Robust Out-of-Order Retrieval for Grid-Based Storage at Maximum Capacity

Tzvika Geft, William Zhang, Jingjin Yu, Kostas Bekris

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

Abstract

This paper proposes a framework for improving the operational efficiency of automated storage systems under uncertainty. It considers a 2D grid-based storage for uniform-sized *loads* (e.g., containers, pallets, or totes), which are moved by a robot (or other manipulator) along a collision-free path in the grid. The loads are labeled (i.e., unique) and must be stored in a given sequence, and later be retrieved in a different sequence—an operational pattern that arises in logistics applications, such as last-mile distribution centers and shipyards. The objective is to minimize the load relocations to ensure efficient retrieval. A previous result guarantees a zero-relocation solution for known storage and retrieval sequences, even for storage at full capacity, provided that the side of the grid through which loads are stored/retrieved is at least 3 cells wide. However, in practice, the retrieval sequence can change after the storage phase. To address such uncertainty, this work investigates k-*bounded perturbations* during retrieval, under which any two loads may depart out of order if they are originally at most k positions apart. We prove that a $\Theta(k)$ grid width is necessary and sufficient for eliminating relocations at maximum capacity. We also provide an efficient solver for computing a storage arrangement that is robust to such perturbations. To address the higher-uncertainty case where perturbations exceed k, a strategy is introduced to effectively minimize relocations. Extensive experiments show that, for k up to half the grid width, the proposed storage-retrieval framework essentially eliminates relocations. For k values up to the full grid width, relocations are reduced by $50\%+$.

Introduction

Modern logistics systems increasingly rely on automation technologies for transporting uniform-sized *loads*, such as containers, pallets, and totes. Operations at some logistics hubs, especially last-mile and small-scale distribution centers, occur in two distinct phases: first, the storage of incoming loads (such as when a delivery semi-truck arrives), and later, their retrieval for onward transport (such as last-mile, local delivery trucks). Such scenarios arise in container terminals (Bela 2024), cross-docking facilities (Yu and Egbelu 2008), automated warehouses with fleets of mobile robots (Wurman, D'Andrea, and Mountz 2008) and Automated Storage and Retrieval Systems (AS/RS) (Roodbergen and Vis 2009; Yalcin 2017; Gue, Uludag, and Furmans 2012). A central challenge these systems must contend with is trading off between maximizing space utilization and storage/retrieval efficiency, as denser storage necessarily makes arbitrary load access more difficult.

This paper investigates this trade-off in high-density, 2D grid-based storage, akin to Puzzle-Based Storage (PBS) (Gue and Kim 2007). In this setting, each cell of a rectangular $r \times c$ grid can hold one load, which can be moved by a mobile robot or manipulator along cardinal directions via empty cells. Assuming the grid is accessible for storage and retrieval from only one side, the following three types of actions are available: *(i) storage* of an arriving load, *(ii) retrieval* of a departing load, or *(iii) relocation (rearrangement)* of a load from one cell to another. Loads must be stored in a given known order A and must then be retrieved according to an *anticipated* retrieval order D, which might change, while minimizing relocations. See Figure 1; a formal problem definition follows below.

Assuming full prior knowledge of the storage and retrieval sequence, prior work shows that rearrangements can be avoided, even when the grid is to be fully occupied, provided that the grid's open, access side is at least 3 columns wide (Geft, Bekris, and Yu 2025). While this result eliminates the aforementioned trade-off under full order observability, in practice, the complete load sequence is generally unavailable due to operational uncertainties. This paper instead asks: *can rearrangements be eliminated or minimized under significant storage/retrieval sequence uncertainty?* This work provides a positive answer through a two-part framework that combines robust storage and effective retrieval, thereby broadening the applicability of high-density grid-based storage under uncertainty and offering design guidelines.

Contribution. This work introduces a novel storage and retrieval problem variant that incorporates uncertainty through k-*bounded perturbations*, under which any two loads can depart (or, interchangeably, arrive) out of order if they are planned to depart at most k positions apart. To solve the problem, two complementary approaches are presented:

Robust storage. We generalize deterministic zero-relocation solution conditions to handle uncertainty through k-*robust* storage arrangements, which solve the problem with no relocations under k-bounded perturbations. Given k, we provide asymptotically tight bounds showing that $\Theta(k)$

