Fully Packed and Ready to Go: High-Density, Rearrangement-Free, Grid-Based Storage and Retrieval

Tzvika Geft, Kostas Bekris, Jingjin Yu

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

Abstract

We consider an ordered storage and retrieval problem: a set of uniform-sized, labeled loads (e.g., containers, pallets, or totes) must be placed in a 2D grid storage area as they arrive sequentially and then retrieved in a different sequence. Each load occupies a grid cell and may be moved by a robot or manipulator along the cardinal directions. Such storage systems arise in logistics, industrial, and transportation domains, where space utilization and retrieval time are critical metrics. To maximize space utilization, loads must be densely packed with some loads blocking access to others, which raises a key question: How should one store the loads to minimize costly rearrangements, i.e., the number of relocated loads? We identify conditions, alongside efficient algorithms, for achieving either zero or near-optimal rearrangements under different knowledge assumptions. While the online case (i.e., no prior knowledge of the storage and retrieval sequences) induces a trade-off between density and rearrangement, we show that even partial prior knowledge essentially eliminates the trade-off. When the sequences are fully known, we further provide an intriguing characterization: rearrangement can always be eliminated if the grid's open side (used to access the storage area) is at least 3 cells wide, even for storage at full capacity.

Introduction

The past two decades witnessed the dramatic rise of robotics and automation technologies for transporting uniform-sized *loads* or *items* (e.g., containers, pallets, totes, etc.) in logistics applications. Notable examples range from thousands of mobile robots roaming in warehouses helping fulfill orders (Wurman, D'Andrea, and Mountz 2008) to automated cranes and AGVs transporting containers at shipping yards (Bela 2024). Operations at logistics hubs typically occur in two distinct phases: first, the storage of incoming items, and later, their retrieval for further transport. Such operations are common in facilities in which goods have to change transport mode or vehicle, with only a limited storage time in between. In addition to the previous examples, this scenario also arises in cross-docking (Yu and Egbelu 2008), where goods delivered by incoming trucks are quickly sorted and reorganized for transport on outbound trucks. The scenario can also arise as a subtask in Automated Storage and Retrieval Systems (AS/RS) (Roodbergen and Vis 2009; Yalcin 2017), where goods must be resequenced as they arrive from (longer-term) storage for robotic palletization or further transport (Gue, Uludag, and Furmans 2012). Space (footprint-wise) and time-efficient algorithms for routing loads are required as the alternative options involve coupling system components (thereby slowing down delivery) or a lot of "buffer" space (Gue, Uludag, and Furmans 2012).

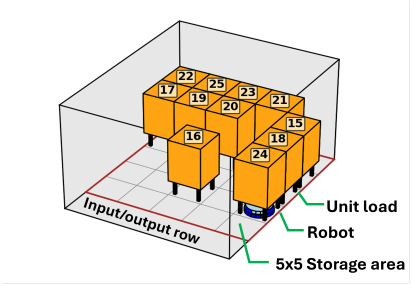

Figure 1: Illustration of a grid-based storage area where a robot can go beneath a load and can move it along the four grid directions.

A central challenge in the design and operation of the aforementioned environments is determining how to temporarily store loads to minimize time and energy costs. A fundamental trade-off arises between storage *density*, i.e., the percentage of space used for storage (as opposed to space for access, such as aisles), and rearrangement costs of storage and retrieval, which increase in high-density storage due to limited accessibility to loads. Each relocation of loads, requiring time-consuming pick-up and drop-off operations, incurs additional costs. Adding to the challenge, information on the arrival and departure time/order of loads can be limited or even unknown (van Brink and van der Zwaan 2014; Geft et al. 2026).

This work investigates the coupling between storage density and rearrangement efforts by asking the following question: How far can we push to maximize storage space utilization while simultaneously minimizing load rearrangement? We focus on automated grid-based storage systems, akin to Puzzle-Based Storage (PBS) (Gue and Kim 2007), which are common as loads are typically uniform-sized. The setup, mirroring practical settings (Wurman, D'Andrea, and Mountz 2008; Guo and Yu 2023), corresponds to a 2D grid-based

storage where each grid cell can store a load and a single robot is tasked to transfer the loads.

We consider the following setup (a formal problem statement follows): the storage area W is a rectangular grid with r rows and c columns, accessible only via the bottom side; see Fig. 2. W stores n labeled (i.e., distinguishable) loads, each occupying one grid cell. We assume that the loads are moved sequentially, i.e., one load at a time by a single robot, with the following natural *actions*: *storage* of an arriving load, *retrieval* of a departing load, or *relocation* (*rearrangement*) of a load from one cell to another, always via a collision-free path. Loads arrive sequentially in the order $A = (a_1, \ldots, a_n)$, where each load needs to be stored in W before the next arrival. Similarly, loads must be later retrieved via another sequence $D = (d_1, \ldots, d_n)$. The objective is to minimize the total number of actions by avoiding rearranging loads. In the best case, $2n$ actions (a storage action and a retrieval action for each load) are necessary. We call such solutions *rearrangement-free*. A problem/solution is *offline* when A and D are known in advance. Otherwise, the problem is *online*, where two versions are examined. The first assumes a *lookahead* ℓ, where only the ℓ next arriving loads are known, along with their positions in the departure sequence D. In the second, *fully online* setting, no knowledge of A and D is assumed, other than the number of loads n.

Contributions. This work introduces a novel ordered storage and retrieval problem on grid-based storage systems with the goal of minimizing relocations. We investigate the extent to which the problem can be solved without relocations under varying assumptions on prior knowledge of the arrival and departure sequences. The key insight of this work is that the density-rearrangement tradeoff, which appears under complete uncertainty, can be essentially eliminated with some prior information on these sequences. The main results are as follows:

- **Offline setting:** Relocation may be required when the number of columns is $c \leq 2$. For any $c \geq 3$, however, it is *always* possible to avoid relocations, for any number of loads, even at full capacity.
- For the **online with lookahead** setting, strong guarantees remain possible, even with the lowest lookahead of 1:
  - If $n \leq r(c-1)+1$, i.e., when it is possible to keep a single column nearly empty, there exists a rearrangement-free solution.
  - When $c \leq r$, a solution with at most $r-1$ relocations exists (for any density), yielding a solution within $9/8$ of the optimal number of actions.
- **Fully online setting:** Without prior information, density must be sacrificed to limit rearrangements. We characterize the tradeoff by providing the maximum achievable density for a given bound on the number of actions allowed per load. In particular, guaranteeing no relocations requires a natural aisle-based layout, with maximum density $2/3$ (see Figure 7, left).

