally intensive in high-occupancy settings [20].

Recent work has explored rearrangement-free retrieval in fully packed grids for two-step storage and retrieval setups [5], including under uncertainty for the departure sequence [6]. These efforts motivate the current work but do not address the multi-agent coordination aspects required for fast and effective storage and retrieval in practice.

3 Problem Formulation

3.1 System Setup

Environment and Layout. We consider a rectangular storage area W consisting of R rows and C columns of grid cells. Two special rows lie below W to facilitate storage and retrieval of loads: the **I/O row** at index $R+1$, used as a temporary buffer, and the **conveyor** at index $R+2$, which supplies arriving loads and receives departing loads sequentially. Robots are not permitted on the conveyor. The I/O row and the conveyor are carefully designed to enable lock-free multi-robot operations.

There are $n = R \cdot C$ distinct loads labeled $1, \ldots, n$, and $m \leq C$ robots. Each cell can contain at most one load and at most one robot. Each robot can carry at most one load. An *arrangement* $\mathcal{A}$ of the loads is an injective mapping that specifies a cell in W (i.e., a row and a column) for each load.

Robot Actions and Movement. Each robot is oriented along one of the four cardinal directions. Time is discretized, and at each timestep, a robot can

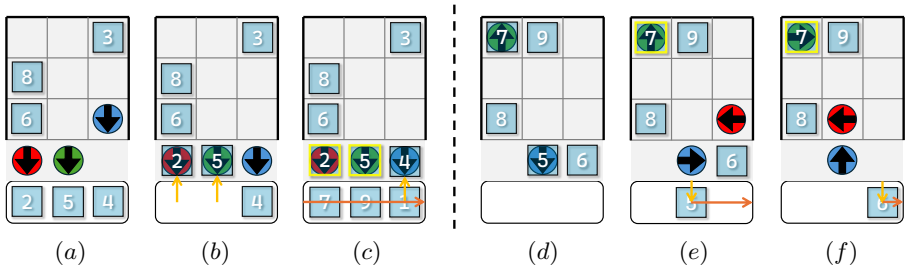


Fig. 3: Robots are represented by circles with arrows indicating direction. Loads are indicated by labeled squares, where the labels correspond to the departure sequence $D = (1, 2, \ldots, 9)$ without loss of generality. Yellow borders indicate a load currently held by a robot. The left sequence shows three frames that highlight conveyor dynamics during the storage phase for the arrival sequence $A = (8, 3, 6, 4, 5, 2, 1, 9, 7)$. Frame (b) shows the conveyor pushing 2 and 5 to the I/O row in one timestep. Frame (c) shows a conveyor pushing 4 to the I/O row, followed by a subsequent batch of $(1, 9, 7)$ moved into the conveyor row. During the retrieval phase, frame (d) shows loads 5 and 6 on the I/O row. In (e), load 5 is first received by the conveyor given the departure order D and instantly departs to the right. Then load 6 follows (f).

perform one of the following actions: **move** (forward one cell in the robot's current orientation), **rotate** (90-degrees CW or CCW), **pick up**, or **drop off** a load in the same cell, and **wait**.

While standard MAPF abstracts agents as points, explicitly modeling rotation captures the reality of physical robots in tight storage grids. At the same time, however, turning actions make the problem more challenging, especially in dense environments.

A load moves only while it is carried by a robot and is stationary once dropped off, except for moving to/from the conveyor (explained below). A two-level height model is adopted in which robots not carrying a load may pass beneath stationary loads without colliding with them.

The following collisions must be avoided: (1) **Positional collisions**, where two entities of the same type (two robots or two loads) occupy the same cell in the same timestep, and (2) **Directional collisions**, where a robot enters a cell at time t that was occupied by another robot at $t - 1$, unless both robots are moving in the same cardinal direction (i.e., "train" movement is permitted, but orthogonal and swapping motions are prohibited).

3.2 Problem Definition

Operational Phases. The system operates in two phases. In the **storage phase**, loads arrive via the conveyor in ordered batches of size C (one load per column) according to the arrival sequence $A = (a_1, \ldots, a_n)$. To decouple the conveyor's motion from the robots' motion, assume that the conveyor operates as an independent unit (see Fig. 3). If the entire conveyor row is empty, the conveyor places the next batch of C loads from left to right on the conveyor

row instantly. This batching structure exposes multiple loads simultaneously, enabling the robots to access them in parallel. The conveyor pushes any load on it onto the I/O row in one timestep for robot pickup. In the **retrieval phase**, loads can be delivered to the I/O row for departure. The conveyor can receive loads from the I/O row in one timestep. The conveyor strictly follows the departure order D. Without loss of generality, D is defined by $D = (1, 2, \ldots, n)$.