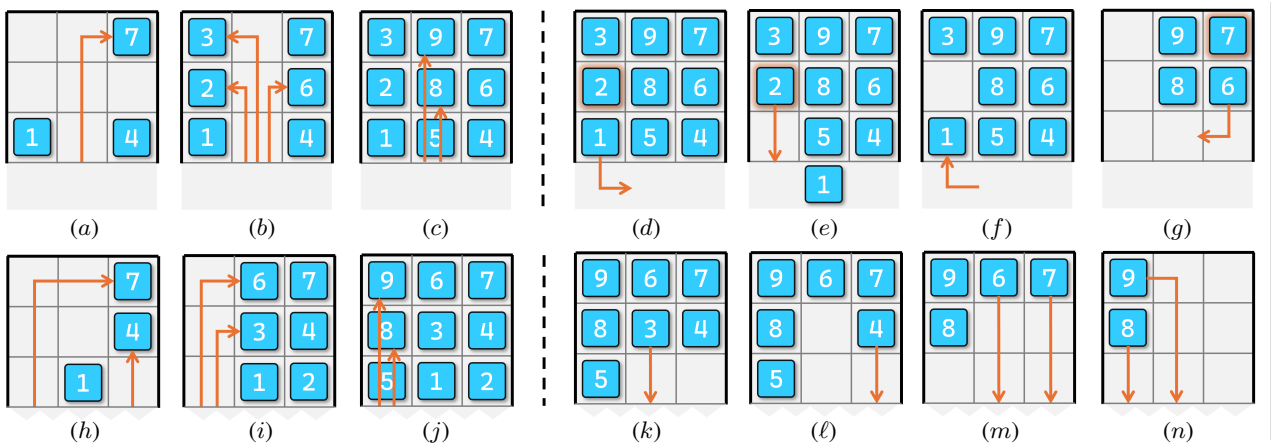


Figure 1: Consider a 3×3 storage area W accessible only from one side (bottom) that must store 9 loads arriving in the order $A = (4, 1, 7, 6, 3, 2, 9, 8, 5)$ and planned to depart in the order $D = (1, 2, \ldots, 9)$. **Top row:** a solution that avoids relocations if D does not change. (a) The first three arriving loads, $4, 1, 7$, are stored. (b) Loads $6, 3, 2$ are stored. (c) Loads $9, 8, 5$ are stored. At this point, while the loads can depart according to D (not shown), the actual retrieval sequence becomes $\tilde{D} = (2, 1, 3, 5, 4, 7, 6, 9, 8)$, a slight perturbation of D, requiring relocations: (d) Load 2 is blocked, so 1 is relocated outside W. (e) Load 2 is retrieved. (f) Load 1 is stored back until it is needed. (g) After $1, 3, 5, 4$ are retrieved, load 7 is next but is blocked by 6, which is relocated within W. **Bottom row:** (h)–(j) Loads are stored according to A but in a robust arrangement, as proposed in this work. (k)–(n) In this solution, loads can be retrieved without relocations, not only according to D, but also under $\tilde{D}$.

columns are necessary and sufficient to find a k-robust arrangement. We also develop a fast solver that finds robust arrangements for k values in line with our bounds.

Retrieval strategy. As rearrangements may be required as k increases, a load relocation problem arises: Given a target load for retrieval that is blocked by other loads, compute relocation actions for the blocking loads with the goal of minimizing future relocations. We propose a greedy approach that prioritizes relocations within the storage area while accounting for future retrievals.

Comprehensive experiments for storage at full capacity show that the two approaches combined significantly outperform baselines, reducing the number of relocations by up to 60-70%, the usage of a buffer row outside of the storage area, and the distance traveled by loads, while maintaining computational efficiency as the grid size grows.

Related Work

Many works study either high-density grid-based storage or ordered storage/retrieval problems involving a sequence of loads. Their intersection, however, has received less attention. The puzzle-based storage (PBS) model addresses layout and retrieval for 2D grids with only one or few empty *escort* cells that enable motion (Gue and Kim 2007; Kota, Taylor, and Gue 2015; Bukchin and Raviv 2022). Retrieval problems in PBS focus on one or a few target loads at time, without considering a complete retrieval sequence. In settings where multiple loads are to be retrieved, the retrieval order is freely chosen (Mirzaei, Koster, and Zaerpour 2017; He et al. 2023).

Storage and retrieval with a given sequence has been studied for train-yards (Hanou, de Weerdt, and Mulderij 2023) and for stack-based ship containers. In the Block Relocation Problem (BRP) (Caserta, Schwarze, and Voß 2012a) uniform loads are stored in vertical stacks where only the top load in a stack is accessible. BRP asks to retrieve all the loads in a given order while minimizing relocations of loads between stacks. In the spirit of this work, the BRP family addresses deviations from a planned storage/retrieval sequence. (Boge, Goerigk, and Knust 2020) treat the retrieval priorities (where each priority refers to a batch of retrieved loads) as uncertain: the realized order may differ from the planned one by at most Γ pairwise inversions. The Stochastic Container Relocation Problem (Galle et al. 2018) instead reveals the retrieval sequence in batches, with the order unknown within each batch. (Boschma, Mes, and de Vries 2023) models uncertainty for both storage and retrieval, applying approximate dynamic programming.

Nevertheless, the ordered retrieval of loads in grid settings with relocation minimization remains nascent. A recent early effort (Disselnmeyer et al. 2024) applies BRP techniques, assigning a removal direction to each load, which requires many empty aisles. Multi-Agent Path Finding (MAPF) (Stern et al. 2019) addresses motion planning for robots in grids. MAPF works for warehouses (Li et al. 2021) typically assume aisles to increase throughput (sacrificing storage space), while we focus on maximizing capacity with sequential motion.

Problem Definition

Consider a rectangular $r \times c$ grid storage area W with r rows and c columns. The bottom (front) row of W is the open side of W through which loads are stored/retrieved. Denote the columns by $C_1, \ldots, C_c$ in a left-to-right order. The loads have distinct labels $1, \ldots, n \leq rc$. The *density* of a storage space having n loads is $n/(rc)$. Unless otherwise stated, assume full capacity storage, i.e., $n = rc$. Each load occupies exactly one grid cell. An *arrangement* $\mathcal{A}$ of a set of

loads is an injective mapping of loads to grid cells, i.e., an arrangement specifies a distinct (row, column) pair for each load. Two loads are *adjacent* in an arrangement if they are located in horizontally or vertically adjacent grid cells.