All positive results are accompanied by algorithms that run in (near) linear time. Besides the strong guarantees on minimizing relocations, our solutions result in desirable paths for accessing loads. More specifically, in our rearrangement-free solutions, each path used for storage and retrieval lies in one column with a possible additional lateral segment to a cell adjacent to that column, i.e., each path is *distance-optimal* up to an additive factor of 1. We refer to these paths as **column-adjacent**. Beyond speeding up access by limiting distance and turns, such paths enable the use of a straight-access manipulator with only a limited lateral reach from outside the storage area, e.g., (Huang et al. 2022).

Related Work

We distinguish prior work along two main axes. The first is whether storage/retrieval is *ordered*, with sequence(s) specified in advance, or *online*, where requests arrive with little prior information. The second is the motion model: *Grid-based* systems, which include this work, allow loads to move along both axes of the grid. In *stack-based* systems, often motivated by shipyards, loads are placed in vertical stacks with Last-In-First-Out (LIFO) access, so that a load can only be retrieved after removing those above it. Under this dichotomy, ordered storage and retrieval have been studied almost exclusively in stack-based systems, whereas for grid-based systems–our focus–they have received little attention.

Grid-based storage. Gue (2006) introduces an algorithm to design the layout of a rectangular grid-based warehouse given a *depth* bound–the number of objects that may be in the way of another object. Gue and Kim (2007) presents the "Puzzle-Based Storage" (PBS) concept where items are stored on a high-density 2D grid, potentially with only one empty cell called an *escort*. An item can only be moved to a neighboring escort cell, i.e., retrieval is facilitated by repeatedly moving escorts. As opposed to this work, PBS focuses on the retrieval of one or a few loads at a time, where other loads are movable obstacles, and a storage phase is generally not considered. When multiple loads are to be retrieved, their retrieval is not ordered, i.e., the order is freely chosen (Bukchin and Raviv 2022).

Objectives used in PBS include retrieval time or distance traveled by items (Bukchin and Raviv 2022). Gue and Kim (2007) empirically compare average retrieval distances of dense PBS warehouses to traditional aisle-based ones. Optimal retrieval policies for the sequential case have been derived for one (Gue and Kim 2007) and two escorts (Kota, Taylor, and Gue 2015). Various extensions to the PBS concept exist, using sequential or parallel motions to move items to/from designated input/output ports; see (Bukchin and Raviv 2022) and references within.

Ordered retrieval appears in conveyor-based systems (Gue, Uludag, and Furmans 2012) where decentralized sorting policies receive randomly arriving loads and output them in a specified sequence. Such systems differ fundamentally from our model, as they rely on frequent relocations to clear paths via extensive mechanical infrastructure and dedicate grid rows/columns to motion rather than storage. All these aspects contrast with our focus on rearrangement minimization guarantees at (or near) maximum density, which are also applicable in simple setups.

Stack-based storage. Ordered retrieval has been addressed primarily in stack-based systems (van Brink and

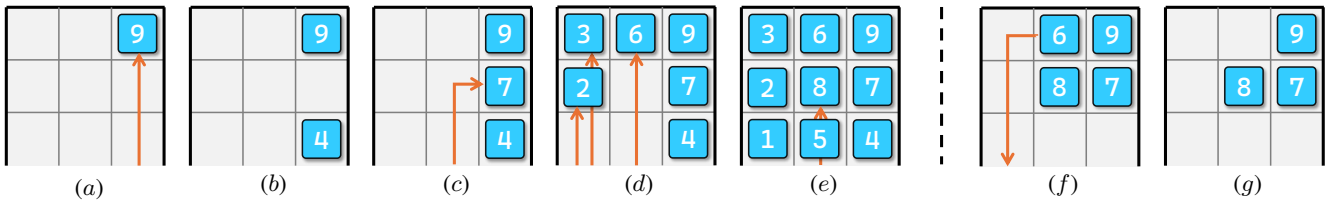

Figure 2: A relocation-free solution for an arrival sequence $A = (9, 4, 7, 3, 6, 2, 1, 8, 5)$ and departure sequence $D = (1, 2, \ldots, 9)$ for a (3×3) grid accessible from the bottom. (a) The first arriving (load) 9 can be directly stored at the top using a (straight) upward path. (b) Next, 4 arrives and can be stored in front of 9, leaving the space in between. (c) 7 is stored using an upward path with a single turn, i.e., a column-adjacent path. (d) 3, 6, 2, can be stored as shown using upward paths. (e) 1, 8, 5 can be stored similarly. At this stage, all loads have arrived. (f) 1, 2, 3, 4, and 5 can be retrieved sequentially directly using downward paths. Then, 6 can be retrieved using a downward path with a turn, another column-adjacent path. (g) 7, 8, and 9 can be retrieved using downward paths.

van der Zwaan 2014; Zehendner, Feillet, and Jaillet 2017; Hakan Akyüz and Lee 2014), in which the LIFO constraint substantially alters the problem structure: even deciding whether relocations can be avoided under given arrival and departure sequences is NP-hard (Boge and Knust 2020). A similar NP-hardness has been established for train-yards, in which tracks also impose LIFO access (Hanou, de Weerdt, and Mulderij 2023). The classic Block Relocation or Premarshalling problems (Lersteau and Shen 2022; Caserta, Schwarze, and Voß 2020) focus on minimizing relocations during or in preparation for retrieval after storage is complete; these too remain NP-hard. Due to the intractability, in all of these settings it is inherently impossible to provide the desirable rearrangement-free solution guarantees that are the focus of our work.

Mechanical search asks to retrieve objects from cluttered environments like shelves, where objects may occlude one another so that their positions may be unknown (Dogar et al. 2014; Huang et al. 2022, 2021). This raises the challenge of optimizing the initial object arrangement so as to minimize retrieval efforts (Chen, Huang, and Goldberg 2024).

