Cube-Based Automated Storage and Retrieval Systems: A Multi-Agent Path Finding Approach

Yi Zheng[1], **Yimin Tang**[1], **T. K. Satish Kumar**[1], **Sven Koenig**[2]
[1]University of Southern California
[2]University of California, Irvine
yzheng63@usc.edu, yimintan@usc.edu, tkskwork@gmail.com, sven.koenig@uci.edu

Abstract

Automated Storage and Retrieval Systems (AS/RS) require the coordination of multiple agents in a common warehouse environment. They often rely on Multi-Agent Path Finding (MAPF) technologies for the path-coordination of agents that transport inventory pods. In this paper, we study the more recent cube-based AS/RS and the relevance of MAPF technologies to them. In these systems, storage bins are stacked on top of each other in a tight 3D grid environment without aisles. Robots operate only at the top level to retrieve and organize the storage bins. While cube-based AS/RS maximize the storage density, they require the robots to "dig" out the storage bins whenever necessary. Despite their advantages and recent adoption, limited research has been reported on the relevance of MAPF technologies to them. In this paper, we adopt novel MAPF perspectives on the cube-based AS/RS and propose two approaches for planning in such domains. In the first approach, the robots are treated as the agents: Plans are generated for them using 'move' and 'dig' actions. In the second approach, there are two phases. In the first phase, the storage bins are treated as the agents: Plans are generated for them to move autonomously. In the second phase, the robots are treated as the agents: Plans are generated for them to chaperone the movements of the storage bins. We demonstrate the value of MAPF technologies in cube-based AS/RS and experimentally show that the second approach marginally outperforms the first.

1 Introduction

The automation of warehouses is a large multi-billion dollar industry currently led by companies such as Amazon Robotics and Alibaba (Salzman and Stern 2020). Automated warehouses are often formally referred to as Automated Storage and Retrieval Systems (AS/RS). In such systems, multiple agents—typically robots—transport inventory pods between their storage locations and various workstations where they are required. These agents operate in a common environment and are required to avoid conflicts with each other while planning their paths.

For the required path coordination, AS/RS rely on Multi-Agent Path Finding (MAPF) technologies. Specifically, they invoke MAPF solvers to plan collision-free paths for all the agents while optimizing one or more performance metrics. In fact, the end-to-end performance metrics of AS/RS are often well-aligned with the performance metrics for which the MAPF solvers are optimized. Moreover, modern MAPF solvers encapsulate powerful algorithmic techniques and can scale up to realistic AS/RS planning problems with hundreds of agents.

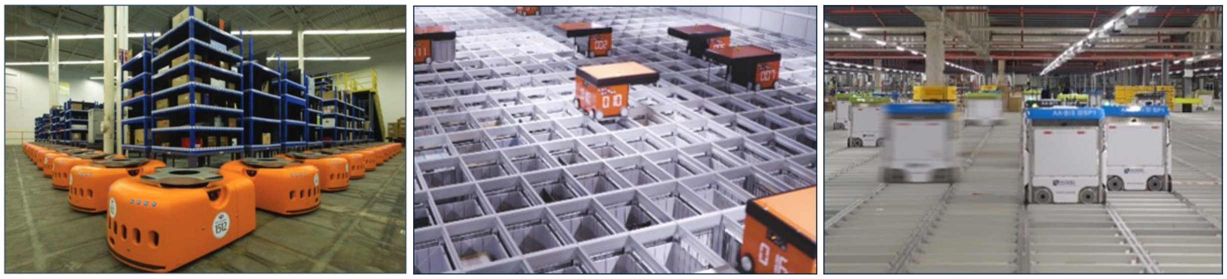
Many existing AS/RS operate in an environment that can be modeled as a 2D grid map with aisles between the storage locations of the inventory pods. Thus, such systems can be expanded by increasing the floor space and the number of agents. However, in such systems, it is not easy to increase the storage density, that is, the amount of inventory per unit area of the floor space. Figure 1a shows an example via a schematic of the Kiva system (Wurman, D'Andrea, and Mountz 2008). Here, increasing the storage density is prohibitive because: (a) Decreasing the space reserved for the aisles can cause more conflicts and congestion among the robots that transport the inventory pods; and (b) Vertically expanding the shelf space in an inventory pod can make it unstable and/or overload the robot that carries it.[1]

More recently, cube-based AS/RS have been developed to improve the storage density in automated warehouses (Trost, Kartnig, and Eder 2023; Trost and Eder 2024). Figures 1b and 1c show two examples of cube-based AS/RS: AutoStore and Ocado, respectively. Unlike the Kiva-style AS/RS, cube-based AS/RS employ a 3D grid space for storage bins. The storage bins are stacked directly on top of each other, in an attempt to maximize the storage density. The robots operate on a 2D grid above the storage area to retrieve and deliver the storage bins to workstations, whenever necessary.

The cube-based AS/RS have several significant advantages over Kiva-style AS/RS. One such advantage is that the cube-based AS/RS have a higher storage density. Another advantage is that they are easily extensible and capable of utilizing available space of any shape within a building. A third advantage is that the robots operate on a 2D grid above the storage area, which is primarily open space without obstacles or narrow aisles.

The above advantages of the cube-based AS/RS incur certain bin "digging" costs. Bin digging refers to the following two-fold complexity. First, a robot is required to lift (lower)

[1]Adding a second or third floor to a warehouse may not increase its land space but still increases its floor space.



(a) Kiva (b) AutoStore (c) Ocado

Figure 1: Shows the schematics of different types of AS/RS. (a), borrowed from (Wurman, D'Andrea, and Mountz 2008), shows a Kiva-style system, in which robots transport inventory pods on a 2D grid map with aisles. (b) and (c), borrowed from https://commons.wikimedia.org/wiki/File:Euro-friwa-autostore.jpg and https://commons.wikimedia.org/wiki/File:Ocado_warehouse_bots.jpg, respectively, show the AutoStore and the Ocado cube-based AS/RS, respectively, in which the storage bins are stacked on top of each other and the robots operate only above the storage areas.

a storage bin from (onto) the stack beneath it. Hence, the robots have to be equipped with special apparatuses for not only carrying but also lifting (lowering) storage bins (albeit one at a time). Each lift (lower) operation can be expensive, both in terms of time and energy consumption. Second, the storage bins that are stacked on top of a target storage bin have to be relocated from that stack to clear the path for the target storage bin.

Overall, the advantages of cube-based AS/RS outweigh the costs of the bin digging problem incurred by them. In fact, their recent adoption by companies, such as AutoStore and Ocado, is testimony to this cost-benefit analysis. Moreover, in the near future, cube-based AS/RS are expected to become cheaper and increasingly popular for warehouse automation.