I/O row. We assume there is an Input/Output (I/O) row adjacent to the front row of W; loads appear/disappear on the I/O before/after storage. The I/O row also serves as a temporary *buffer* that can be used when relocating loads, e.g., to facilitate access to a load. The I/O row is not used for storage. Denote this row as IO and $W^+ = W \bigcup IO$.

Load movement. Each load can be moved by a robot via a path of empty cells along the four cardinal directions (up, down, left, or right). To pick up a specific target load, the robot must reach the cell occupied by the target load. The following types of *actions* are valid:

- Storage: A load can be *stored* in an empty cell $v \in W$ via a path (of empty cells) from any cell $u \in IO$ to v (i.e., the load to be stored appears on u).
- Retrieval: A load at cell $v \in W$ can be *retrieved* from W via a path (of empty cells) to any cell $u \in IO$.
- Relocation: A load can be *relocated* within W^+ to an empty cell via a path (of empty cells).

Denote the arrival sequence, i.e., the order in which loads arrive so as to be stored, by $A = (a_1, \ldots, a_n)$. Without uncertainty, 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 always be relabeled.

We now introduce the notion of *bounded perturbation* to model the uncertainty of arrival/departure sequences.

Definition 1. Let $\tilde{S}$ be a permutation of sequence $S = (s_1, \ldots, s_n)$. We say that $\tilde{S}$ is a *k-bounded perturbation* of S if for every pair of elements s_i, s_j with $i < j$ in the reversed order in $\tilde{S}$ (i.e., s_j appears before s_i): $j - i \leq k$.

Example: A 2-bounded perturbation of $S_5 = (1, 2, 3, 4, 5)$ is $(2, 3, 1, 5, 4)$. In this example the inverted pairs are $(1, 3)$, $(1, 2)$, and $(4, 5)$. The sequence $(1, 3, 5, 2, 4)$, however, is *not* a 2-bounded perturbation of S_5 since it contains the inverted pair $(2, 5)$ whose elements appear too far apart in S_5.

Problem 1 *Robust* Storage and Retrieval with Minimum Relocations (`R-StoRMR`)**.** *Given a storage area W with r rows and c columns, arrival and departure sequences A and D, and an integer $k > 0$, find a minimum-length sequence of actions that stores all loads according to A and then retrieves them according to a sequence $\tilde{D}$, which is a k-bounded perturbation of D. $\tilde{D}$ is revealed one load at a time during the retrieval phase and is not known during storage.*

Remark 1. Since we fix $D = [n] \coloneqq (1, \ldots, n)$, one may drop D from the input. Refer to A as the input.

Additional objectives. In addition to relocation actions, we consider two additional minimization objectives: *(i)* **I/O row usage**, defined as the number of actions in which a load is present on the I/O row at the start of the action. This metric approximates the time during which the I/O row is occupied by a static load. This time matters in practical storage systems as the I/O row may be needed for other transport operations besides providing access to W. *(ii)* The **total distance** traveled by the loads throughout the sequence of actions.

Analysis of Solutions Without Relocations

A natural goal is characterizing the conditions for solving `R-StoRMR` without relocations. We show that $\Theta(k)$ columns are necessary and sufficient. Interestingly, the bounds hold for any number of rows $r > 1$, i.e., the number of columns is the key parameter that governs zero-relocation solutions under uncertainty.

Preliminary analysis

We recall useful results for relocation-free solutions to `StoRMR` (i.e., when $k = 0$) (Geft, Bekris, and Yu 2025).

Definition 2. An arrangement $\mathcal{A}$ *satisfies* an arrival (resp. departure) ordering A (resp. D) if all loads can be stored (resp. retrieved) in the order specified by A (resp. D) with one action per load with $\mathcal{A}$ as the final (resp. initial) arrangement.

View the problem from the following reverse perspective:

Observation 1. *An arrangement $\mathcal{A}$ satisfies an arrival order A iff $\mathcal{A}$ satisfies the departure order where A is reversed.*

Following Observation 1, it suffices to treat `StoRMR` as the problem of finding an arrangement $\mathcal{A}$ that satisfies two departure orders, the true departure order $[n]$ and a permutation A^R of $[n]$, where A^R is the reverse of A. For example, given $A = (4, 1, 7, 6, 3, 2, 9, 8, 5)$ and $D = (1, 2, \ldots, 9)$ as in Figure 1, the two departure orders to be satisfied are the original D and $A^R = (5, 8, 9, 2, 3, 6, 7, 1, 4)$.

Deciding whether $\mathcal{A}$ satisfies a departure order D, amounts to checking these *adjacency conditions*:

Observation 2. *An arrangement $\mathcal{A}$ satisfies a departure order $D = (d_1, \ldots, d_n)$ iff 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., $j < i$.*

One can verify that the adjacency conditions hold for each of D and A^R in the arrangements of Figure 1, e.g., consider load 2 in (c); 2 is adjacent to 1 (as needed for D) and also adjacent to 8 (as needed for A^R). A key result for `StoRMR` is an algorithm ***BaseS*** (baseline storage), that finds a zero-relocation solution given $c \geq 3$ (Geft, Bekris, and Yu 2025):

Theorem 1. *Let W be a $r \times c$ storage area with $c \geq 3$ columns, and A and D be storage and retrieval sequences, respectively, for $n \leq rc$ loads. An arrangement $\mathcal{A}$ that satisfies both A and D can always be found in $O(n)$ time.*

Robustness to perturbations, however, requires revisiting the adjacency conditions for `R-StoRMR`.