Problem Definition, Notation, Terminology

Consider a rectangular $r \times c$ grid storage space W with r rows and c columns. Fix the orientation of W so that its bottom row, also called *front* row, is the open side of W through which loads are stored and retrieved by a single robot. Denote the columns by $C_1, \ldots, C_c$ in a left to right order. The loads have distinct labels $1, \ldots, n$, with $n \leq rc$. The *density* of a storage space having n loads is $n/(rc)$. An *arrangement* $\mathcal{A}$ of the loads is an injective mapping that specifies a cell in W, i.e., a (row, column) pair, for each load. Two loads are *adjacent* in a given arrangement if they are located in horizontally or vertically adjacent grid cells.

Loads are moved one at a time by a robot, via a path of empty cells along the four cardinal directions (up, down, left, or right). Specifically, the following *actions* are considered:

- Storage: The robot can *store* a load in an empty cell v in W via a path from any cell in the front row to v.
- Retrieval: The robot can *retrieve* a load from W via a path from its current cell to any cell in the front row.
- Relocation: The robot can *relocate* a load within W to an empty cell v via a path from its current cell to v. The empty cell v and the original cell of load must be accessible from the front row.

Loads must be all stored and then retrieved according to the prescribed sequences. The departure sequence, i.e., the order in which loads are to be retrieved, is fixed to be $D = (1, \ldots, n)$ without loss of generality, as loads can be relabeled. Denote the arrival sequence, i.e., the order in which loads must be stored, by $A = (a_1, \ldots, a_n)$. The term *access paths* denotes the paths used to store/retrieve a load.

Storage and Retrieval with Minimum Relocations (`StoRMR`): Given a storage area W, and arrival and departure sequences A and D, respectively, find a minimum-length sequence of actions for a single robot that stores all loads per the sequence A and then retrieves the loads per the sequence D.

In addition to minimizing relocation actions, this work also examines the *total distance* traveled by the loads carried by the robot, which is the sum of the lengths of the paths corresponding to the action sequence.

Online variants. In practice, it may not be possible to know the full arrival sequence A in advance. Therefore the following two online settings are considered:

- **Online with lookahead** ℓ: Only the next ℓ arriving loads in A are known in advance. However, D is fully known, meaning the final position of each arriving load in the departure sequence is available.
- **Fully online**: Neither A nor D are known in advance, but the number of loads n is known.

This work focuses on the case that a single robot moves a single load at a time to focus the analysis on the number of relocations needed for full space utilization. This single robot setup also arises in practical and cost-constrained setups, such as smaller-scale, last-mile distribution centers, where avoiding rearrangements makes single-robot operation fast and effective.

We present our results in decreasing order of available prior information, i.e., (1) the offline setting, (2) the online setting with lookahead, and, finally, (3) the fully online setting.

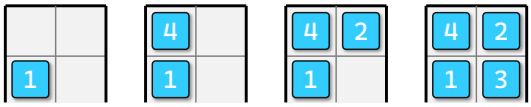

Figure 3: Consider an instance with $A = (1, 4, 2, 3)$ and $D = (1, 2, 3, 4)$. Given that load 1 must depart first, it has to be stored in the front to avoid relocations. This forces the above-shown storage sequence (or its vertical mirror, where 1 is placed on the bottom right). This leaves load 2 buried behind loads 3 and 4. This means it cannot be retrieved without a rearrangement.

The Offline Setting

This section presents a complete characterization of the existence of rearrangement-free solutions in the offline setting. In this setting, relocations may be necessary to support 100% density for narrow grid openings:

Observation 1. *For a 2×2 storage space relocations may be necessary.*

Proof. Consider the sequence $A = (1, 4, 2, 3)$ and recall that the departing sequence is fixed w.l.o.g. to be $D = (1, 2, 3, 4)$. Clearly, load 1 must be stored on the front row because it departs first, say w.l.o.g. on the left column (see Fig. 3). As load 4 arrives, we must place it behind load 1 to not block later arriving loads. This leaves an empty column, which acts like a stack, i.e., we must place load 2 in the back row and load 3 in the front. Finally, observe that the retrieval of load 2 requires relocating either of the loads 3 or 4. □

The observation extends to any $r \times 2$ case where $r \geq 3$.

An arrangement $\mathcal{A}$ is defined to *satisfy* an arrival (resp. departure) sequence A (resp. D), if all the loads can be stored (resp. retrieved) according to sequence A (resp. D) with one action per load, where $\mathcal{A}$ is the final (resp. initial) arrangement. Fix $D = [n]$, where $[n] \coloneqq (1, \ldots, n)$, leaving A as the sole sequence that can change per problem instance.

Observation 2. *An arrangement $\mathcal{A}$ satisfies an arrival sequence A if and only if $\mathcal{A}$ satisfies the departure sequence in which A is reversed.*

Proof. All the actions and corresponding access paths that bring the items into the arrangement $\mathcal{A}$ in the order A can be applied in reverse to retrieve the items in the reverse order, and vice versa. □

Following Observation 2, finding a rearrangement-free solution to `StoRMR` is equivalent to finding an arrangement $\mathcal{A}$ that satisfies two departure sequences, namely the true departure order $[n]$ as well as a permutation D' of $[n]$, where D' is the reverse of A in the original input. For example, for $A = (1, 4, 2, 3)$ and $D = (1, 2, 3, 4)$ as in Fig. 3, the two departure orders to be satisfied are the original $D = (1, 2, 3, 4)$ and $D' = (3, 2, 4, 1)$, which is the reverse of A.

Consequently, it suffices to check the following *local adjacency conditions* to determine whether a given arrangement $\mathcal{A}$ satisfies a departure sequence D':

Observation 3. *An arrangement $\mathcal{A}$ satisfies a departure sequence $D' = (d_1, \ldots, d_n)$ if and only if every load d_i is either in the bottom row or is adjacent in $\mathcal{A}$ to a load d_j that departs earlier, i.e., $j < i$.*

Assume the worst case of a maximum density of 100% (otherwise, the situation is simpler). Let D' be an input permutation of $[n]$. The following algorithm constructs a valid arrangement $\mathcal{A}$ by iteratively assigning placements bottom-up, ensuring the local adjacency conditions of Observation 3 are met for sequences $[n]$ and D'. Recall that C_1, C_2, C_3 are the columns in left-to-right order.