While MAPF technologies have played a critical role in the success of Kiva-style AS/RS, limited research has been reported on their relevance to cube-based AS/RS[2]. In Kiva-style AS/RS, MAPF technologies have been used to address the one-shot MAPF problem (Cohen et al. 2016; Okumura 2023), the lifelong MAPF problem (Ma et al. 2019b; Li et al. 2021b; Varambally, Li, and Koenig 2022), and the warehouse layout design problem (Zhang et al. 2023).

In this paper, we explore the applicability of MAPF technologies to cube-based AS/RS in anticipation of a significant long-term impact similar to that in Kiva-style AS/RS. However, as a first milestone, we study a version of the coordination problem in which all tasks are known a priori. We use two novel perspectives, each of which leads to a different approach, but both of which are based on the Conflict-Based Search (CBS) framework (Sharon et al. 2015). In the first approach, the robots are treated as the agents: Plans are generated for them using 'move' and 'dig' actions. In the second approach, there are two phases. In the first phase, the storage bins are treated as the agents: Plans are generated for them to move autonomously. In the second phase, the robots are treated as the agents: Plans are generated for them to chaperone the movements of the storage bins.

The second approach initially treats the storage bins as the agents and decouples them from the robots. Despite this initial decoupling, we intuitively expect the second approach to outperform the first approach for the following reason: In the cube-based AS/RS, the storage bins are tightly packed in a 3D grid space, while the robots operate on an open 2D grid above the storage area. Consequently, the tendency of storage bins to block each other is more than the tendency of robots to block each other. Therefore, the coordination of the storage bins is more critical than the coordination of the robots and should be addressed as early as possible. The second approach indeed follows this strategy.

While both of our approaches are based on the CBS framework, they require careful generalizations and adaptations to cube-based AS/RS. This is because of the unique challenges of cube-based AS/RS that primarily stem from the added complexity of bin digging. We describe how to address the unique challenges of cube-based AS/RS in both our approaches. We also provide a comprehensive set of experimental results that compares the two approaches. In general, we observe that the second approach marginally outperforms the first in terms of efficiency and success rate. Our results conform to the intuition described above.

2 Background

In this section, we describe the background material that is relevant to the rest of the paper.

2.1 Multi-Agent Path Finding

The classic MAPF problem is specified by an undirected unweighted graph $G = (V, E)$ and a set of k agents $\{a_1, a_2 \dots a_k\}$, where a_i is required to move from a start vertex $s_i \in V$ to a goal vertex $g_i \in V$. Time is discretized into timesteps. At each timestep, each agent can either move to an adjacent vertex or wait at its current vertex, both with unit cost. A path of a_i is a sequence of move and wait actions that lead a_i from s_i to g_i. A vertex conflict (a_i, a_j, v, t) occurs when two different agents a_i and a_j are at the same

[2]An existing master's thesis (Djupesland 2023) claims to extend the relevance and use of reinforcement learning in MAPF to cube-based AS/RS. However, we do not have access to this thesis since it will not be made publicly available until May 2026.

vertex v at timestep t. An edge conflict (a_i, a_j, u, v, t) occurs when two different agents a_i and a_j traverse the same edge (u, v) in opposite directions between timesteps t and $t + 1$. A solution to a MAPF problem is a set of paths, one for each agent, without conflicts. The quality of the solution is measured by the sum of costs (the sum of the numbers of timesteps required by every agent to reach its goal vertex) or the makespan (the number of timesteps within which all agents reach their goal vertices).

The MAPF problem defined above is commonly referred to as the one-shot MAPF problem: Each agent is required to reach a single goal location from its start location and stay there. The one-shot MAPF problem has many practical applications, such as in automated warehousing (Wurman, D'Andrea, and Mountz 2008), virtual network embedding (Zheng et al. 2022, 2023), railway planning (Li et al. 2021a) and multi-drone delivery (Choudhury et al. 2020). Another variant of the MAPF problem is the lifelong MAPF problem: Continually, each agent is assigned a new goal location after it reaches the previously assigned goal location (Ma et al. 2019b). The lifelong MAPF problem is relevant in domains where agents are required to accomplish a queue of tasks, such as in Kiva-style AS/RS. A survey of common MAPF variants, objectives, and benchmarks can be found in (Stern et al. 2019).

The one-shot MAPF problem is NP-hard to solve optimally for various objective functions (Yu and LaValle 2013). However, many kinds of solvers have been developed for solving it efficiently in practice. These include CBS (Sharon et al. 2015), improved versions of CBS (Cohen et al. 2016), Priority-Based Search (Ma et al. 2019a), and LaCAM (Okumura 2023). Many kinds of solvers have also been developed for solving the lifelong MAPF problem. These include two categories of solvers: those that assume that all goal locations are known a priori and those that do not. In the first category, solvers have been developed based on Answer Set Programming (Nguyen et al. 2019) and Conflict-Based Steiner Search (Ren, Rathinam, and Choset 2023). In the second category, the solvers include Token Passing with Task Swaps (Ma et al. 2017), Rolling-Horizon Collision Resolution (Li et al. 2021b), and Priority Inheritance with Back Tracking (Chen et al. 2024). In general, lifelong MAPF solvers require repeated replanning and an intelligent outer-loop task assigner to reduce the idle time of the agents.

2.2 Cube-Based AS/RS

Cube-based AS/RS, exemplified by AutoStore and Ocado's grid-based automated warehouses, organize inventory in a dense 3D grid of storage bins. The inventory items are stored in storage bins stacked vertically in columns within the grid structure. There are no internal aisles in the grid: Instead, multiple robots move on an open 2D grid above the storage area to retrieve the storage bins from the stacks beneath them and deliver the storage bins to the workstations. The robots should avoid collisions with each other.

To retrieve an item, a robot positions itself over the stack of the target storage bin and lowers a lifting apparatus to pull it up. Then the robot transports this storage bin to a workstation. However, before the robot pulls up the target storage bin, each storage bin on top of the target storage bin also has to be pulled up by the same robot or by any other robot. Each such storage bin also has to be relocated to another available stack. Hence, cube-based AS/RS introduces bin digging as a new complexity to the coordination problem. After being processed at a workstation, a storage bin is delivered back to an available stack in the grid by an available robot. In many cases, the robot that brings a storage bin to a workstation also delivers it back to an available stack on the grid after waiting at the workstation for it to be processed.