Definition 3. Given a departure/arrival sequence S, an arrangement $\mathcal{A}$ is *k-robust for S* (or just *k-robust* if S is obvious) if $\mathcal{A}$ satisfies every k-bounded perturbation $\tilde{S}$ of S.

Proposition 1 (Robust adjacency conditions). *Let $D = (d_1, \ldots, d_n)$ be a departure sequence with $n = rc$ and let $k \geq 0$. An arrangement $\mathcal{A}$ is k-robust for D 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 appears in D at least $k + 1$ loads earlier in D, i.e., $i - j \geq k + 1$.*

Proof. Suppose $\mathcal{A}$ is k-robust but for some i the load d_i is neither on the bottom row nor adjacent to any d_j with

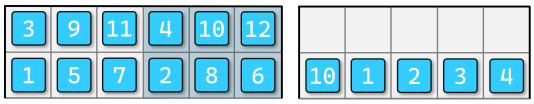


Figure 2: Examples of the column bounds for $k = 1$. **Upper bound** (left): Consider a 2×6 grid with $A = (7, 3, 11, 1, 9, 4, 6, 12, 2, 10, 8, 5)$. Per Theorem 2, we partition A into subsequences of odd and even loads: $A_0 = (7, 3, 11, 1, 9, 5)$ and $A_1 = (4, 6, 12, 2, 10, 8)$. We treat A_0 as a StoRMR instance on a 2×3 grid with $D = (1, 3, 5, 7, 9, 11)$ and use the solution to fill the leftmost 3 columns. Similarly, we treat A_1 as a separate instance for the rightmost 3 columns (shaded). The combined solution is a robust arrangement. **Lower bound** (right): Consider A^R starting with 10, 3, 4. As $k = 1$, we must have 1 and 2 on the bottom row. Treating A^R as a departure sequence to satisfy, 10 must also be on the bottom row. Next, to avoid placing 3 in the bottom row, we must store it in a cell adjacent to both 10 and 1. As no such cell exists, 3 must also be placed in the bottom row. Similarly, 4 must be adjacent to one of 1 and 2 and one of 10 and 3 to avoid the bottom row. Again, this is not possible, so 4 is also assigned to the bottom row. Thus, 5 columns are required.

$j \leq i - (k + 1)$. Define a k-bounded perturbation D' of D by taking load d_i and inserting it right after load $d_{i-(k+1)}$, or placing d_i as the first load if $i < k + 1$. That is, move d_i as early as possible and push other loads back. Since d_i has no neighbor among $\{d_1, \ldots, d_{i-(k+1)}\}$, by Observation 2, $\mathcal{A}$ doesn't satisfy D', which contradicts k-robustness.

Now assume that the adjacency condition holds in $\mathcal{A}$, and let D' be any k-bounded perturbation of D. We claim that Observation 2 applies to D'. Indeed, for each i, if d_i is not on the front row it is adjacent to some d_j with $j \leq i - (k + 1)$. No load can appear more than k spots earlier, so any load originally before position $i - (k + 1)$ still appears before d_i in D'. Thus, the adjacency condition of Observation 2 holds for d_i. Since this is the case for every d_i, $\mathcal{A}$ satisfies D'. As D' is an arbitrary perturbation, $\mathcal{A}$ is k-robust. □

Putting everything together, we have the following:

Corollary 1. *The problem of finding a zero-relocation solution for* R-StoRMR *is equivalent to finding an arrangement that satisfies A and is k-robust for D.*

Given the equivalence we ask: how many columns in terms of k, guarantee the existence of this arrangement?

Bounds on required number of columns

The bounds are tight up to 1.5, for the necessary and sufficient number of columns of a zero-relocation R-StoRMR solution.

Theorem 2 (Upper bound)**.** *For an $r \times c$ storage area W with any $r \geq 1$ rows, $3k + 3$ columns suffice to guarantee a zero-relocation solution to* R-StoRMR.

Proof. We present an algorithm and prove its correctness.

Algorithm. The idea is to partition the loads into $k + 1$ subsets, each of which will be treated as an independent StoRMR instance: More specifically, we partition the arrival sequence A into $k+1$ subsequences where each subsequence contains a_i's of the same congruence class (i.e., remainder) modulo $k + 1$. Formally, for each $j \in \{0, \ldots, k\}$, define $A_j \coloneqq (a_i \mid a_i \equiv j \pmod{k+1})$. These subsequences form a partition of the loads. We then assign the loads in each A_j to 3 dedicated contiguous columns, thus using $3k + 3$ columns in total. We then treat each A_j as a StoRMR instance for 3 columns and apply algorithm BaseS to assign the loads. See Figure 2.

Correctness. Theorem 1 guarantees that (non-robust) adjacencies are met for each A_j within the respective 3 columns. We claim that the adjacencies are also robust, as required by Proposition 1. Fix a load x that is not assigned to the bottom row. Load x has a neighboring load y in A_j that departs earlier. As A_j consists of loads with departure indices congruent modulo $k + 1$, y must appear at least $k + 1$ positions before x in D, as required. □

Theorem 3 (Lower bound)**.** *For an $r \times c$ storage area W with $r > 1$ rows, $2k + 3$ columns are necessary to guarantee a zero-relocation solution for* R-StoRMR.