Algorithm 1. In the first stage, the algorithm iterates jointly over $[n]$ and D' in order and repeats the following: Let x and y be the first two items in $[n]$ and D', respectively, which have not been assigned a cell. If $x \neq y$, assign x and y to the lowest unassigned cells of C_1 and C_2, respectively. Otherwise, if $x = y$, assign x to the lowest unassigned cell in C_3. Repeat this process until one (or more) of the columns becomes fully assigned, at which point we proceed to the next stage, which has two cases:

Case 1: If C_1 and C_2 are fully assigned (notice that since we assign items to them together, they become fully assigned together), we continue filling up C_3 (bottom-up) by adding the remaining unassigned items of $[n]$ (in order).

Case 2: Otherwise, C_3 becomes fully assigned first. In this case, we fill in C_2 (bottom-up) with the remaining items of D' until C_2 is filled up. Then, we fill C_1 similarly with the remaining items of $[n]$.

Theorem 1. *For an $r \times 3$ storage area with $r \geq 1$, there is an offline rearrangement-free solution. The solution can be found in $O(n)$ time, where n is the number of loads.*

Proof. The proof is based on the above algorithm. It verifies that the local adjacency conditions hold for each non-bottom item, when it is assigned a cell in $\mathcal{A}$.

For the first phase and any item x in C_1, the adjacent item z below it (resp. y to its right) appears before x in $[n]$ (resp. in D'), which follows from the order of placement. Thus, the adjacency conditions hold for x. A similar argument follows for C_2. Finally, notice that for any item w in C_3, the adjacent item z below it must appear before w in both $[n]$ and D'.

Let us examine the second phase:

Case 1: For an item w placed in this phase, the adjacent item z on its left (resp. z' below it) appears before w in D' (resp. in $[n]$).

Case 2: For an item w added to C_2, the adjacent item z on its right appears before w in both $[n]$ and D'. When an item w is added to C_1, the adjacent item z on its right (resp. z' below it) appears before w in D' (resp. in $[n]$).

Finally, it is straightforward to verify that the algorithm runs in $O(n)$ time. □

We can now use Theorem 1 as a building block to obtain a solution without relocation for storage spaces with more than three columns. The proof of Theorem 2 will be provided when we prove Theorem 3, a more general result.

Theorem 2. *For an $r \times c$ storage area with $r > 1$, an offline rearrangement-free solution always exists if and only if $c \geq 3$.*

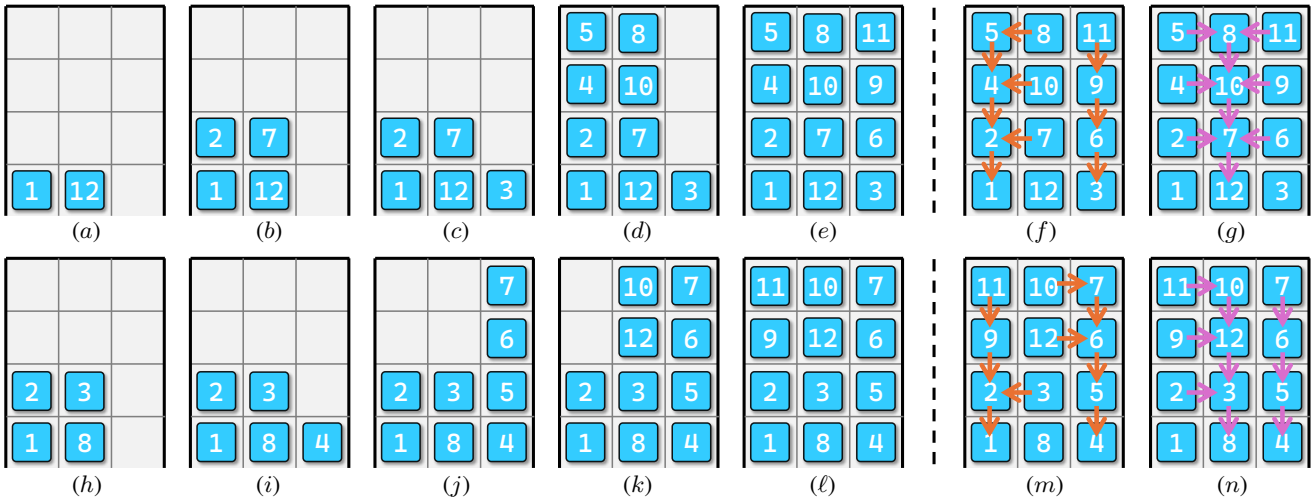

Figure 4: Running Algorithm 1 on the inputs $D' = (\underset{1}{12}, \underset{2}{7}, \underset{3}{3}, \underset{4}{1}, \underset{5}{10}, \underset{6}{8}, \underset{7}{9}, \underset{8}{11}, \underset{9}{6}, \underset{10}{4}, \underset{11}{2}, \underset{12}{5})$ (top row) and $D' = (\underset{1}{8}, \underset{2}{3}, \underset{3}{4}, \underset{4}{5}, \underset{5}{6}, \underset{6}{2}, \underset{7}{7}, \underset{8}{12}, \underset{9}{1}, \underset{10}{10}, \underset{11}{9}, \underset{12}{11})$ (bottom row). **Top**: (a)(b) Partial solutions after the first two iterations, where non-equal matching loads of D and D' are put in the left two columns C_1 and C_2. (c) In the third iteration, D and D' have an equal matching load 3, which is put in C_3. (d) The next two iterations fill up columns C_1 and C_2. This leads to the solution following case 1 from here on. (e) The leftover loads in C_3 are filled up to complete the arrangement. The solution with arrows showing the guaranteed local adjacencies for $[12]$ and D' is shown in (f) and (g), respectively. **Bottom**: (h)(i) A partial solution after two and three iterations. (j) The next three iterations fill up C_3, leading the algorithm into case 2. (k) The next two iterations fill up C_2 with loads 10 and 12. (l) C_1 is filled up to complete the arrangement. (m) and (n) show the solution with local adjacency as in the top row. Notice that the arrows indicate guaranteed paths for retrieval (in (f) and (m)) and for storage (in (g) and (n), when reversed. Better paths requiring fewer sideway-move actions may exist.