A *plan* Π is a sequence of joint actions $\langle \mathbf{a}_t, c_t \rangle$ for $t = 1, \ldots, T$, where $\mathbf{a}_t = \{u_{1,t}, \ldots, u_{m,t}\}$ is the set of actions performed by the robots and c_t denotes the conveyor action. A plan is *collision-free* if its actions avoid motion with collisions as defined above. The *makespan* of a plan is the total number of timesteps T.

Formal Problem Statement. The **storage problem** is defined by the tuple $\langle W, \mathcal{R}, \mathcal{A}, A \rangle$, where $\mathcal{R}$ is the initial robot configuration, $\mathcal{A}$ is the target storage arrangement, and A is the arrival sequence. The **retrieval problem** is defined by $\langle W, \mathcal{R}, \mathcal{A}_{\text{init}}, D \rangle$, where $\mathcal{A}_{\text{init}}$ is the load arrangement after storage at full capacity and D is the departure sequence.

In both cases, the output is a collision-free plan Π that transforms the system to the target configuration. The plan can be generated offline or executed online. The objective is to minimize the makespan T.

4 Preliminaries

The proposed multi-robot approach builds upon the guarantees provided by sequential relocation-free storage arrangements. An arrangement $\mathcal{A}$ is an injective mapping $\{1, \ldots, n\} \to \{1, \ldots, R\} \times \{1, \ldots, C\}$ assigning each load to a unique grid cell [5, 6]. The notation $\mathcal{A}[\ell]$ denotes the cell to which ℓ is mapped.

Definition 1. *An arrangement $\mathcal{A}$ (or respectively D) satisfies an arrival (or departure) sequence if every load in that sequence can be stored (or retrieved) sequentially without moving any other load, i.e., without relocations.*

Geometrically, satisfiability implies the existence of a collision-free path from the I/O row to the target cell (or vice-versa), avoiding specific subsets of loads:

- **Storage:** For an arrival sequence $A = (a_1, \ldots, a_n)$, the path for storing load a_j from the I/O row must be collision-free given all previously placed loads $\{a_1, \ldots, a_{j-1}\}$.
- **Retrieval:** For a departure sequence $D = (d_1, \ldots, d_n)$, the path for retrieving load d_j and moving it to the I/O must be collision-free given all loads scheduled for later departure $\{d_{j+1}, \ldots, d_n\}$.

Figure 4 provides an example of an arrangement that satisfies A and D. Note that loads can be stored to or retrieved from any location of the I/O row. These definitions establish *static feasibility*: they guarantee that valid paths exist if loads are moved one at a time, independent of conveyor dynamics or robot parallelization. Both of these aspects are the focus of the current work.

The following define parameters for departure uncertainty. To account for uncertainty in the departure order, the notion of k-robustness is useful:

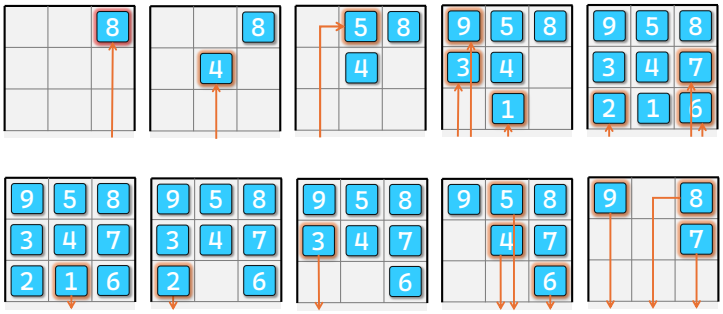


Fig. 4: An arrangement that satisfies $A = (8, 4, 5, 3, 1, 7, 6, 2)$ and $D = (1, 2, 3, 4, 5, 6, 7, 8, 9)$ with no relocations is shown. The storage sequence is shown at the top, and the retrieval sequence is shown at the bottom (both from left to right). Each load is stored or retrieved sequentially in the specified order via a collision-free path. The figure is based on [5, 6].

Definition 2. *Let $\tilde{S}$ be a permutation of sequence $S = (s_1, \ldots, s_n)$. Then, $\tilde{S}$ is a k-bounded perturbation if for every pair (s_i, s_j) where $i < j$ but s_j appears before s_i in $\tilde{S}$ (i.e., an order inversion), the index distance satisfies $j - i \leq k$.*

Definition 3 (k-Robustness). *An arrangement is k-**robust** with respect to S if it satisfies every valid k-bounded perturbation $\tilde{S}$ of S.*

The following sections include the main contribution of this work, which take storage arrangements designed for sequential motion as inputs. The multi-robot planning algorithms are designed to succeed on any input arrangement $\mathcal{A}$ that satisfies A and D. The proposed algorithms leverage these invariants to guarantee completeness (Section 6) while parallelizing operations for maximum throughput.