Proof sketch. We present an example for the case where there are $n \geq 2k + 3$ loads that completely fill W. We set the first $k+2$ loads of A^R to be $n, k+2, k+3, \ldots, 2k+2$ (i.e., these are the last $k+2$ loads in A) and set $D = (1, 2, \ldots, n)$. First, notice that the loads $1, 2, \ldots, k + 1$ must be stored in the bottom row, as each of them may need to depart first under perturbations. Next, one may verify using induction that the first $k+2$ loads of A^R must also be stored in the bottom row; see example in Figure 2. Thus, we have $2k + 3$ loads in total that must be on the bottom row to ensure no relocations. □

Rephrasing our bounds, we conclude that for $k/c \approx k/(c - 3) \leq 1/3$ we can avoid relocations, whereas for $k/c \approx k/(c - 3) \geq 1/2$ we cannot. We observe that even though the lower bound requires $2k + 3$ columns in general, fewer columns may suffice for some instances, as occurs in Figure 1 (where 3 columns suffice for $k = 1$ instead of 5).

With guidance from the theoretical bounds, we now turn to a practical algorithm, which achieves a high success rate at finding robust arrangements even when k is roughly half the grid width, closely matching Theorem 3.

Finding a Robust Arrangement

The previous section provides guidelines on what uncertainty levels can be supported by a given storage area. We now constructively find a robust arrangement without relocations under these conditions. Although $2k+3$ columns are required in general, fewer columns might suffice for a given instance. In practice, instead of fixing k, one might rather adapt it based on the storage and retrieval sequences to maximize robustness by maximizing k.

This raises the following problem: given a R-StoRMR instance, find an arrangement $\mathcal{A}$ that is k-robust for $D = [n]$ and also satisfies A. Such an $\mathcal{A}$ corresponds to a zero-relocation solution as we establish in Corollary 1. We call such an arrangement *valid*. Similarly, a load x is *valid* if it is on the bottom row or is both *(i) D-valid*, i.e., adjacent to

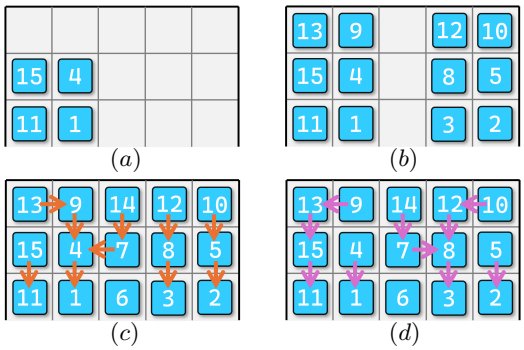


Figure 3: The storage algorithm for a 3×5 grid with $A^R = (11, 3, 15, 8, 2, 7, 13, 12, 1, 9, 10, 14, 6, 4, 5)$, $D = [15]$, and $k = 2$. (a)(b) Snapshots showing partial arrangements as the algorithm runs. First column pair: 1 and 11 are placed per column initialization. Next, we set $x = 4$ as the first load in D that can be placed above 1. In the inner loop, $y = 3$ is discarded as it does not satisfy adjacencies for D; we proceed to $y = 15$, which is valid. The main loop then sets $x = 7$, but no matching valid y is found until we reach $y = 7$, at which point 7 cannot be placed in R. Continue to $x = 8$, reaching the same conclusion. Then, for $x = 9$, a valid match is found with $y = 13$. (c)(d) Arrows showing valid adjacencies for departures and arrivals (treating arrivals in reverse).

load y with $y \leq x - k - 1$ and *(ii) A-valid*, i.e., adjacent to a load z that appears before x in A^R.

Main algorithm

The main idea is to fill pairs of adjacent columns, denote them as L and R (for left and right), from the bottom row upwards. Jointly iterate over D and A^R and greedily find pairs of loads that we can store adjacent to each other on L and R so that the loads are valid in the current (partial) arrangement. In this setup, we may skip loads as we iterate until we find valid loads to assign. We aim to have column L maintain bottom-up adjacencies for A^R while column R maintains the same for D. For the loads on L to also satisfy the adjacencies for D we rely on horizontal adjacencies. Analogously, for R and A^R. If the algorithm reaches a stage where there is no pair of loads that can be assigned, a failure is declared. Refer to Algorithm 1 (where an even number of columns is assumed for simplicity) and to Figure 3.

Initializing a column pair. For each column pair (L, R), the bottom cells are initialized to establish vertical adjacency chains (lines 5-6). For L, we initialize with next unassigned load in A^R. We aim to place the first $k + 1$ loads from D $(1, \ldots, k + 1)$ in the front row, as these loads may need to depart first under perturbations. Therefore, we initialize R with the smallest unassigned load.

Main loop (line 7). Find the next load x that can be assigned to R by taking x to be the first unassigned D-valid load, i.e., $x \geq x' + k + 1$, where x' is assigned and adjacent to the lowest empty cell on R. With x fixed, iterate over A^R, until a matching load y is found so that all adjacency conditions are met. When checking adjacencies, consider all neighboring cells of x and y that have a load assigned, to maximize success (i.e., y might "rely" on a load to its left rather than on x or the load below y). If $y = x$, **continue** as we aim to assign a pair of loads at each step.[1] If the main loop iterates without finding a valid pair to assign, return failure.