Online Setting with Lookahead

This section analyzes the variant where the position of each arriving load in the departure sequence D is known but only a limited lookahead is available for the arrival sequence A. Practical considerations for the presented storage placement strategies are also discussed.

Unlike the offline case of Theorem 1, there is no longer access to the reversed version of A, so the following analysis works with the original arrival sequence A. Consider now a generalization of Theorem 1 and the proof of Theorem 2 via the following more general statement.

Theorem 3. *For an $r \times c$ storage area with $c \geq 3$, a rearrangement-free solution can be found with a $\ell = 3r - 1$ lookahead and in $O(n \log r)$ time. Furthermore, this can be done with column-adjacent access paths.*

Proof. Assume a density of 1; otherwise, the situation is simpler. The argument proceeds in two stages. First, iterate over the $c - 3$ leftmost columns in left-to-right order, filling up a column at a time as follows (see Figure 5 for an example): Let C denote the current column, and let $s = (a_{(1)}, a_{(2)}, \ldots, a_{(r)})$ denote the next r loads in A sorted per departure order. Notice that we can determine s using a lookahead of $r < \ell$. We proceed to place each load $a_{(i)}$ in the cell of column C located on the (i)-th row away from the front row. At this stage, notice that each cell in C is accessible from the next column to the right, which is empty. Consequently, each storage path is column adjacent. The placement also ensures that each load can be retrieved directly using only C, as the load below it departs first.

After the first stage, we apply Algorithm 1 used in Theorem 1 to the remaining three columns and inherit the theorem's properties. The algorithm is offline and requires knowing the full arrival sequence A, i.e., knowing the remaining $3r$ arriving loads. A $3r - 1$ lookahead suffices as the last arriving load can be easily deduced given the rest.

Finally, the running time follows from sorting r loads at a time for at most n/r batches. □

The proof of Theorem 3 makes it clear that for a known departure order, it is possible to fill up columns as loads arrive without much lookahead, for most of the storage space, while guaranteeing no rearrangement is needed during the retrieval phase. Proposition 1 formalizes this observation.

Proposition 1. *For an $r \times c$ storage area and $n \leq r(c-1)+1$ loads, a rearrangement-free solution can be found with a lookahead of 1 and $O(n)$ time. Furthermore, this can be done with column-adjacent access paths.*

Proof. Let us assume there are $n = r(c-1)+1$ loads, as it is straightforward to handle fewer loads. The goal is to fill the leftmost $c - 1$ columns top-down such that at the departure stage, the loads in each column C_i depart after the loads at column C_{i+1} have departed. We store each load in a column at the current topmost available cell based on its departure order: Load 1 is stored at C_c, at its front cell. Loads $2, \ldots, 2+r-1$ are stored at C_{c-1}, loads $2+r, \ldots, 2+2r-1$

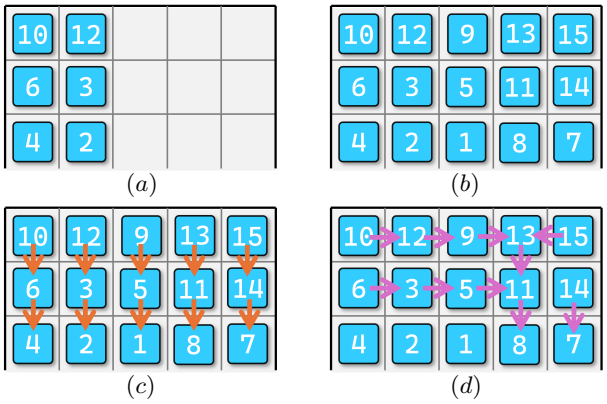

Figure 5: An example execution of the algorithm described in Theorem 3, which also works for proving Theorem 2. $A = (4, 10, 6, 12, 2, 3, 9, 15, 1, 14, 13, 7, 5, 11, 8)$ and $D = [15]$. (a) Knowing the first three loads to arrive are $4, 10, 6$, we store them in the order they depart, which is $4, 6, 10$ from bottom to top. These go to C_1. Similarly, the next three arrivals are stored in C_2 as $2, 3, 12$, from bottom to top. (b) For the next 9 arrivals, we run the algorithm from Theorem 1. For this, we turn the remaining loads of A backward to get $D' = (8, 11, 5, 7, 13, 14, 1, 15, 9)$ and the corresponding portion of D is $(1, 5, 7, 8, 9, 11, 13, 14, 15)$. (c)(d) Arrows showing how departures and arrivals can be handled without rearrangements, respectively. Note that the arrows for arrivals are drawn backwards to be consistent with Fig. 4.

are stored at C_{c-2}, and so on. Each load is stored via a straight path and retrieved using either a straight or column-adjacent path (via the column to the right). $O(n)$ time suffices, as each load requires constant time to calculate its respective paths. □

The observation leading to Proposition 1 allows to show that if W is square-shaped or wider (at its open side), full-capacity storage with limited relocations is possible.

Theorem 4. *For an $r \times c$ storage area with $r \leq c$, a solution with at most $r - 1$ relocations can be found with a lookahead of 1 and $O(n)$ time.*

Proof. We generalize the column-filling approach from Proposition 1 to a path-filling approach, i.e., we define a sequence of paths, each of which will contain loads departing only after the loads on the next path. The first $c - r$ paths are the leftmost $c - r$ columns (we have $c - r \geq 0$). The remaining portion of W is an $r \times r$ square. Take the next path to be the square's leftmost column and top row, i.e., an upward segment and a rightward segment. Define the next path similarly for the remaining $(r - 1) \times (r - 1)$ square, and so on. Figure 6 visualizes the corresponding notions.

Denote the resulting complete sequence of paths by $\pi_1, \ldots, \pi_c$. Each path contains a cell on the front row, so we can fill each path with arriving loads back to front (similarly to a stack). Loads are assigned to the paths based on their departure order: Load 1 is assigned to π_c (which is a single cell), loads $2, 3, 4$ are assigned to π_{c-1}, and so on, with the last $|\pi_1|$ departing loads assigned to π_1.