5 Prioritized Planning Algorithm

The proposed algorithm utilizes an asynchronous prioritized planning strategy where robots are assigned tasks and compute paths only upon becoming idle after the completion of prior tasks. Collision-free coordination is maintained via a global reservation table, which tracks space-time constraints as tuples (p, t, d), which represent the position p, timestep t, and prohibited entry direction d, to explicitly prevent directional following conflicts.

To support the two-level height model, the reservation system tracks obstacles on two distinct layers: *Robot Layer:* Active robots reserve cells along their planned trajectories. Once a robot completes its path, it is treated as a static obstacle at its final cell for all future timesteps until assigned a new task. *Load Layer:* Stored loads are marked as static obstacles. The status of load obstacles is updated given robot paths: a load-obstacle is removed from the table at the

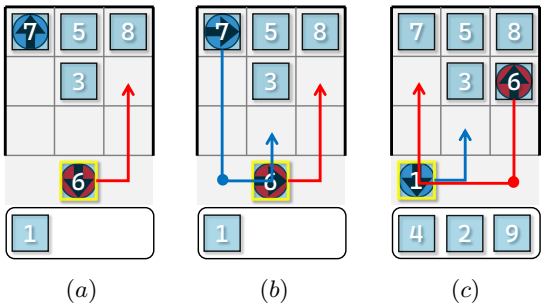


Fig. 5: An example instance of the storage algorithm with $A = (3, 7, 5, 8, 6, 1, 9, 2, 4)$. Loads being carried are highlighted in yellow. (a) The red robot has a reserved path (in red), while the blue robot has just stored 7 and is now idle. (b) The solver plans a path for blue robot, which includes the pickup point marked with a blue circle. (c) A few timesteps later, the red robot has finished its path, while the blue robot has a small segment remaining to drop load 1. The conveyor loads the next batch $(9, 2, 4)$ according to the arrival sequence. The red robot plans a path to pick up the next load 9, while respecting the prioritized blue path.

timestep a robot is planned to execute a *pickup*, and a new load-obstacle appears at the timestep a robot is planned to perform a *drop-off*.

5.1 Storage

Consider an arrival sequence $A = (a_1, a_2, \ldots, a_n)$, and a storage arrangement $\mathcal{A}$ that satisfies A. Assume planning is performed online, where robots claim tasks dynamically and do not possess knowledge of a load's destination until assignment.

Task Assignment. Whenever a robot becomes idle, it is considered for assignment to the next unclaimed load a_k in the arrival sequence. If multiple robots are idle simultaneously, a_k is greedily assigned to the robot closest to the pickup cell.

Pickup and Planning. The pickup position for a_k is located on the I/O row at the column corresponding to its batch index as indicated in A (line 6 of algorithm 1). The planner executes a space-time A* search to generate a time-minimal trajectory from the robot's current position to the pickup cell and subsequent storage location $\mathcal{A}[a_k]$. This search checks against the global reservation table to ensure collision-free movement. If the A* search fails to find a valid path in line 8, the robot waits and replans in the next timestep without claiming a load, allowing another robot to potentially successfully plan a path for the load.

Conveyor Synchronization. The planner implicitly coordinates with the conveyor. If a valid path allows the robot to arrive at the pickup cell at time t, the system reserves the I/O slot at time $t-1$ for the conveyor to push the load from the conveyor row. If the I/O slot is blocked at $t-1$ (e.g., by another robot), the A* search treats t as invalid for pickup and searches for a later arrival time. See Fig. 5 for a storage sequence example.

Algorithm 1: Multi-Robot Prioritized Planning for Storage

Input: Robots $\mathcal{R}$, Arrival order A, Planned arrangement $\mathcal{A}$
Output: Storage execution plan Π

```
1  while A has unclaimed loads or active robots exist do
2      R_idle ← {r ∈ R | r.queue is empty}
3      while R_idle is not empty and A has unclaimed loads do
4          ℓ ← next unclaimed load in A
5          r ← arg min_{r'∈R_idle} dist(r', pickupSpot(ℓ))
6          p_start ← pickupSpot(ℓ)
7          p_goal ← A[ℓ]
8          π ← planPath(r, p_start, p_goal)
9          if π is success then
10             Mark ℓ as claimed
11             r.queuePath(π)
12         R_idle ← R_idle \ {r}
13     Execute next step for all robots
```