In comparison to Kiva-style AS/RS, cube-based AS/RS eliminate unused aisle space and maximize storage density (Trost, Kartnig, and Eder 2023). Their design and performance have been studied in (Trost, Kartnig, and Eder 2023) via simulation of an AutoStore-style system. Cube-based AS/RS have also been studied via several other system simulations (Beckschäfer et al. 2017; Franke and Schuderer 2021; Chen, Yang, and Shao 2022). Some research has also focused on attaining the optimal stack height and/or length-to-weight ratio (Zou, de Koster, and Xu 2016), the optimal storage locations of items (Beckschäfer et al. 2017), and the optimal sequencing of delivery tasks to the workstations (Ko and Han 2022). Another work has utilized the single-agent pathfinding algorithm D^* Lite in cube-base AS/RS (Tan-Thuan Banh 2024): However, this approach only exercises the local repair of individual robot paths. A comprehensive survey of cube-based AS/RS is presented in (Trost and Eder 2024).[3]

2.3 Conflict-Based Search

CBS is a two-level heuristic search framework that has been used for solving the MAPF problem optimally (Sharon et al. 2015). On the high level, CBS performs a best-first search on a Constraint Tree (CT). Each CT node N contains a set of spatiotemporal constraints $N.constraints$ that are used to coordinate agents to avoid conflicts. CT node N has a set of paths $N.paths$, one for each agent, that respects $N.constraints$. The cost of CT node N is the sum of the costs of the paths in $N.paths$. The root CT node contains an empty set of constraints and a set of shortest paths that may contain conflicts. When CBS expands a CT node N, it first checks for conflicts between every pair of paths in $N.paths$. If there are none, the CT node is a goal CT node and CBS returns the paths as the solution. Otherwise, CBS chooses one of the conflicts and resolves it by splitting it into two child CT nodes, each with an additional constraint prohibiting one agent from the conflict from using the conflicting vertex or conflicting edge at the conflicting timestep. CBS then uses its low-level search, such as A^*, to replan the path of the agent to satisfy the new constraint. The fewer conflicts there are to resolve, the faster CBS terminates. A complexity analysis of CBS is presented in (Gordon, Filmus, and Salzman 2021). CBS guarantees completeness by eventually exploring both ways of resolving each conflict and optimality

[3]3D warehouses differ from cube-based AS/RS in that they admit aisle space and are amenable to regular MAPF algorithms (Wang et al. 2024). However, unlike Kiva-style AS/RS, the 3D warehouses can have both horizontal and vertical aisles.

of the generated solution by performing best-first search on both its high and low levels.

However, since solving the MAPF problem optimally is hard, suboptimal solution procedures can be investigated to increase the runtime efficiency. Enhanced CBS (ECBS) has been developed to produce suboptimal solutions in the CBS framework by trading the solution cost for runtime (Barer et al. 2014). ECBS utilizes the power of a bounded-suboptimal search algorithm called focal search.

Focal search maintains two lists of nodes *OPEN* and *FOCAL*. *OPEN* is the regular open list, as in A^*, whose nodes n are sorted by an admissible cost function $f(n) = g(n) + h(n)$ where $h(n)$ is an admissible heuristic function. Let $w > 1$ be a user-specified suboptimality factor and $f_{min} = \min_{n_i \in \mathit{OPEN}} f(n_i)$ be the minimum f-value in *OPEN*. *FOCAL* contains the nodes n in *OPEN* for which $f(n) \leq w \cdot f_{min}$, sorted by a secondary heuristic function $d(n)$ that estimates the number of hops from node n to a goal node. $d(n)$ can be inadmissible. Unlike A^*, focal search always expands a node n with the minimum d-value in *FOCAL*. Let C^* be the cost of the optimal solution. Focal search guarantees that the cost of the returned solution is at most $w \cdot C^*$ since f_{min} is a lower bound on C^*.

ECBS is a bounded-suboptimal variant of CBS whose high-level and low-level searches are both focal searches. Both these searches use measures related to the number of conflicts as the secondary heuristic function. ECBS(w) refers to ECBS with the user-specified factor w to be used in its focal searches. ECBS(w) is w-suboptimal since it guarantees a solution—if one exists—with a cost that is no larger than $w \cdot C^*$ (Barer et al. 2014; Cohen et al. 2016). Thus, ECBS(w) with a reasonably small value of w has the flexibility of expanding CT nodes with fewer conflicts than the CT nodes chosen for expansion by CBS. This often makes ECBS(w) faster than CBS.

3 The Cube-Based AS/RS Coordination Problem

In this section, we formulate the cornerstone coordination problem of cube-based AS/RS. While many aspects of it resemble the MAPF problem, the bin digging aspect of it introduces a new complexity.

3.1 The Cube-Based AS/RS Environment

A cube-based AS/RS environment can be viewed as an undirected 3D grid graph $G = (V, E)$. V represents the union of the set of possible storage locations of the storage bins and the set of possible locations of the robots that operate on top of the storage area. Each vertex $v \in V$ is labeled by a tuple of 3D coordinates (x_v, y_v, z_v): x_v and y_v are the coordinates projected on the XY-plane on which the storage area stands; z_v is the depth measured from the top of the storage area. Hence, the robots operate on the 2D vertex-induced subgraph with $z_v = 0$. E is a set of edges that represents the union of the set of possible movements of the storage bins and the set of possible movements of the robots to adjacent locations. Hence, $(v_i, v_j) \in E$ if and only if

$$(x_{v_i} = x_{v_j}) \wedge (y_{v_i} = y_{v_j}) \wedge (|z_{v_i} - z_{v_j}| = 1) \quad \text{or}$$
$$(x_{v_i} = x_{v_j}) \wedge (|y_{v_i} - y_{v_j}| = 1) \wedge (z_{v_i} = z_{v_j} = 0) \quad \text{or}$$
$$(|x_{v_i} - x_{v_j}| = 1) \wedge (y_{v_i} = y_{v_j}) \wedge (z_{v_i} = z_{v_j} = 0).$$

The robots and workstations operate at the top level of the warehouse, that is, at $z = 0$. The storage bins are at $z = 0$ while they are being carried by the robots or are being processed at the workstations. Otherwise, they are at $z > 0$. For the XY-plane on which the storage area stands, let the height of the storage column at the location (x, y) be denoted by $h_{(x,y)}$. Hence, for any vertex v, $z_v \in \{0, 1 \ldots h_{(x_v,y_v)}\}$.

3.2 Robots, Storage Bins, and Workstations

In addition to the graph $G = (V, E)$, cube-based AS/RS are characterized by a set of robots $R = \{r_1, r_2 \ldots r_k\}$, a set of storage bins $B = \{b_1, b_2 \ldots b_m\}$, and a set of workstations $P = \{p_1, p_2 \ldots p_l\}$. For simplicity, time is discretized into logical timesteps. At any timestep, we use $loc(\cdot)$ to denote the location of a robot, storage bin, or a workstation.