Algorithm 1: Find robust arrangement

Input: R-StoRMR instance $r, c \geq 5, A, D, k$
Output: A valid arrangement $\mathcal{A}$ or **Failure**

1 $\mathcal{P} \leftarrow [(C_1, C_2), (C_5, C_6), \ldots, (C_{c-1}, C_c), (C_3, C_4)]$;
2 **foreach** *pair* (L, R) *in* $\mathcal{P}$ **do**
3 $X \leftarrow$ iterator over unassigned loads in D starting from $k + 2$;
4 $Y \leftarrow$ iterator over unassigned loads in A^R;
5 bottom cell of $L \leftarrow$ first unassigned load from Y;
6 bottom cell of $R \leftarrow$ smallest unassigned load;
7 **while** L *and* R *not full* **do**
8 Advance $x \in X$;
9 **if** X *is exhausted* **then return Failure**;
10 **if** x *is not* D*-valid when assigned to* R **then continue** ;
11 **for** y *in* Y **do**
12 **if** $y = x$ **then continue** ;
13 **if** x *and* y *are valid when* x *and* y *assigned to* R, L *respectively* **then**
14 Assign x, y to R, L respectively;
15 **break**;
16 **return** resulting arrangement $\mathcal{A}$;

Assigning the last loads. To increase the likelihood of the assignments succeeding for last loads assigned, we leave special column(s) empty for them, to be filled after all other columns. If c is odd, we leave C_3 (the third column) as the last empty column, with the remaining loads assigned in the order in which they appear in D. Otherwise, we use C_3 and C_4 as the last column pair to be filled. By leaving C_3 and possibly C_4 as the last columns, we use C_2 and C_4 (or C_5), which contain early appearing loads in D and A^R, for potential horizontal adjacencies.

Since we always check whether loads are valid before assigning them, any returned arrangement is valid. However, whenever a failed is returned, it does not mean that there is no valid arrangement. Therefore, we present an enhancement that considers more potential arrangements.

Load-skipping enhancement

We introduce an enhancement to Algorithm 1 to increase its success rate. A valid arrangement only requires that the first load in A^R is in the bottom row, while freedom exists for other loads. We exploit this by considering all options for the choice of the first load of A^R assigned to the left column. That is, for the first column pair filled, we set the iterator Y to start at an offset, initializing the arrival adjacency chain from the middle of A^R. For the remaining column pairs, we proceed as normal. The enhanced version tries all possible offsets from 0 to $n - r$, which one can run in parallel.

[1] As a speed up, when not filling the last column pair, **break** from the inner loop instead to examine the next load for x. This is done because y must appear before x in A^R for x to satisfy the adjacency for A^R. If $y = x$, there is no point considering later loads in A^R.

Solving the 2D Grid Relocation Problem

R-StoRMR may require relocations as k increases. This section addresses the retrieval phase where, given an initial arrangement $\mathcal{A}$, the goal is to minimize relocations during retrieval. Already for stack-based storage and a known retrieval order, the problem is NP-hard (Caserta, Schwarze, and Voß 2012b). Our setting is more involved if a load to be retrieved is blocked by other loads as (i) there are multiple retrieval paths (that determine the loads to relocate), and (ii) there are more options for where to relocate.

Given the above and so as to minimize I/O row usage, we impose the following constraints: The I/O row must be empty after each load is retrieved, i.e., all relocated loads must end up in W. Furthermore, when loads on the I/O row are stored back in W, simply return them to their original cells in W, as in Figure 1 (d)-(f). This choice reflects that we choose $\mathcal{A}$ to enable robust relocation-free retrieval according to D.

Relocation procedure. Let x denote the *target load* to be retrieved. Compute a retrieval path π from x to the I/O row that minimizes blocking loads, breaking ties by path length. If π does not pass through other loads, x can be retrieved. Otherwise, the blocking loads along π, i.e., the *blockers*, must be relocated, starting from the outermost and proceeding inward. Relocate each blocker b in a greedy manner, assigning it to a favorable empty cell that is not on π (keeping π clear for x to be retrieved). See Figure 4 for an example.

First consider the case where b has reachable empty cells in W. In this case, aim to assign b to a destination cell from which b will not be relocated again during a subsequent retrieval. To this end, compute a set U of *unblocked* loads, which are loads that have direct access to the I/O row. For each candidate assignment of b to a destination cell, check whether a load in U becomes blocked. A load y is considered blocked, if it departs before all of its adjacent loads as well as b, per D, and has no direct path to the I/O row. If there is a destination cell that does not make a load of U blocked, assign b to it, preferring the closest such cell. Otherwise, assign b to the closest empty cell. In both cases, ensure that we assign b to a cell that leaves sufficiently many reachable empty cells in W for the remaining blockers still on π.

Alternatively, there are no empty cells in W for relocations and the remaining blockers are relocated to the I/O row. These blockers will be relocated back to W after x is retrieved. All relocations in this case use π to go to/from the I/O row. Assuming $r \leq c$, the number of cells on the I/O row suffices.

Experimental Evaluation

The objective of the experimental evaluation is to measure the improvement achieved by the proposed storage and retrieval strategies over baselines for total number of relocations, I/O-row usage, and total load distance traveled in square grids. Since the theoretical analysis establishes the ratio k/c as a key parameter affecting relocations in R-StoRMR, four values of the ratio are considered, i.e., k values of $0.25c, 0.5c, 0.75c, c$ per grid size with c columns. We run 50 trials for each combination of grid side length and k, always at 100% density. Each trial randomly generates A, which is given to the storage algorithm, and then draws a k-bounded perturbation of $(1, 2, \ldots, n)$, which is revealed one at a time during retrieval.