5.2 Retrieval

The objective is to maintain the same approach as for storage but mirrored. Each robot handles the retrieval of a load and then goes back to the storage area to wait (idle) under a stored load, to avoid interfering with subsequent retrievals. Applying this approach for retrieval introduces a new challenge, namely the choice of which load to have a robot wait under after it drops a load off at the I/O row. This choice is easy for storage, since a robot can stay under the load it just stored. For retrieval, robots should be positioned under loads according to the departure sequence, but there is no guarantee that a load that is yet to depart can be accessed (e.g., the 5th load to depart can be blocked from all sides by previously departing loads claimed by occupying robots). Furthermore, if knowledge of the departure sequence is limited, then the approach is not clearly defined. Therefore, the approach allows the possibility of sending a robot back under an arbitrary load.

Algorithm 2 details the retrieval algorithm. The input is a departure sequence, $D = (1, 2, \ldots, n)$ and an initial storage arrangement $\mathcal{A}$, which satisfies D. Robots avoid task conflict through a prioritized claiming system. A robot irrevocably claims the next load in the departure sequence only upon the successful calculation of a valid path. This ensures that it can transition to the next task immediately after its current delivery and wait safely under its next claimed load. At the start, no robots have claimed loads, so each robot finds a path to the next unclaimed departing load in D, namely load j (line 21). If a robot has claimed a load and finished its path to the load's position, $\mathcal{A}[j]$, it will plan a shortest timestep path to any drop-off location on the I/O row and then to its next assigned load, which is the next unassigned departing load in D (line 8). Each path will be reserved in the form of obstacles for future path planning, as in the storage algorithm. See Fig. 6 for an example sequence.

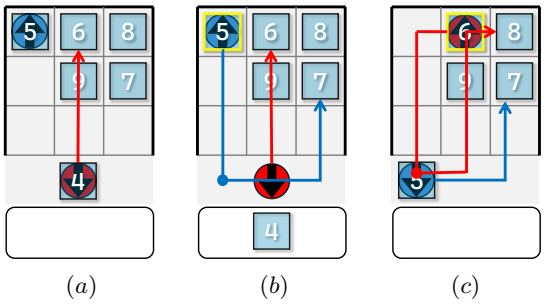


Fig. 6: An example instance of the retrieval algorithm with $D = (1, 2, 3, 4, 5, 6, 7, 8, 9)$. Loads being carried are highlighted in yellow. (a) The red robot has an already reserved path in red, while the blue robot has just reached 5 and is now idle. (b) The solver plans a path for the blue robot, which includes the drop-off point marked with the blue circle and the next unclaimed load in D, load 7. The conveyor can also receive the next load in D, which is load 4 from the red robot. (c) A few timesteps later, the red robot has finished its path, while the blue robot has a segment remaining. The red robot plans a path to drop off the current load, 6, and then move to the next unclaimed load, 8. The leftmost drop-off spot is reserved until the blue robot has finished its drop-off.

If planning fails in line 8, the robot executes one of two fallback options. First, the robot attempts to plan a path to drop the load on the I/O row then move under the closest accessible unclaimed load (line 13). This prevents subsequent retrieval paths from being blocked (see Section 6). Alternatively, if there is no available load to wait under, the robot plans a path to drop the load on the I/O row and then moves to an empty cell in the furthest back row to avoid obstructing the paths of other robots (line 18). If planning still fails, the robot waits and replans in the next timestep. If a robot executes a fallback as described above due to a planning failure at line 8, the robot completes its delivery without claiming a subsequent load. Consequently, upon finishing its path, the robot has no claimed load and re-enters the planning phase at line 21 to claim and move toward the next available load.

Departure sequencing. Since the storage $\mathcal{A}$ is guaranteed to satisfy D but not necessarily an arbitrary sequence, the approach enforces the departure order by ensuring a robot plans a path only for load j toward the I/O row when the previous load $j-1$ in the departure sequence has a queued path to the I/O row (line 7 of algorithm 2). Then, each retrieval path reserves its drop-off cell on the I/O row until the conveyor receives the load. As the conveyor accepts loads strictly in departure order, this reservation prevents out-of-sequence loads from prematurely occupying the I/O row and blocking access for the current target load. Section 6 uses this strict departure sequence to provide completeness arguments. To enforce this order for idle robots, line 2 processes robots in a sorted manner. Robots with claimed loads take precedence, ordered by the departure index of their load. Robots without a load are then ordered by their distance to the next unclaimed load, following the same logic used for storage.