The robots are the only truly autonomous agents in the system. Each robot can occupy any location $(x_v, y_v, 0)$ on top of the storage area, where it can execute one of several actions. First, it can lift and hold a storage bin that is on top of the stack directly beneath it. If the storage bin is at depth $1 \leq d \leq h_{(x_v,y_v)}$, the robot needs to lower its apparatus from depth 0 to depth d and lift the storage bin from depth d to depth 0, for a total of $2d$ timesteps. Second, it can lower a storage bin that it holds onto the top of the stack directly beneath it. If the storage bin is eventually placed at depth $1 \leq d \leq h_{(x_v,y_v)}$, the robot needs to lower the storage bin from depth 0 to depth d and lift the apparatus from depth d to depth 0, for a total of $2d$ timesteps. Third, the robot can move to an adjacent location that is also at $z = 0$, with or without holding a storage bin. The robot expends one timestep for doing so. Fourth, the robot can wait at its current location for one timestep, with or without holding a storage bin. A robot can hold only one storage bin at a time. A robot that holds a storage bin occupies the same space as a robot that does not. When robots move on the top level, they have to avoid vertex conflicts and edge conflicts with each other.

The storage bins cannot move autonomously: They have to be chaperoned by the robots. Each storage bin can occupy any location (x_v, y_v, z_v), where $1 \leq z_v \leq h_{(x_v,y_v)}$. However, a storage bin is subject to gravity and can only be either on top of another storage bin or at $(x_v, y_v, h_{(x_v,y_v)})$, that is, at the bottom of the storage column. Consequently, a robot can only place a storage bin in such a location without dropping it, that is, the robot cannot drop the storage bin to a midair location inside a storage column. A storage bin that is not on top of its stack cannot be retrieved without first relocating the storage bins that are on top of it. No two storage bins can be at the same location at the same timestep.

Workstations are processing units, each of which occupies a single grid cell on the top level of the storage area. Hence, the robots can access them easily since they are at the same level. The robots are required to transport target storage bins from the storage area to the workstations for inventory picking operations. For simplicity, we assume that a robot arrives at a workstation with a storage bin, waits there for one timestep for the picking operation to be executed on the storage bin, and then returns the storage bin to the storage area.

3.3 Problem Instance

A problem instance specifies the values of k, m, and l. It also specifies the start location s_i of each robot $r_i \in R$, the target storage bins $B_{tar} \subseteq B$, the location $loc(p_i)$ of each workstation $p_i \in P$, and the destination workstation $g_i^P \in \{loc(p_1), loc(p_2) \dots loc(p_l)\}$ for each $b_i \in B_{tar}$. The tuple (b_i, g_i^P) is referred to as a *delivery task*. A solution satisfies all the constraints stated in Sections 3.1 and 3.2. It consists of k action sequences $\{A_1, A_2 \dots A_k\}$: Each A_i assigns an action for robot r_i at each timestep. Collectively, the action sequences of the robots transport the target storage bins from their start locations to their destination workstations and then return them back to the storage area. The objective can be to minimize either the sum of the numbers of timesteps taken by all the robots (sum of costs) or the maximum of these numbers (makespan). In a lifelong version of the problem, there is usually the need for an outer loop that intelligently specifies where a storage bin should be placed after it is processed at a workstation. However, for easy exposition, we assume that the storage bin is returned to the nearest available stack.

4 Planning Framework and Approaches

We now present two preliminary approaches for solving the cube-based AS/RS problem formalized in Section 3. Both approaches are based on adapting the ECBS framework (Barer et al. 2014). The first approach, which we refer to as the robot-centric approach, treats the robots as the agents. The second approach, which we refer to as the bin-centric approach, takes a different perspective and treats the storage bins as the agents in its first phase and the robots as the agents in its second phase: The plan generated for the robots in the second phase chaperones the plan generated for the storage bins in the first phase.

Both approaches use the notions of a relocation task and a retrieval task. A *relocation task* is of the form (b, g), indicating that the storage bin $b \in B$ needs to be moved from its current location to the goal location g. g specifies the XY-coordinates of a storage column. A relocation task is not specified in the problem instance but is generated by our algorithms internally. A *retrieval task* is of the form (b_i, g_i^P, g), indicating that the storage bin $b_i \in B_{tar}$ needs to be moved from its current location to the workstation location g_i^P and later to the goal location g by the same robot. g specifies the XY-coordinates of a storage column. While b_i and g_i^P are specified in the problem instance as the delivery task (b_i, g_i^P), g is generated by our algorithms internally.

4.1 Approach 1: Robot-Centric Planning

Algorithm 1 shows the pseudocode for the robot-centric planning approach. It takes as input the environment graph

Algorithm 1: Robot-Centric Planner

```
Input: $G$, $R$, $B$, $B_{tar}$, $P$, $del\_tasks$
Output: a sequence of actions for each robot
Parameter: suboptimality factor $w$
 1: Sort elements in $del\_tasks$ in increasing order of the depth of
    their storage bins. (Break ties using the index order.)
 2: for $r_i \in R$ do
 3:     $TQ_i \leftarrow \{\}$
 4: end for
 5: $expanded\_tasks \leftarrow \emptyset$
 6: $precedences \leftarrow \emptyset$
 7: $system\_copy \leftarrow$ locations of all storage bins.
 8: while $del\_tasks \neq \emptyset$ do
 9:     Pop the first element $(b_i, g_i^P)$ from $del\_tasks$.
10:     $decomp \leftarrow \emptyset$.
11:     Let $on\_top$ be the set of storage bins on top of $b_i$ in increasing order of depth.
12:     while $on\_top \neq \emptyset$ do
13:         Pop the first element $b_j$ from $on\_top$.
14:         $g_j \leftarrow$ a nearby storage column, preferably avoiding the
            storage columns of the storage bins in $del\_tasks$.
15:         Append the relocation task $(b_j, g_j)$ to $decomp$.
16:         Update $system\_copy$ with the new location of $b_j$ set to
            $g_j$.
17:     end while
18:     $g_i \leftarrow$ a nearby storage column, preferably avoiding the storage columns of the storage bins in $del\_tasks$.
19:     Append the retrieval task $(b_i, g_i^P, g_i)$ to $decomp$.
20:     Append $decomp$ to $expanded\_tasks$.
21:     Update $system\_copy$ with the new location of $b_i$ set to $g_i$.
22: end while
23: for $d \in expanded\_tasks$ do
24:     Add $\tau \prec \tau'$ to $precedences$ if task $\tau$ is mentioned before
        task $\tau'$ in $d$.
25: end for
26: for $d_i, d_j \in expanded\_tasks$ do
27:     Add $\tau \prec \tau'$ to $precedences$ if $\tau$ and $\tau'$ mention the same
        storage bin or $\tau'$ places a storage bin at the location of $\tau$'s
        storage bin, with $\tau \in d_i$, $\tau' \in d_j$, and $i < j$.
28: end for
29: while $expanded\_tasks$ (flattened) $\neq \emptyset$ do
30:     Choose $\tau \in expanded\_tasks$ such that no $\tau' \in$
        $expanded\_tasks$ yields $\tau' \prec \tau$.
31:     Let $r_i$ be the robot that can start doing $\tau$ at the earliest
        timestep after completing the tasks in $TQ_i$.
32:     Append $\tau$ to $TQ_i$ and remove it from $expanded\_tasks$.
33: end while
34: return $ECBS_R(G, R, \{TQ_1, TQ_2 \dots TQ_k\},$
    $precedences, w)$.
```