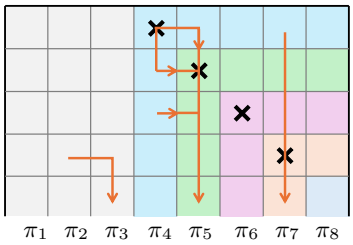

Figure 6: The storage strategy of Theorem 4 for a (5×8) grid. Paths π_1–π_3 are the left three vertical columns. Paths π_4–π_7 are L-shaped and distinguished using different colors. π_8 is a single cell. Corner cells are marked with crosses, and the possible types of retrieval paths are shown using arrows, except for loads that can be relocated.

As loads arrive, they are stored using the paths above, using one action and at most one turn. On departure, each load located solely on a vertical segment of some π_i (for $i < c$) can be retrieved using a column-adjacent path using the column to the right, i.e., via π_{i+1}, since the loads on π_{i+1} depart earlier. Loads located solely on a horizontal segment of some π_i can be similarly retrieved via a downward path. Both of the latter types of loads require only one action. A load located on a "corner" of some π_i, however, may not be adjacent to an earlier departing load and hence may require relocation. There are $r - 1$ such loads and hence at most $r - 1$ relocations. Let x be such a load that is blocked at its departure. We relocate the adjacent load y to the right of x one cell down and one cell right, which is possible since π_{i+1} is empty at this stage. The relocation results in a clear retrieval path for x, allows y to be retrieved via a downward path, and does not block other loads. Notice that it may also be possible to relocate a blocking load of x one cell right/down for x to be retrieved, depending on which loads have already departed. Overall, each load requires at most 2 actions to be retrieved.

The total time to compute all actions is $O(n)$ as each load's storage and retrieval takes constant-time computation. □

We combine our results to obtain a $9/8$-approximation algorithm for `StoRMR`:

Corollary 1. *For an $r \times c$ storage area with $r \leq c$, `StoRMR` can be solved with a number of actions within $9/8$ of the optimum using a lookahead of 1 and $O(n)$ time.*

Proof. If $n \leq r(c - 1) + 1$, we can apply Proposition 1 and achieve no relocations. Otherwise, suppose $n = rc - k$ for some $0 \leq k < r - 1$, i.e., we have k empty cells after filling W. Then, we apply Theorem 4 while not using k of the $r - 1$ corners. Since only a load placed in a corner can result in a relocation during retrieval, we have at most $2n + (\#\text{corners occupied}) = 2n + r - 1 - k$ total actions. Therefore, substituting n, we get the approximation

$$\frac{2n + r - 1 - k}{2n} = \frac{2rc + r - 1 - 3k}{2rc - 2k} \leq \frac{2rc + r - 1}{2rc} \leq \frac{9}{8}.$$

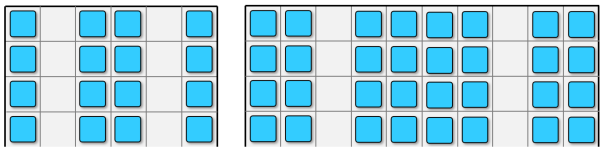

Figure 7: Left: A 4×6 storage area with 1-deep aisles. Right: A 4×10, 2-deep aisle arrangement.

The first inequality is straightforward, while for the second one, the ratio is maximized for $r = c = 2$. □

The Fully Online Setting

In the fully online setting, density typically has to be limited to minimize rearrangement. This means that the arrangement of the loads in W needs to be carefully chosen. This section analyzes the relationship between storage density and the number of actions required for storage and retrieval.

Depth and density. Recall first the number of loads that may potentially block a load to be retrieved; an arrangement of loads $\mathcal{A}$ has *depth* k if every load can be retrieved by first relocating/retrieving no more than $k - 1$ other loads.

Given a fixed depth $k \in \mathbb{N}$, a key question is to determine the maximum density that can be accommodated while ensuring an arrangement of the loads with depth at most k exists. Denote this density by $\rho(k)$. This relationship is investigated by (Gue 2006), which establishes the following:

Theorem 5 Upper bound, (Gue 2006)**.** *For rectangular 2D grids,* $\rho(k) \leq 2k/(2k + 1)$.

In the setting of (Gue 2006), however, all the free cells are required to form a single connected component containing a single input/output cell. The proposed setting in this work does not impose such a requirement, as the assumption is that the whole bottom row of W can be used to store and retrieve loads. Therefore, it is possible to realize the upper bound using the following simple aisle-based approach.

Proposition 2. *For a given depth* k*, if the number of columns* c *in* W *is a multiple of* $2k + 1$*, then there is an arrangement that achieves the optimal density of* $2k/(2k + 1)$.

Proof. We partition the columns into contiguous groups of $2k + 1$ columns and leave only the middle column of each group empty as an aisle. See Figure 7. This construction achieves the optimal density while ensuring a depth of k. □

We call the latter pattern a k**-deep aisle** arrangement and call the empty columns in it *aisles*. We remark that the 1-deep aisles case is prevalent in practice, e.g., at libraries or stores, where goods can be stored and retrieved using a single action via an aisle. Proposition 2 explains how this natural strategy is effective in balancing storage density and access efficiency by setting $k = 1$ in this particular case.

Action bounds and density. We now analyze the connection of depth and density to the worst-case number of actions in the fully online setting. We show that the trade-off between the number of actions and density carries over.

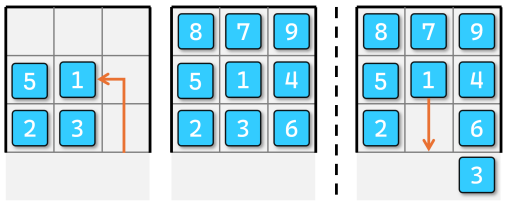

Figure 8: Snapshots of the baseline solution for $A = (5, 2, 3, 1, 8, 7, 9, 4, 6)$. Left: The first three arriving loads are stored in their respective designated rows. When 1 arrives, storing it in the front row would block access to empty cells, and so it is stored in the next row. Middle: The last three loads are stored in the right column in their arrival order. Right: Upon retrieval, 3 is relocated outside of W to let 1 out. Load 3 is then stored back in its assigned cell, after which no relocations occur.