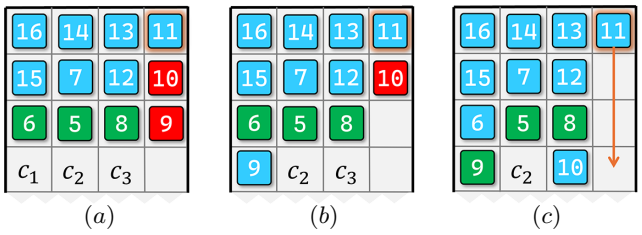


Figure 4: Example relocations: (a) Load 11 is to be retrieved. π is a straight downward path and 9, 10 are blockers (red). To relocate 9, we compute the set of unblocked loads $U = \{5, 6, 8\}$ (green). Destination cell c_3 is not chosen because it would disconnect 10 from empty cells and c_2 is also not chosen as placing 9 there would block 5. (b) 9 is relocated to c_1 which keeps 6 unblocked due to its adjacency to 5. (c) U is recomputed and 10 is relocated to c_3, as it does not block any loads.

We compare the following algorithm variants for storage:

- BaseS: store for 0 relocations assuming a known departure sequence per prior work (Geft, Bekris, and Yu 2025).
- **RobustS**: The proposed improved algorithm for a k-robust arrangement. If this algorithm does not find a k-robust arrangement (not always guaranteed to exist), decrement k until one is found. For $k = 0$, BaseS is used as a fallback, though this was not frequently observed in experiments.

We compare the following algorithm variants for retrieval:

- BaseR: We find a retrieval path π that contains the fewest blocking loads, breaking ties by path length. When the target load x is blocked, relocate the blocking loads to the I/O row, retrieve x, and place the blocking loads back in their original cells. Loads can be moved using π.
- **ImpR**: Our improved relocation algorithm.

Results

We present results using Python on an Apple M3 with macOS 15.5 for combinations of the above variants for storage and retrieval. Numerical results for the number of relocations and I/O row usage are presented in Figure 5. We also plot average results across all k values for varying grid sizes, and separately across k for a fixed grid size in Figure 6. This figure includes *distance suboptimality (subopt)* plots, defined as total grid path length traveled by loads minus a lower bound accounting for a fully packed grid, given by $cr(r+1)$, i.e., excess over the optimum.

Experiments show that the combined (**RobustS + ImpR**) outperform the baseline and variants where only the storage or retrieval is improved. When k is at most half the grid width ($k/c \leq 0.5$), **RobustS** nearly eliminates rearrangements. These results are in line with Theorem 3, which indicates that relocations are unavoidable when k approaches half the grid width. For larger k, relocations are reduced by up to 60-70%. For every grid size, **RobustS** and **ImpR** run in under 1 min. and 1 sec., respectively; see Figure 7.

In nearly all cases, loads relocated within W are not relocated again, indicating that we avoid cascading relocations.

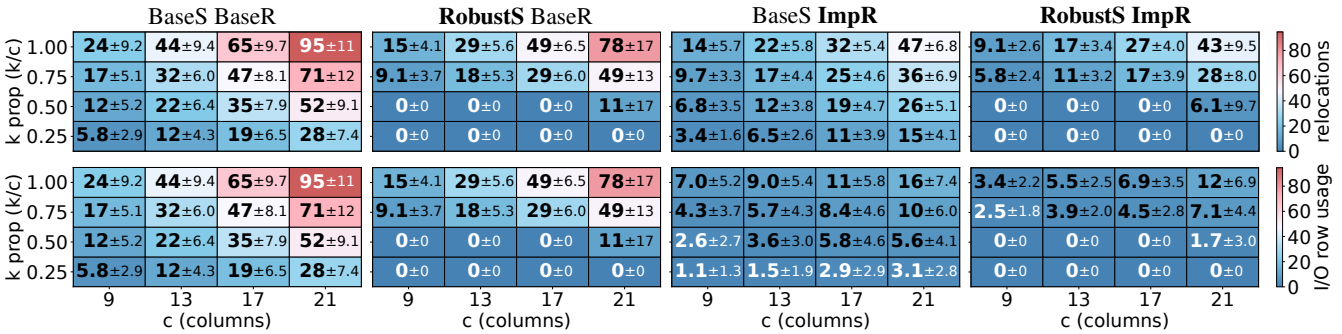
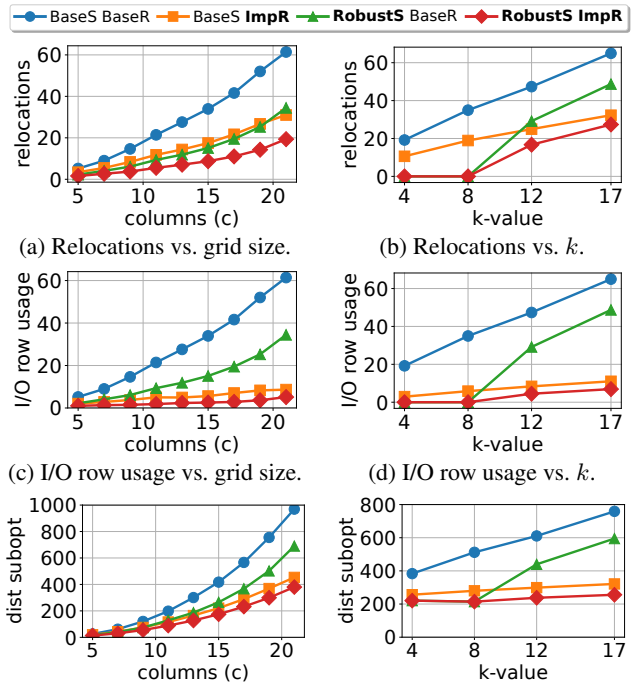


Figure 5: Mean relocations ± st. dev. (top) and mean I/O row usage ± st. dev. (bottom) for varying grids and k values, comparing storage and retrieval algorithms. The heatmaps are the same in the two leftmost columns since BaseR always uses the I/O row.