G, the set of robots R with their start locations, the set of storage bins B with their start locations, the set of target storage bins B_{tar}, the set of workstations P with their locations, and the set of delivery tasks del_tasks that need to be accomplished by the robots. It outputs a sequence of actions for each robot as a solution. It uses a suboptimality factor w for invoking ECBS on Line 34.

On Line 1, the algorithm starts by sorting the delivery tasks in del_tasks in increasing order of the depth of their storage bins: If two target storage bins are on the same stack, the digging operations used for retrieving the storage bin that is placed higher on the stack also serve the purpose of retrieving the storage bin that is placed lower on the stack. Beyond this, the sorting breaks ties in favor of the indexing order used in the input. On Lines 2–7, the algorithm initializes the data structures TQ_i, for $1 \leq i \leq k$, $expanded\_tasks$, $precedences$, and $system\_copy$. TQ_i is intended to contain the sequence of relocation tasks and retrieval tasks that robot r_i attends to. $expanded\_tasks$ is intended to contain a set of relocation tasks and retrieval tasks with precedence constraints between them recorded in $precedences$: Respecting these constraints guarantees the execution of every task without interferences from other tasks. $system\_copy$ is intended to track the locations of all storage bins.

On Lines 8–22, the algorithm uses a loop to generate the relocation tasks and the retrieval task required for each delivery task. It starts each iteration of this loop by popping the first element (b_i, g_i^P) from del_tasks on Line 9. It then initializes a data structure $decomp$ on Line 10: $decomp$ is intended to glean (a) the relocation tasks necessary for the storage bins on top of b_i and (b) the retrieval task that moves b_i to g_i^P and then back to a storage location. On Lines 11–17, the algorithm iterates through every storage bin b_j on top of b_i, in increasing order of their depth, and relocates b_j to a storage column at g_j. g_j can be chosen using one of many strategies. In our current implementation, the algorithm first rules out the storage columns of all the storage bins that appear in the remaining delivery tasks in del_tasks. It then chooses the nearest storage column that is not ruled out. If no such storage column exists, it simply chooses the nearest storage column. On Line 15, the algorithm appends the resulting relocation task to $decomp$ and, on Line 16, updates $system\_copy$. On Line 18, the algorithm chooses a storage location g_i for returning b_i from g_i^P. g_i is chosen using the same logic as on Line 14. On Line 19, the algorithm appends the resulting retrieval task (b_i, g_i^P, g_i) to $decomp$. On Line 20, it appends $decomp$ to $expanded\_tasks$ and, on Line 21, updates $system\_copy$.

On Lines 23–25, the algorithm records the precedence constraints between tasks within each individual $decomp$ element d: This corresponds to the order in which tasks are generated on Lines 9–21 for each delivery task in del_tasks. On Lines 26–28, the algorithm records the precedence constraints between tasks across pairs of $decomp$ elements d_i and d_j: (a) If two tasks $\tau \in d_i$ and $\tau' \in d_j$, for $i < j$, mention the same storage bin, τ has to finish before τ' can start, and (b) If the task $\tau' \in d_j$ places a storage bin at the location of the storage bin of the task $\tau \in d_i$, τ has to finish before τ' can start.

On Lines 29–33, the algorithm iterates through the tasks in $expanded\_tasks$, for which $expanded\_tasks$ is flattened to remove the construct of the $decomp$ element. In each iteration, it first gathers on Line 30 a task τ that can be executed without precedence constraints. On Lines 31–32, it then assigns a robot r_i to accomplish τ based on a simple heuristic: The heuristic examines all robots and the tasks currently assigned to them and picks the robot that can attend to τ at the earliest timestep after completing its currently assigned tasks. The earliest timestep is evaluated for each robot while ignoring the other robots.

On Line 34, the algorithm invokes and returns the output of a modified ECBS procedure $ECBS_R$. This procedure takes as input G, R, the task queue TQ_i of each robot r_i, the precedence constraints $precedences$, and the suboptimality factor w. It outputs a sequence of actions for each robot: Together, they accomplish all the tasks while meeting the precedence constraints and avoiding the vertex conflicts and edge conflicts with each other.

$ECBS_R$ modifies ECBS as follows. In $ECBS_R$, each CT node N not only contains the fields $N.paths$ and $N.constraints$ but also contains the two additional fields $N.actions$ and $N.task\_labels$. $N.paths$ describes the location that each robot occupies at each timestep. $N.actions$ describes the action that each robot takes at each timestep. Lifting (lowering) a storage bin from (to) a depth d takes $2d$ timesteps. $N.constraints$ records the conflict-resolution constraints imposed by the high-level search. $N.task\_labels$ describes the task that each robot attends to at each timestep. The task label of a robot is updated when: (a) the robot starts lifting the storage bin for a new task and (b) the robot finishes lowering the storage bin for an ongoing task. In between tasks, the robot is considered to be in the ‘idle’ state.

In the high-level search, $ECBS_R$ recognizes conflicts and resolves them via branching. The conflicts are of two kinds: (a) the vertex conflicts and edge conflicts and (b) violations of the precedence constraints. The vertex conflicts and edge conflicts are resolved via spatiotemporal constraints, as described in Section 2.3. However, the violations of precedence constraints are resolved differently. A precedence constraint $\tau \prec \tau'$ is violated when a robot r_j attends to the task τ' before the task τ is completed. Suppose that the task τ is assigned to the robot r_i, which completes it at timestep t_c. The violation of the precedence constraint is resolved by generating two child CT nodes (Zhang et al. 2022). The first child CT node enforces the robot r_j to start attending to the task τ' after the timestep t_c. The second child CT node enforces the robot r_j to start attending to the task τ' before or at the timestep t_c and the robot r_i to complete the task τ before the timestep t_c. For the first (second) child CT node, $ECBS_R$ runs the low-level search to replan the path and actions for the robot r_j (r_i). The resolution of the violations of precedence constraints is done with higher priority than the resolution of vertex conflicts and edge conflicts.