Algorithm 2: Multi-Robot Prioritized Planning For Retrieval

Input: Robots $\mathcal{R}$, Departure order D, Storage arrangement $\mathcal{A}$
Output: Retrieval execution plan Π

```
while D has unclaimed loads or active robots exist do
    for r ∈ R where r.queue is empty do
        ℓ ← next unclaimed load in D
        p ← A[ℓ]
        if r has claimed a load then
            if ℓ − 1 has not queued a path to I/O row then
                continue
            π ← PlanPath(r, I/O row, p)
            if π is success then
                Claim ℓ
                r.queuePath(π)
            else
                foreach next closest unclaimed load ℓ′ ∈ A do
                    p′ ← A[ℓ′]
                    π ← PlanPath(r, I/O row, p′)
                    if π is success then
                        r.queuePath(π)
                        continue
                π ← PlanPath(r, I/O row, back row)
        else
            π ← PlanPath(r, p)
            if π is success then
                Claim ℓ
                r.queuePath(π)
    Execute next step for all robots
```

6 Properties

This section establishes the completeness of the multi-robot planning algorithms.

Theorem 1. *The storage algorithm is complete.*

Proof. Let $A = (a_1, a_2, \ldots, a_n)$ be the arrival sequence and $\mathcal{A}$ be a storage arrangement that satisfies A. Assume that the robots are all initially positioned on the front row of W (the row above the I/O row). This allows each robot to initially have an unobstructed path to the I/O row. The proof is by induction on k that load a_k can be stored.

For the base case, consider the first load a_1. There exists a path π from the I/O row to the target $\mathcal{A}[a_1]$, which passes through at most one cell with a robot r. If such a robot r exists, then path planning for r succeeds: r can move via π to pick up the load and then move it to $\mathcal{A}[a_1]$. If π does not pass through an occupied cell, path planning succeeds for any robot.

Assume loads $a_1, \ldots, a_{k-1}$ are already stored. Consider the arrival of load a_k. Suppose there is a time step at which all the robots are idle. If planning succeeds for a_k before that, then the point is shown. Otherwise, the following arguments show that it will also be true. By the definition of $\mathcal{A}$, there exists a path π from the I/O row to $\mathcal{A}[a_k]$ that avoids all previously stored loads. Robots are positioned either under stored loads or at their initial cells (for robots that have not yet picked up a load). Therefore, the only obstacles on π may be robots still at their initial cells. If such a robot-obstacle r exists, assume that it is the only one by the choice of the initial robot configuration. Therefore, path planning will succeed for r as in the base case. If there are no robot-obstacles on π, then there exists a robot that can reach the I/O row to pick up and store a_k via π.

For retrieval, we follow a similar approach to prove completeness.

Theorem 2. *The retrieval algorithm is complete.*

Proof. The proof is by induction on k that load k can be retrieved. For the base case, consider load 1, the first to depart. First, a robot must claim load 1. Indeed, as idle robots are only obstructed by other robots, some robot r can claim load 1 and proceed to $\mathcal{A}[1]$. Next, we need to show that r can drop load 1 off by successfully planning two path segments: (i) from $\mathcal{A}[1]$ to the I/O row and (ii) then back to W. By the definition of the arrangement $\mathcal{A}$, there exists a path from $\mathcal{A}[1]$ to the I/O row (more specifically, $\mathcal{A}[1]$ must be adjacent to the I/O row). Path planning for r must also succeed for the second path segment: there is a load under which r can wait, which can be either the next unclaimed load or an arbitrary one. Other robots cannot interfere with the two path segments, as no drop-off paths are planned for them until a drop-off for load 1 is queued.

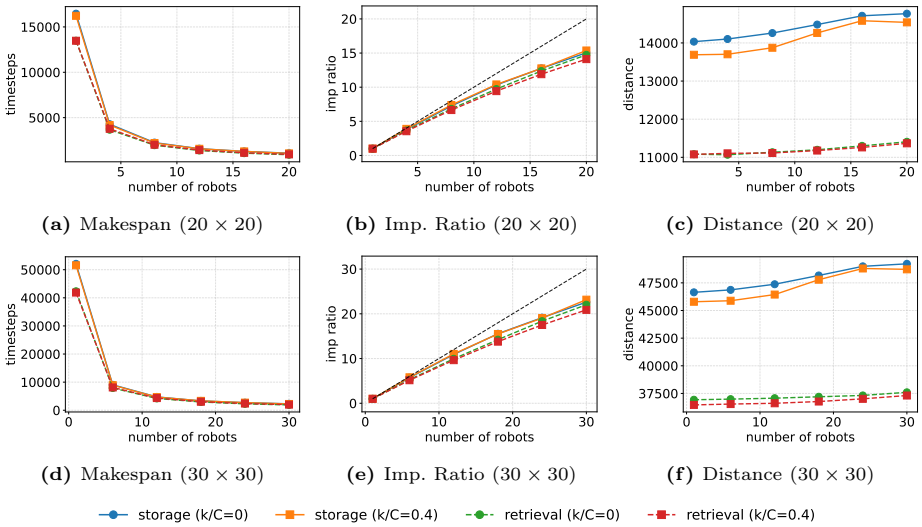
Assume loads $1, \ldots, k-1$ have been retrieved and consider load k. As in the storage case, let us assume that all the robots become idle in W before k is retrieved, as otherwise there is nothing to show. There exists a load-free path π from $\mathcal{A}[k]$ to the I/O row as $\mathcal{A}$ satisfies D. First, some robot r can claim load k and proceed to $\mathcal{A}[k]$, which is accessible to an idle robot due to the existence of π. We now assume that every idle robot is under a load (idle robots not under a load will be handled below), which means that r can drop load k off via π.