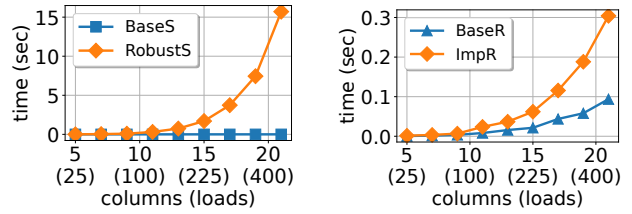


(a) Relocations vs. grid size. (b) Relocations vs. k.

(c) I/O row usage vs. grid size. (d) I/O row usage vs. k.

(e) Distance subopt. vs. grid size. (f) Distance subopt. vs. k.

Figure 6: Relocations (top), I/O row usage (middle), and distance subopt. (bottom) for varying grid sizes and k values. In (a)(c)(e) we average across all four k's per grid size. In (b)(d)(f) we fix a 17×17 grid. Each point averages 50 trials.



(a) Storage running times. (b) Retrieval running times.

Figure 7: Run-times averaged across k values per grid size.

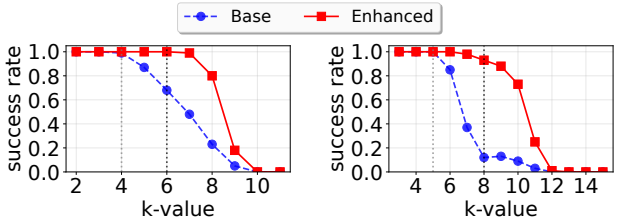


Figure 8: Ablation showing the success rate of **RobustS** with and without the load-skipping enhancement on a 15×15 (left) and 19×19 (right) grids (100 trials per data point). The enhanced **RobustS** achieves 80%+ success even for higher k than the theoretical limits of 6 and 8 respectively (thick vertical dashed line) beyond which relocations can occur.

Qualitatively, relocation choices are typically straightforward: early on, blockers are moved to the I/O row due to limited space; later, as the grid empties, accessible cells simplify blocker placement decisions. While **ImpR** significantly reduces relocations to the I/O row, **RobustS** provides a further reduction, up to their elimination for $k/c \leq 0.5$.

Lastly, we show the improvement of **RobustS** due to the enhancement; see Figure 8.

Conclusion

This study expands the applicability of grid-based storage and retrieval by introducing uncertainty to this setting through novel theoretical and empirical results. It establishes design guidelines by relating the grid's opening width to the feasibility of robust zero-relocation solutions. Empirically, our storage and retrieval approach significantly reduces relocations for storage at full capacity and confines most relocations to within the grid.

This work opens directions for further investigation: Is it possible to determine in polynomial time whether a robust arrangement exists for a given k (strengthening the heuristic approach)? Can one always find a robust arrangement with $2k + 3$ columns, closing the gap between our bounds? One could also consider the multi-robot case and MAPF reasoning in our setting.

Acknowledgements

We thank the reviewers and editorial staff for their insightful suggestions. We thank Gur Lifshitz for useful discussions. 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.; Goerigk, M.; and Knust, S. 2020. Robust optimization for premarshalling with uncertain priority classes. *European Journal of Operational Research*, 287(1): 191–210.

Boschma, R.; Mes, M. R. K.; and de Vries, L. R. 2023. Approximate dynamic programming for container stacking. *European Journal of Operational Research*, 310(1): 328–342.

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. 2012a. A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research*, 219(1): 96–104.

Caserta, M.; Schwarze, S.; and Voß, S. 2012b. A mathematical formulation and complexity considerations for the blocks relocation problem. *Eur. J. Oper. Res.*, 219(1): 96–104.

Disselnmeyer, M.; Bömer, T.; Pfrommer, J.; and Meyer, A. 2024. The Static Buffer Reshuffling and Retrieval Problem for Autonomous Mobile Robots. In *ICCL*, volume 15168 of *Lecture Notes in Computer Science*, 18–33. Springer.

Galle, V.; Manshadi, V. H.; Boroujeni, S. B.; Barnhart, C.; and Jaillet, P. 2018. The Stochastic Container Relocation Problem. *Transportation Science*, 52(5): 1035–1058.

Geft, T.; Bekris, K. E.; and Yu, J. 2025. Fully Packed and Ready to Go: High-Density, Rearrangement-Free, Grid-Based Storage and Retrieval. *CoRR*, abs/2505.22497.

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*.

Hanou, I. K.; de Weerdt, M. M.; and Mulderij, J. 2023. Moving Trains like Pebbles: A Feasibility Study on Tree Yards. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, Prague, Czech Republic, July 8-13, 2023*, 482–490. AAAI Press.

He, J.; Liu, X.; Duan, Q.; Chan, W. K. V.; and Qi, M. 2023. Reinforcement learning for multi-item retrieval in the puzzle-based storage system. *European Journal of Operational Research*, 305(2): 820–837.

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.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. S.; and Koenig, S. 2021. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11272–11281.

Mirzaei, M.; Koster, R. B. M. D.; and Zaerpour, N. 2017. Modelling load retrievals in puzzle-based storage systems. *International Journal of Production Research*, 55(22): 6423–6435.

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.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *SOCS*, 151–159. AAAI Press.

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.