In the low-level search, $ECBS_R$ modifies and uses Multi-Label A^* (MLA^*) (Grenouilleau, van Hoeve, and Hooker 2019) instead of spatiotemporal A^*. MLA^* finds the shortest path for an agent with respect to an ordered list of goals.[4] $ECBS_R$'s modifications of MLA^* are similar to those in (Li et al. 2021b) but with some differences. Each low-level node n records both the robot's location and its action at that location. The possible actions are ‘move’, ‘wait’, ‘lower’ apparatus, and ‘lift’ apparatus. Each low-level node

[4]A straightforward approach may use spatiotemporal A^* to compute the shortest path between each pair of consecutive goals and then concatenate these paths. However, this does not guarantee an overall shortest path because spatiotemporal constraints introduce dependencies between different path segments.

Algorithm 2: Bin-Centric Planner

```
Input: $G$, $R$, $B$, $B_{tar}$, $P$, $del\_tasks$
Output: a sequence of actions for each robot
Parameter: suboptimal factor $w$
 1: $bin\_paths \leftarrow ECBS_B(G, B, B_{tar}, del\_tasks, w)$.
 2: for $r_i \in R$ do
 3:     $TQ_i \leftarrow \{\}$
 4: end for
 5: Sort $bin\_paths$ in increasing order of the storage bins' earliest
    timestep to be able to move.
 6: while $bin\_paths \neq \emptyset$ do
 7:     Pop the first element $path$ from $bin\_paths$.
 8:     Let $s$ be the start location of $path$.
 9:     Let $r_i$ be the robot that can start to chaperone $path$ at the
        earliest timestep after completing its tasks in $TQ_i$.
10:     Add $(s, path)$ to $TQ_i$.
11: end while
12: return
    $ECBS_R(G, R, \{TQ_1, TQ_2 \dots TQ_k\}, bin\_paths, w)$
    to chaperone $bin\_paths$.
```

n also has an attribute $n.task\_label$ that records the current task of the robot. $n.task\_label$ updates when: (a) the robot starts lifting the storage bin for a new task and (b) the robot finishes lowering the storage bin for an ongoing task. In between tasks, $n.task\_label$ is set to 'idle'.

The tasks in TQ_i are translated into a sequence of goals. Since every relocation and retrieval task consists of lifting a storage bin, lowering it, with or without delivering it to a workstation, the accomplishment of these actions constitutes the sequence of goals. Each low-level node n uses an attribute $n.label$ to keep track of the number of accomplished goals. If $n.label$ equals the cardinality of the goal sequence, the low-level search terminates and returns both the path and the sequence of actions for that robot. The child nodes generated for the low-level node n must respect the spatiotemporal constraints and the precedence constraints imposed by the high-level CT node; otherwise, it is pruned. The heuristic value of the low-level node n is the heuristic value from n to the next goal plus the sum of the heuristic values between the remaining consecutive goals. Heuristic values are computed by ignoring the constraints imposed by the high-level CT node. Finally, the low-level search is a focal search with suboptimality factor w. The focal list is sorted by the foregoing heuristic values; ties are broken in favor of a smaller number of conflicts against other planned paths.

4.2 Approach 2: Bin-Centric Planning

Algorithm 2 shows the pseudocode for the bin-centric approach. It has the same input and output formats as Algorithm 1.

On Line 1, the algorithm starts by invoking a modified ECBS procedure $ECBS_B$ that takes as input G, B, B_{tar}, del_tasks, and w. It outputs a sequence of locations (path) for each storage bin, which is stored in the data structure bin_paths. bin_paths contains information on (a) how to move the target storage bins from their start locations to their assigned workstations, (b) how to move the target storage bins from their workstations back to the storage area, and (c) how to relocate any storage bins necessary. The paths of the storage bins in bin_paths are conflict-free as outputted by $ECBS_B$.

$ECBS_B$ modifies ECBS as follows. All storage bins in B are treated as the agents that operate on the input graph G: They are subject to vertex conflicts and edge conflicts on G. Moreover, since storage bins can only be lifted (lowered) from (onto) a stack in a storage column one at a time, two storage bins b_i and b_j moving within the same storage column simultaneously is treated as a column conflict. A column conflict $(b_i, b_j, u, v, u', v', t)$ occurs when two storage bins b_i and b_j move from u to v and from u' to v', respectively, between timesteps t and $t+1$ such that the XY-coordinates of u, v, u', and v' are identical. Such a conflict is resolved by generating two child CT nodes: One child CT node prohibits the movement of the storage bin b_i from u to v between timesteps t and $t+1$; the other child CT node prohibits the movement of the storage bin b_j from u' to v' between the same timesteps. Storage bins that are not in B_{tar} move if and only if doing so clears the path(s) for the target storage bin(s). For relocating a storage bin, $ECBS_B$ finds the shortest path to the nearest available storage column, similar to Line 14 of Algorithm 1. Moreover, $ECBS_B$ uses MLA^* in its low-level search since the target storage bins have two consecutive goal locations.

On Lines 2–4, Algorithm 2 initializes the data structure TQ_i that is intended to contain the sequence of path chaperone tasks that the robot r_i attends to. On Line 5, it sorts the elements in bin_paths in increasing order of the storage bins' earliest timestep to be able to move. On Lines 6–11, the algorithm retrieves each $path$ from bin_paths and assigns the corresponding path chaperone task to a robot. This assignment uses the same heuristic as on Line 31 of Algorithm 1. On Lines 8 and 10, the algorithm retrieves the start location of a path to facilitate fast downstream computations. The tuple $(s, path)$ in TQ_i directs the robot r_i to move to the start location s to start the path chaperone task on $path$. On Line 12, Algorithm 2 runs a slightly different version of $ECBS_R$ compared to Line 34 of Algorithm 1. Here, $ECBS_R$ plans the path of each robot so that it moves to the start location of each path chaperone task assigned to it before starting that task. In between, $ECBS_R$ simply conforms to the movements of the storage bins that it chaperones. Finally, $ECBS_R$ composes the complete action sequences and paths for each robot to move the storage bins.