From here, two cases remain for the return path to W: in the first, r returns to wait under some load. Otherwise, there is no accessible load for r to go under. Assume the second case, as otherwise we are done. This case is presented in the full proof in the appendix.

7 Experiments

This section evaluates the performance of the proposed prioritized planner integrated with the arrangement choices of the Storage and Retrieval with Minimum Relocations algorithms [5, 6].

The experiments are designed to validate three primary claims. *Efficiency:* The algorithm effectively utilizes multiple robots to reduce the total makespan



(a) Makespan (20×20) **(b)** Imp. Ratio (20×20) **(c)** Distance (20×20)

(d) Makespan (30×30) **(e)** Imp. Ratio (30×30) **(f)** Distance (30×30)

Fig. 7: Performance comparison. Top row: 20×20 grids; Bottom row: 30×30 grids. Columns (left to right): Total Makespan, Makespan Improvement Ratio, and Total Distance traveled. Each data point averages 50 experimental trials.

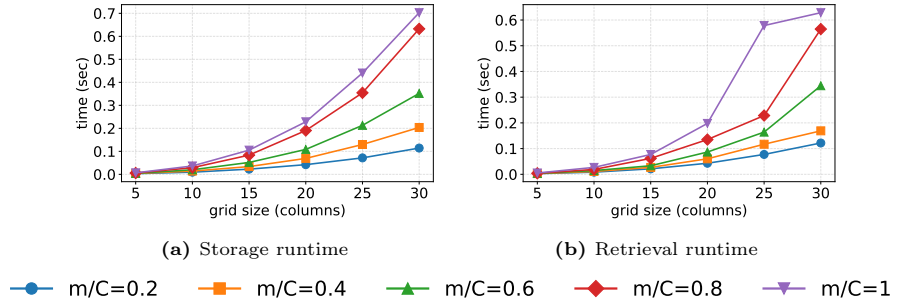
of storage and retrieval tasks, achieving near-linear speedup in the number of robots. *Scalability:* The planner maintains low runtime per task, making it suitable for online, real-time operation. *Solution quality:* The solutions exhibit low suboptimality when compared to a theoretically optimal (but computationally expensive) coupled planner.

The experiments are executed on a variety of relocation-free storage arrangements and for 1 to C robots. To generate relocation-free storage arrangements, the experiments use the `StoRMR` solver ($k = 0$) [5] and the `R-StoRMR` solver [6] for $k = 0.4 \cdot C$ for a k-robust arrangement.

Each instance randomizes an arrival sequence A and a departure sequence D. For experiments testing robust arrangements given k, we generate a k-perturbed sequence $\tilde{D}$. Experiments were run on Ubuntu 22.04.5 LTS with an Intel Xeon Gold 5220 CPU (2.20 GHz). The appendix includes a comprehensive listing of experimental results for additional configurations (grid sizes, k values).

To quantify the performance gain provided by multiple robots relative to the system scale, consider the *makespan improvement ratio* (S_m): the makespan improvement ratio is defined as $S_m = \frac{T_1}{T_m}$, where for a given grid size $C \times C$ and load count $n = C \cdot C$, let T_1 be the optimal makespan achieved by a single robot and T_m be the makespan achieved by m robots.

Figure 7 demonstrates that the prioritized planning approach achieves near-linear scalability relative to the number of robots (m). Figures 7(a)(b) provide the absolute makespan and the improvement ratio. They demonstrate the significant decrease in makespan with the introduction of additional robots, which directly relates to the throughput efficiency of the storage system. S_m adheres



(a) Storage runtime **(b)** Retrieval runtime

Fig. 8: Mean runtime per load for storage (left) and retrieval (right) as the square grid size increases (the x-axis corresponds to the number of columns C). Each plotted line corresponds to a different number of robots m as a proportion of C.