Theorem 6. *To guarantee at most* a *actions for each fully online storage and retrieval, the density must be at most* $2a/(2a + 1)$*. The bound is asymptotically tight: a* $2a/(2a + 1) - \varepsilon$ *density is achievable for a small* $\varepsilon > 0$.

Proof. By Theorem 5 we cannot obtain a density greater than $2a/(2a + 1)$, as otherwise after storing all the loads, there would be a load that requires more than a actions to be retrieved due to blocking loads.

We can achieve a density that approaches $2a/(2a + 1)$ as either (or both) of the grid dimensions tends to infinity. We use an a-deep aisle arrangement $\mathcal{A}$, with the slight change of keeping a "buffer" of $a - 1$ empty cells, which we take to be any cells adjacent to an aisle. We first store the loads per $\mathcal{A}$, which can be easily done using one action per load since we do not assign loads to specific cells.

For retrieval, our goal is to keep each aisle empty after each load is retrieved, which ensures that the depth of any subsequent arrangement remains at most a. To this end, when we retrieve a load, we relocate each load that blocks the way to an empty non-aisle cell (potentially temporarily moving the blocking load outside of the grid on its way to an empty cell). Since we start with $a-1$ such "buffer" cells, and at most $a - 1$ loads are relocated, which can always be done. □

Note that without any information on the departure/arrival sequences, one has to be conservative when choosing an appropriate arrangement and sacrifice density to ensure that retrievals can happen within a given number of actions.

Experimental Validation

This section experimentally validates theoretical guarantees established for the offline setting (Theorem 3). Although the guarantees already eliminate relocations at full density, we compare it to a natural baseline to illustrate how omitting the structural principles identified by this work and the proposed method affects performance. We also provide a set of experiments that consider uncertainty in the departure sequence D and demonstrate that the offline algorithm's performance degrades gracefully, further affirming its applicability.

		(a) Offline					(b) n swaps					(c) $3n$ swaps				
metric	# rows/columns	10	15	20	25	30	10	15	20	25	30	10	15	20	25	30
# actions	baseline	127	283	496	769	1,104	125	284	500	772	1,142	128	287	496	780	1,110
	proposed	**100**	**225**	**400**	**625**	**900**	**106**	**233**	**411**	**637**	**916**	**110**	**244**	**419**	**652**	**929**
	baseline subopt.	27%	26%	24%	23%	23%	25%	26%	25%	23%	27%	28%	27%	24%	25%	23%
	proposed subopt.	**0%**	**0%**	**0%**	**0%**	**0%**	**6%**	**4%**	**3%**	**2%**	**2%**	**10%**	**8%**	**5%**	**4%**	**3%**
total distance ($\times 10^3$)	baseline	1.39	4.56	10.5	20.0	34.3	1.38	4.55	10.5	20.2	34.9	1.40	4.54	10.5	20.3	34.5
	proposed	**1.17**	**3.77**	**8.73**	**16.8**	**28.7**	**1.20**	**3.83**	**8.84**	**16.9**	**28.9**	**1.22**	**3.92**	**8.92**	**17.1**	**29.1**
	baseline subopt.	26%	27%	25%	23%	23%	25%	26%	25%	24%	25%	27%	26%	25%	25%	24%
	proposed subopt.	**6%**	**5%**	**4%**	**3%**	**3%**	**9%**	**6%**	**5%**	**4%**	**4%**	**11%**	**9%**	**6%**	**5%**	**4%**

Table 1: Performance of the proposed algorithm and the baseline during retrieval in the offline setting (a) and under two levels of departure uncertainty (b-c) in which random pairs of loads in D are swapped. Distances are reported in thousands.

As no prior work readily applies to our setup, we define a **baseline** that stores earlier departing loads towards the front: specifically, it aims to store the first c departing loads on the front row, the next c departing loads on the second row, and so on. Each row is filled from left to right without relocations in the storage phase: If storing a load in its designated row disconnects empty cells from the front row, the load is stored on the next available row such that access to remaining empty cells is maintained. Distance-optimal paths are used, except that during retrieval, we first minimize the number of blocking loads along the path (and then minimize distance). As opposed to the proposed algorithm, the baseline may have to temporarily move blocking loads outside of W; see Figure 8. Each load that blocks the current target load is moved outside of the grid along the retrieval path and, after the target load is retrieved, then stored back in its original cell along the same path. A supplementary video shows the solutions of the proposed algorithm and the baseline side by side.

We use square grids filled to capacity, measuring actions during retrieval and the total distance traveled by loads (in grid cell units), averaged over 25 random instances for each grid size; see Table 1 (a). For the baseline, the shown distance considers the motion of loads only inside W, as this is the focus of this work. For each cost metric, the suboptimality (subopt.) is reported as the percentage increase over the optimum for actions, or over a lower bound, for distance. For distance, we take the lower bound of $m^3 + m^2$ for a grid side length of m in lieu of the optimum.

The results for the proposed algorithm confirm the guarantees: it is optimal for relocation actions and at most 6% suboptimal for total distance, confirming the near-optimality of short sideways motions in the column-adjacent paths. For both metrics, the baseline is always $> 20\%$ suboptimal.

Robustness to departure uncertainty. In practice, perfect prior information may not be available, as unforeseen delays or changes in priority can alter the retrieval sequence. We evaluate the robustness of our approach under such uncertainty as follows. We begin as in the first set of experiments, storing the loads using the arrangement computed by each algorithm (using A and D). We then perturb the departure sequence D by randomly swapping pairs of adjacent loads: we consider n swaps, i.e., the number of loads, (Table 1 (b)) and $3n$ swaps (Table 1 (c)). We then perform retrieval using the perturbed sequence, where any relocations for a blocked load are performed as before. For both uncertainty levels, the proposed method proves robust: its suboptimality for the metrics stays below 11%, diminishing with grid size, and maintaining superiority over the baseline. These results further demonstrate the practicability of the proposed method, indicating that its strong performance is not confined to the perfect information case. We remark that an algorithmic extension for retrieval uncertainty is given in (Geft et al. 2026).