5 Empirical Evaluation

In this section, we present an empirical evaluation of our proposed approaches for cube-based AS/RS: the robot-centric approach and the bin-centric approach. We implemented the algorithms in both approaches using C++ and conducted our experiments on a desktop running Ubuntu with a 4.20 GHz AMD Ryzen 7 7800X3D processor and 16 GB memory. We built a simulation of the warehouse environment using the Robot Operating System (ROS) and Gazebo. Our algorithms can interface with this environment. Figure 2 shows a snapshot of a simple cube-based AS/RS simulation in Gazebo. For all algorithms, we set a time limit

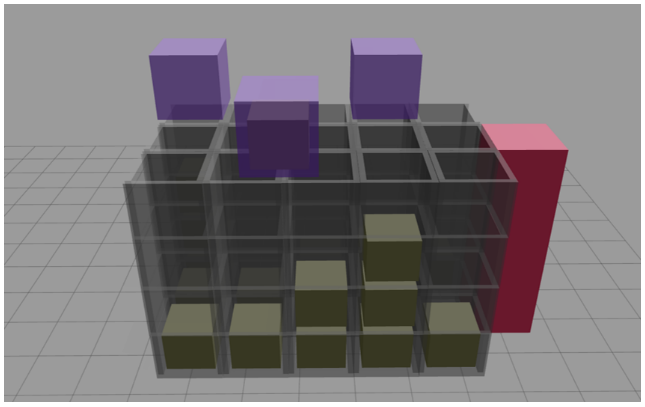


Figure 2: Shows a snapshot of a simple cube-based AS/RS simulation in Gazebo. The gray grid represents the storage area. The gray boxes stacked in columns represent the storage bins. The purple cubes that move on top of the storage area represent the robots. The frontmost robot is shown carrying a storage bin within its hull. The red column represents the workstation.

of 60 seconds on each problem instance. We set $w = 2$ in all cases.

We used a $10 \times 10 \times 10$ storage area to generate our problem instances. The storage area stands on an XY-plane with a footprint of 10×10 2D grid cells. A column of height 10 stands on each such cell, accounting for a total of $1,000$ storage cells. We generated 5 different warehouse maps by filling these $1,000$ storage cells with 600 storage bins arranged as stacks within the 100 columns. In each warehouse map, the stacks are placed randomly on the XY-plane and are of random heights ≤ 10. In each warehouse map, the start locations of 10 robots are also specified. Moreover, across all warehouse maps, we fixed the locations of 3 workstations on one side of the storage area, as shown in Figure 2.

We draw empirical results by varying the number of target storage bins from 5 to 30 in increments of 5. For each setting of the number of target storage bins, we formulate 50 problem instances: We formulate 10 problem instances from each of the 5 warehouse maps. In each problem instance, the target storage bins are chosen randomly from the 600 storage bins. Moreover, a delivery task is formulated for each target storage bin by choosing one of the 3 workstations at random as its destination workstation.

Figure 3 shows the comparative results of the robot-centric approach and the bin-centric approach with respect to various performance metrics. The top panel shows the comparison with respect to the success rate. Each data point represents, as a fraction, the number of problem instances (out of 50) that can be solved within the time limit. We observe that the success rate of the bin-centric approach is marginally better than that of the robot-centric approach.

The middle panel shows the comparison with respect to the average runtime (measured in seconds). Each data point is averaged only over the successful runs. For smaller problem instances, we observe that the bin-centric approach has an average runtime that is marginally better than that of the robot-centric approach. For larger problem instances, we observe the opposite pattern. However, this is because the bin-centric approach has a higher success rate and solves hard problems that contribute more to the average runtime. The bottom panel shows the comparison with respect to the average sum of costs, that is, the average quality of the solutions. Each data point is averaged only over the successful runs. We observe that there is only a marginal difference between the robot-centric approach and the bin-centric approach.

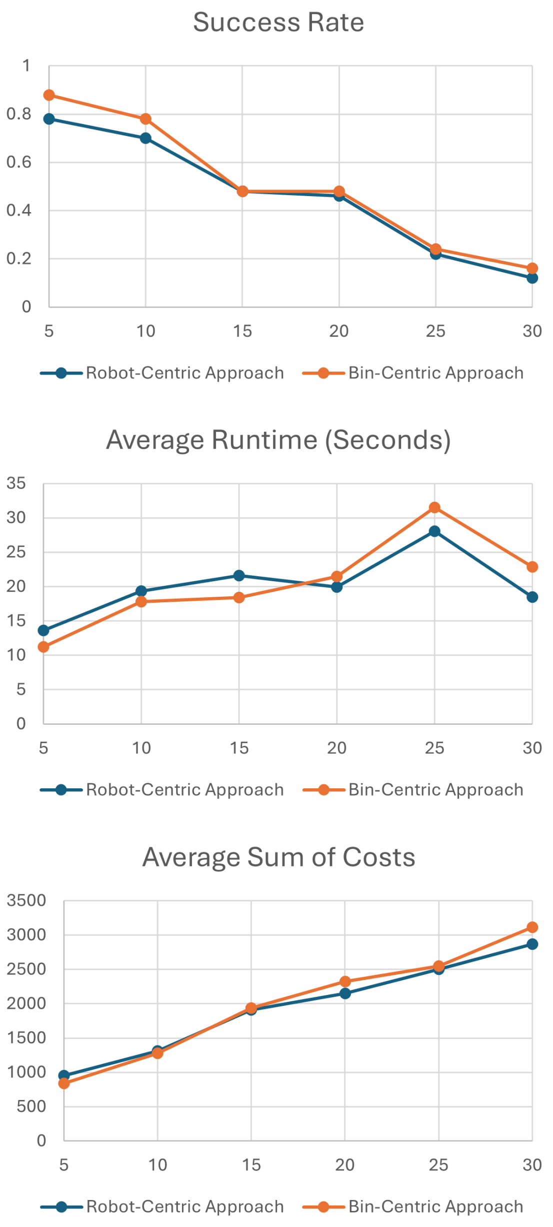


Figure 3: Shows the comparative results of the robot-centric approach and the bin-centric approach with respect to various performance metrics. The horizontal axis shows an increasing number of target storage bins (number of delivery tasks). The vertical axis shows the success rate, average runtime, and the average sum of costs in the three panels, top to bottom, respectively.