Table 1: Makespan suboptimality ratio for 2 robots. (20 trials per entry)

(a) Storage

Grid	3×3	4×4	5×5	6×6
Ratio	1.091	1.162	1.210	1.176

(b) Retrieval

Grid	3×3	4×4	5×5	6×6
Ratio	1.086	1.108	1.129	1.111

closely to the ideal linear benchmark up to approximately $S_m = 15$ for a 20×20 grid with 20 robots. This suggests that the system can support high density of active robots before interference significantly impacts efficiency.

Robustness. A key contribution of this work is the integration of `R-StoRMR` arrangements. These arrangements allow completeness under uncertain departure sequences that are revealed sequentially during retrieval. Notably, Fig. 7 reveals that the execution penalty for using robust arrangements ($k = 0.4C$) compared to standard arrangements ($k = 0$) is negligible (the plots for the robust and the non-robust storage overlap). This indicates that the safety introduced by `R-StoRMR` to handle retrieval uncertainty do not hinder robot motion.

Coordination overhead. Figure 7(c)(f) shows the average total distance traveled by robots. While the total distance naturally increases with m due to conflict avoidance maneuvers, the slope is remarkably shallow (less than 5% increase for $m = 20$ compared to the single robot case). This indicates that there is only a small sacrifice due to multi-robot coordination and collision avoidance.

Runtime. Figure 8 measures the runtime per load, which is the average planning time it takes to successfully calculate a path to store or retrieve a load as the grid size increases and for a different number of robots. There is a reasonable increase in planning runtime as the size of the grid increases as well as the number of robots increases. Nevertheless, the runtime always remains in the sub-second territory for the experiments performed, which can easily be accommodated by a real-world system in this space and allow the execution of these plans without any robot waiting for path planning.

Makespan optimality. To evaluate the solution quality of the prioritized planning algorithm, this section compares against an optimal coupled, centralized planner based on A* search in the joint configuration space of two robots, which is executed for both robots every time a robot is assigned a new storage or retrieval task. To ensure computationally feasible runtimes for the optimal solver, we decompose the problem into batches of size C. That is, we run the storage (or retrieval respectively) with the goal of storing (or retrieving) a segment of C loads in A or D. Then, the robots are placed back to their spots at the end of the batch and the solver is rerun for the next batch. Each trial sums up the makespan of each batch in A or D. Given the computational explosion of the A* search, the method was only executed for this small multi-robot instances due to memory constraints. See the appendix for details about this solver.

Table 1 shows that for small instances where the optimal coupled solver is tractable, the proposed algorithm exhibits a suboptimality ratio ranging from 1.09 to 1.21. A portion of this suboptimality comes from the optimal solver taking advantage of the conveyor model, which allows loads to be stored and retrieved slightly out of order as long as no blockages occur. The prioritized approach uses the fixed arrival and departure sequences for completeness guarantees.

8 Conclusion

This paper presents a comprehensive algorithmic framework for multi-robot ordered storage and retrieval in maximum-capacity environments. By bridging the gap between the zero-relocation guarantees of `StoRMR` and `R-StoRMR` arrangements and online prioritized planning, it demonstrates that high-density warehouses can achieve a scalable high-throughput two-step, storage and retrieval, process even at maximum capacity.

The theoretical analysis confirms that the specific invariants of these storage arrangements provide sufficient clearance to guarantee completeness for sequential prioritized planning, avoiding the deadlocks common in prioritized MAPF. Experimentally, we show that this approach achieves near-linear makespan improvement up to $m = C$. A critical finding is the negligible execution cost of robustness: the `R-StoRMR` layouts, which protect against departure sequence uncertainty, can be executed just as efficiently as the `StoRMR` layouts. While the approach incurs a suboptimality gap compared to coupled search, it maintains scalability for large instances (30×30 grids with 30 robots).

Future work could focus on narrowing the optimality gap by investigating other MAPF techniques, such as PIBT and LaCAM [18, 17]. Additionally, investigations into `StoRMR` and `R-StoRMR` could reveal better arrangements that tailor to multiple robots. If departure uncertainty becomes too high, target loads may become blocked. This can motivate the development of a multi-robot blocked retrieval problem. Beyond algorithmic variations, this work can be extended to different environments, such as those that allow multiple-side access as well as static obstacles within the storage space.