Conclusion

This paper analyzes the planning of ordered storage and retrieval of uniform loads in high-density 2D grid-based storage while minimizing intermediate load rearrangement. We establish an intriguing, sharp characterization for the offline case, showing that rearrangement can always be avoided if and only if we can access a side at least 3 cells wide. When only the departure sequence is known, we also identify positive results. These positive results contrast with stack-based storage settings, where related problems are NP-hard (Hanou, de Weerdt, and Mulderij 2023; Caserta, Schwarze, and Voß 2020). In this sense, our results, especially Theorem 1, are surprising, as at first glance, some of our variants appear NP-hard as well. From a practical standpoint, our results – both the proposed dense storage configurations and the associated algorithms – are promising for real-world adoption.

Despite the sequential nature of the plans produced by our key algorithmic results, we expect these plans to be amenable to parallel execution by multiple robots/AGVs. As an illustrative example, when executing plans according to Theorem 3, it is done column by column, until the last three. For the leftmost $c - 3$ columns, loads can be carried in parallel from the right side of the column by multiple robots, which can then exit downward from the column underneath the loads, with minimum delays and conflicts. As the last three columns get filled, after storing a load, robots can move to the top and then left to exit downward from C_{c-3}.

Additional future research could allow storage and retrieval to be interleaved in an arbitrary manner, as opposed to storage strictly followed by retrieval. Finally, for cases where we require a lookahead greater than 1, it is natural to ask whether a lower lookahead is possible.

Acknowledgements

We thank the reviewers and editorial staff for their insightful suggestions. This work is supported in part by NSF awards IIS-1845888, IIS-2021628, IIS-2132972, CCF-2309866, and an Amazon Research Award.

References

Bela, V. 2024. China stakes global dominance in race to build intelligent ports. https://www.scmp.com/news/china/science/article/3250341/china-stakes-global-dominance-race-build-intelligent-ports. Accessed: 2025-01-24.

Boge, S.; and Knust, S. 2020. The parallel stack loading problem minimizing the number of reshuffles in the retrieval stage. *Eur. J. Oper. Res.*, 280(3): 940–952.

Bukchin, Y.; and Raviv, T. 2022. A comprehensive toolbox for load retrieval in puzzle-based storage systems with simultaneous movements. *Transportation Research Part B: Methodological*, 166: 348–373.

Caserta, M.; Schwarze, S.; and Voß, S. 2020. Container rehandling at maritime container terminals: A literature update. *Handbook of terminal planning*, 343–382.

Chen, L. Y.; Huang, H.; and Goldberg, K. 2024. Optimal Arrangement and Rearrangement of Objects on Shelves to Minimize Robot Retrieval Cost. *IEEE Trans Autom. Sci. Eng.*, 21(3): 2184–2198.

Dogar, M. R.; Koval, M. C.; Tallavajhula, A.; and Srinivasa, S. S. 2014. Object search by manipulation. *Autonomous Robots*, 36: 153–167.

Geft, T.; Zhang, W.; Yu, J.; and Bekris, K. 2026. Robust Out-of-Order Retrieval for Grid-Based Storage at Maximum Capacity. In *AAAI*, volume 40, 36245–36252.

Gue, K. R. 2006. Very high density storage systems. *IIE transactions*, 38(1): 79–90.

Gue, K. R.; and Kim, B. S. 2007. Puzzle-based storage systems. *Naval Research Logistics (NRL)*, 54(5): 556–567.

Gue, K. R.; Uludag, O.; and Furmans, K. 2012. A high-density system for carton sequencing. In *Proceedings of the international material handling research colloquium*.

Guo, T.; and Yu, J. 2023. Toward Efficient Physical and Algorithmic Design of Automated Garages. In *ICRA*, 1364–1370. IEEE.

Hakan Akyüz, M.; and Lee, C.-Y. 2014. A mathematical formulation and efficient heuristics for the dynamic container relocation problem. *Naval Research Logistics (NRL)*, 61(2): 101–118.

Hanou, I. K.; de Weerdt, M. M.; and Mulderij, J. 2023. Moving Trains like Pebbles: A Feasibility Study on Tree Yards. In *ICAPS*, 482–490. AAAI Press.

Huang, H.; Danielczuk, M.; Kim, C. M.; Fu, L.; Tam, Z.; Ichnowski, J.; Angelova, A.; Ichter, B.; and Goldberg, K. 2022. Mechanical Search on Shelves using a Novel "Bluction" Tool. In *ICRA*, 6158–6164. IEEE.

Huang, H.; Dominguez-Kuhne, M.; Satish, V.; Danielczuk, M.; Sanders, K.; Ichnowski, J.; Lee, A.; Angelova, A.; Vanhoucke, V.; and Goldberg, K. 2021. Mechanical Search on Shelves using Lateral Access X-RAY. In *IROS*, 2045–2052. IEEE.

Kota, V. R.; Taylor, D.; and Gue, K. R. 2015. Retrieval time performance in puzzle-based storage systems. *Journal of Manufacturing Technology Management*, 26(4): 582–602.

Lersteau, C.; and Shen, W. 2022. A survey of optimization methods for Block Relocation and PreMarshalling Problems. *Computers & Industrial Engineering*, 172: 108529.

Roodbergen, K. J.; and Vis, I. F. A. 2009. A survey of literature on automated storage and retrieval systems. *Eur. J. Oper. Res.*, 194(2): 343–362.

van Brink, M.; and van der Zwaan, R. 2014. A Branch and Price Procedure for the Container Premarshalling Problem. In *ESA*, volume 8737 of *Lecture Notes in Computer Science*, 798–809. Springer.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1): 9–9.

Yalcin, A. 2017. *Multi-agent route planning in grid-based storage systems*. Ph.D. thesis, Europa-Universität Viadrina Frankfurt.

Yu, W.; and Egbelu, P. J. 2008. Scheduling of inbound and outbound trucks in cross docking systems with temporary storage. *Eur. J. Oper. Res.*, 184(1): 377–396.

Zehendner, E.; Feillet, D.; and Jaillet, P. 2017. An algorithm with performance guarantee for the Online Container Relocation Problem. *Eur. J. Oper. Res.*, 259(1): 48–62.