6 Conclusions and Future Work

In this paper, we studied cube-based AS/RS and the coordination problems that arise in them. We formalized one such cornerstone coordination problem and discovered its combinatorial similarities to the MAPF problem. Hence, we proposed the use of MAPF technologies for planning in cube-base AS/RS. We proposed two viable approaches: the robot-centric approach and the bin-centric approach. In both approaches, we carefully adapted the popular CBS framework from the MAPF domain to address the unique challenges of cube-based AS/RS: These unique challenges are centered around the added complexity of bin digging. In the robot-centric approach, the robots are treated as the agents and plans are generated for them to transport the storage bins. In the bin-centric approach, the storage bins are first treated as the agents and plans are generated for them to move autonomously; the robots are then treated as the agents and plans are generated for them to chaperone the movements of the storage bins. Overall, we demonstrated the value of MAPF technologies in cube-based AS/RS and experimentally showed that the bin-centric approach marginally outperforms the robot-centric approach. For doing so, we also built a simulation environment in Gazebo that interfaces with our algorithms.

Given the advantages of cube-based AS/RS, we envision that an efficient way to address the bin digging problem in them is critical to the future of warehouse automation. Hence, in future work, we will import more MAPF technologies to cube-based AS/RS based on the proof of concept provided in this paper.

References

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Proceedings of the Seventh International Symposium on Combinatorial Search*.

Beckschäfer, M.; Malberg, S.; Tierney, K.; and Weskamp, C. 2017. Simulating Storage Policies for An Automated Grid-Based Warehouse System. In *Computational Logistics: 8th International Conference*. Springer.

Chen, X.; Yang, P.; and Shao, Z. 2022. Simulation-Based Time-Efficient and Energy-Efficient Performance Analysis of An Overhead Robotic Compact Storage and Retrieval System. *Simulation Modelling Practice and Theory*.

Chen, Z.; Harabor, D.; Li, J.; and Stuckey, P. J. 2024. Traffic Flow Optimisation for Lifelong Multi-Agent Path Finding. In *Thirty-Eighth AAAI Conference on Artificial Intelligence*. AAAI Press.

Choudhury, S.; Solovey, K.; Kochenderfer, M. J.; and Pavone, M. 2020. Efficient Large-Scale Multi-Drone Delivery Using Transit Networks. In *IEEE International Conference on Robotics and Automation*.

Cohen, L.; Uras, T.; Kumar, T. K. S.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*.

Djupesland, E. E. 2023. *Reinforcement Learning for Lifelong Multi-Agent Pathfinding in AutoStore System*. Master's thesis, The University of Bergen.

Franke, J.; and Schuderer, P. 2021. Simulationsbasierte Untersuchung der Grenzproduktivität von Robotern in einem AutoStore-Lagersystem. *Simulation in Produktion und Logistik*, 197.

Gordon, O.; Filmus, Y.; and Salzman, O. 2021. Revisiting the Complexity Analysis of Conflict-Based Search: New Computational Techniques and Improved Bounds. In *Proceedings of the Fourteenth International Symposium on Combinatorial Search*.

Grenouilleau, F.; van Hoeve, W.; and Hooker, J. N. 2019. A Multi-Label A* Algorithm for Multi-Agent Pathfinding. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*. AAAI Press.

Ko, D.; and Han, J. 2022. A Rollout Heuristic Algorithm for Order Sequencing in Robotic Compact Storage and Retrieval Systems. *Expert Systems with Applications*.

Li, J.; Chen, Z.; Zheng, Y.; Chan, S.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2021a. Scalable Rail Planning and Replanning: Winning the 2020 Flatland Challenge. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling*. AAAI Press.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021b. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*. AAAI Press.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019a. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *AAAI Conference on Artificial Intelligence*.

Ma, H.; Hönig, W.; Kumar, T. K. S.; Ayanian, N.; and Koenig, S. 2019b. Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery. In *The Thirty-Third AAAI Conference on Artificial Intelligence*. AAAI Press.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. ACM.

Nguyen, V.; Obermeier, P.; Son, T.; Schaub, T.; and Yeoh, W. 2019. Generalized Target Assignment and Path Finding Using Answer Set Programming. In *Proceedings of the International Symposium on Combinatorial Search*.

Okumura, K. 2023. Lacam: Search-Based Algorithm for Quick Multi-Agent Pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Ren, Z.; Rathinam, S.; and Choset, H. 2023. CBSS: A New Approach for Multiagent Combinatorial Path Finding. *IEEE Trans. Robotics*.

Salzman, O.; and Stern, R. 2020. Research Challenges and Opportunities in Multi-Agent Path Finding and Multi-Agent Pickup and Delivery Problems. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*.

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 *Proceedings of the Twelfth International Symposium on Combinatorial Search*.

Tan-Thuan Banh, T.-N. L. T.-B. T. B.-T. V., Xuan-Trieu Nguyen. 2024. Efficient Algorithms on Dynamic Obstacle Avoidance for Multi-Robot Agents In Automated Warehouse System. In *Proceedings of the 2024 9th International Conference on Intelligent Information Technology*. Association for Computing Machinery.

Trost, P.; and Eder, M. 2024. A Performance Calculation Approach for A Robotic Compact Storage and Retrieval System (RCS/RS) Serving One Picking Station. *Production & Manufacturing Research*.

Trost, P.; Kartnig, G.; and Eder, M. 2023. Simulation Study of RCS/R-systems with Several Robots Serving One Picking Station. *FME Transactions*.

Varambally, S.; Li, J.; and Koenig, S. 2022. Which MAPF Model Works Best for Automated Warehousing? In *Proceedings of the Fifteenth International Symposium on Combinatorial Search*. AAAI Press.

Wang, Q.; Veerapaneni, R.; Wu, Y.; Li, J.; and Likhachev, M. 2024. MAPF in 3D Warehouses: Dataset and Analysis. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI*.

Zhang, H.; Chen, J.; Li, J.; Williams, B. C.; and Koenig, S. 2022. Multi-Agent Path Finding for Precedence-Constrained Goal Sequences. In *21st International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent System.

Zhang, Y.; Fontaine, M. C.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2023. Multi-Robot Coordination and Layout Design for Automated Warehousing. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*.

Zheng, Y.; Ravi, S.; Kline, E.; Koenig, S.; and Kumar, T. K. S. 2022. Conflict-Based Search for the Virtual Network Embedding Problem. In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling*.

Zheng, Y.; Ravi, S.; Kline, E.; Thurlow, L.; Koenig, S.; and Kumar, T. K. S. 2023. Improved Conflict-Based Search for the Virtual Network Embedding Problem. In *Proceedings of the Thirty-Second International Conference on Computer Communications and Networks*.

Zou, B.; de Koster, R. M. B. M.; and Xu, X. 2016. Evaluating Dedicated and Shared Storage Policies in Robot-Based Compact Storage and Retrieval Systems. *ERIM report series research in management Erasmus Research Institute of Management*.