References

1. Amazon's Proteus Robot Systems. https://robotsguide.com/robots/proteus (2022)
2. Boge, S., Knust, S.: The parallel stack loading problem minimizing the number of reshuffles in the retrieval stage. Eur. J. Oper. Res. **280**(3), 940–952 (2020)
3. Caserta, M., Schwarze, S., Voß, S.: Container rehandling at maritime container terminals: A literature update. Handbook of terminal planning pp. 343–382 (2020)
4. Fu, B., Chen, Z., Chandan, R., Barbosa, A., Caldara, M., Durham, J., Pecora, F.: Symbolic planning and multi-agent path finding in extremely dense environments with unassigned agents. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). vol. 40, p. 29421–29431 (Mar 2026). https://doi.org/10.1609/aaai.v40i35.40183
5. Geft, T., Bekris, K.E., Yu, J.: Fully packed and ready to go: High-density, rearrangement-free, grid-based storage and retrieval. CoRR **abs/2505.22497** (2025)
6. Geft, T., Zhang, W., Yu, J., Bekris, K.: Robust out-of-order retrieval for grid-based storage at maximum capacity. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 40, pp. 36245–36252 (Mar 2026). https://doi.org/10.1609/aaai.v40i43.40943
7. Gue, K.R., Kim, B.S.: Puzzle-based storage systems. Naval Research Logistics (NRL) **54**(5), 556–567 (2007)
8. He, J., Liu, X., Duan, Q., Chan, W.K.V., Qi, M.: Reinforcement learning for multi-item retrieval in the puzzle-based storage system. European Journal of Operational Research **305**(2), 820–837 (2023). https://doi.org/10.1016/j.ejor.2022.03.042
9. Kornhauser, D., Miller, G., Spirakis, P.: Coordinating pebble motion on graphs, the diameter of permutation groups. In: 25th Annual Symposium on Foundations of Computer Science (FOCS). pp. 241–250 (1984). https://doi.org/10.1109/SFCS.1984.715934
10. Kota, V.R., Taylor, D., Gue, K.R.: Retrieval time performance in puzzle-based storage systems. Journal of Manufacturing Technology Management **26**(4), 582–602 (2015)
11. Li, B., Ma, H.: Double-deck multi-agent pickup and delivery: Multi-robot rearrangement in large-scale warehouses. IEEE Robotics and Automation Letters **8**(6), 3701–3708 (2023). https://doi.org/10.1109/LRA.2023.3272272
12. Li, J., Chen, Z., Harabor, D., Stuckey, P.J., Koenig, S.: Mapf-lns2: Fast repairing for multi-agent path finding via large neighborhood search. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 9379–9387 (2022). https://doi.org/10.1609/aaai.v36i9.21168
13. Luna, R., Bekris, K.E.: Efficient and complete centralized multi-robot path planning. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 3268–3275 (2011). https://doi.org/10.1109/IROS.2011.6095085
14. Ma, H., Li, J., Kumar, T.K.S., Koenig, S.: Lifelong multi-agent path finding for online pickup and delivery tasks. In: AAMAS. pp. 837–845. ACM (2017)
15. Makino, H., Ohama, Y., Ito, S.: Marpf: Multi-agent and multi-rack path finding. pp. 8435–8441 (10 2024). https://doi.org/10.1109/IROS58592.2024.10803053
16. Mirzaei, M., Koster, R.B.M.D., Zaerpour, N.: Modelling load retrievals in puzzle-based storage systems. International Journal of Production Research **55**(22), 6423–6435 (2017). https://doi.org/10.1080/00207543.2017.1304660

17. Okumura, K.: Improving lacam for scalable eventually optimal multi-agent pathfinding. In: Elkind, E. (ed.) Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23. pp. 243–251. International Joint Conferences on Artificial Intelligence Organization (8 2023). https://doi.org/10.24963/ijcai.2023/28
18. Okumura, K., Machida, M., Défago, X., Tamura, Y.: Priority inheritance with backtracking for iterative multi-agent path finding. Artificial Intelligence p. 103752 (2022). https://doi.org/10.1016/j.artint.2022.103752
19. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence **219**, 40–66 (2015). https://doi.org/10.1016/j.artint.2014.11.006
20. Sherma, Y., Weiss, E., Salzman, O.: From agent centric to obstacle centric planning: A makespan-optimal algorithm for the multi-agent warehouse rearrangement problem. In: Proceedings of the International Symposium on Combinatorial Search. vol. 18, pp. 136–144 (2025). https://doi.org/10.1609/socs.v18i1.35985
21. Silver, D.: Cooperative pathfinding. In: Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment. vol. 1, pp. 117–122 (2005)
22. Wurman, P.R., D'Andrea, R., Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI magazine **29**(1), 9–9 (2008)