2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Feasibility study: Using Bypass Directly in Structured Warehouse for Multi-Agent Path Finding

Sen Xu[1] and Kai Zhao[1]

***Abstract*— In warehouse environments, corridor conflicts often lead to traffic congestion, which result in many Multi-Agent Path Finding (MAPF) algorithms failing to find solutions within a reasonable time limit. Previous works have studied using *corridor reasoning techniques* or incorporating guidance, such as *highways*, to address this problem. However, these approaches often either encounter timeouts or yield low-quality solutions. In this work, based on Conflict-Based Search (CBS), we propose a technique called *Reversible Lanes*, specifically designed to address corridor conflicts by imposing a new constraint that forces agents to use bypasses for conflict resolution. Our approach is motivated by three key observations from prior research: (1) in warehouse maps, the overhead associated with maintaining solution optimality via *corridor reasoning techniques* is often disproportionate to the benefits gained; (2) the fixed nature of manually designed highways exhibits a lack of adaptability, leading to poor solution quality on certain instances; and (3) the structural properties of warehouse layouts render direct bypass usage feasible and incur minimal additional costs. Theoretically, we demonstrate the feasibility of our algorithm by analyzing its relationship to both *corridor reasoning techniques* and *highways*. Experimentally, the results show that our algorithm provides a more effective approach for resolving corridor conflicts compared to these existing methods, achieving a superior trade-off between solution quality and computational efficiency by finding near-optimal solutions with reduced runtime.**

I. INTRODUCTION

MAPF [1] is a problem of finding a set of collision-free paths for a group of agents on a given map while minimizing a certain objective function, typically the makespan or the sum of agents' action costs. It has been widely used in video games [2], aircraft towing [3], railway planning [4], and automated warehouses [5].

CBS [6] is currently the most widely used two-level search algorithm for solving MAPF. The central idea of CBS is decomposing the MAPF problem into two levels: individual agent path search and conflict detection and resolution. By iteratively identifying and resolving conflicts between paths in the high-level search, and finding the shortest paths for each agent that satisfy the new constraints in the low-level search, the algorithm collaboratively finds collision-free optimal paths for all agents in the environment. Researchers have proposed a number of enhancements to accelerate CBS [7]–[10], which greatly improve the scalability of CBS, further expanding its application fields.

*This work was supported by the National Natural Science Foundation of China under Grant 62403082, in part by the Open Research Fund of Key Laboratory of Machine Intelligence and System Control, Ministry of Education under Grant MISC-202405.

[1]Sen Xu and Kai Zhao are with the School of Automation, Chongqing University, China. Emails: {54xusen@gmail.com, zhaokai@cqu.edu.cn}.

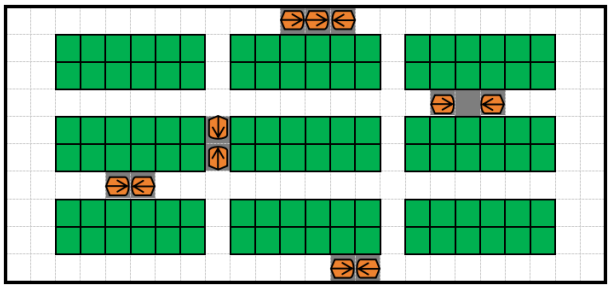
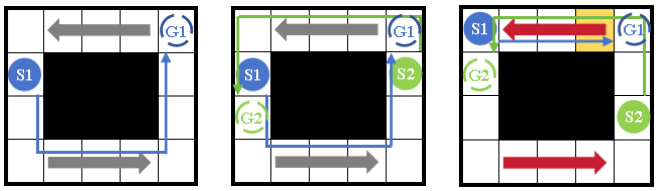


Fig. 1: Illustration of a warehouse map. Green cells are obstacles, non-green cells are traversable and gray cells represent areas where corridor conflicts occur.

Among the many applications of CBS, warehouses receive the most attention [11]–[13], as the setup of a warehouse is usually consistent with many of the assumptions used in the MAPF literature [5]. This work concentrates on structured warehouse maps similar to the one depicted in Fig. 1. These maps, with their dense spatial arrangement, are prone to generating frequent head-to-head collisions within narrow passages, a phenomenon referred to as corridor conflicts [14]. This class of conflicts poses a significant challenge to pathfinding, as the conventional vertex and edge constraints imposed by CBS only enforce conflict resolution at a single time step. However, agents involved in corridor conflicts often immediately encounter new conflicts in subsequent time steps. This process will continue until a conflict-involved agent either abandons the corridor to use a bypass or waits outside the corridor until another passes, which is time-consuming. Experimental evidence shows that resolving a corridor conflict of length k causes the number of expanded high-level nodes to grow exponentially as 2^{k+1} [14]. This computational inefficiency has motivated researchers to focus on specialized methods for addressing this unique conflict type.

[14], [15] propose *corridor reasoning techniques* to accelerate the resolution of corridor conflicts. By detecting corridor conflicts and imposing range constraints [16], it can resolve corridor conflicts within a single branch. This algorithm reduces the runtime of CBS by an order of magnitude while preserving its completeness and optimality. However, before each branch expansion, it has to execute multiple times A^* for preprocessing. As corridor conflicts increase, timeouts frequently occur. [17] introduces *highways* into CBS as another common approach for resolving corridor conflicts. It does not accelerate the conflict resolution process but guides agents to traverse corridors in a prede-



(a) (b) (c)

Fig. 2: Instances when the highways perform poorly. The gray arrows represent the direction of the *strict-limited highways*, while the red arrows represent the direction of the *soft-limited highways*(w=2). (a) there is no conflict on the shortest path, resulting in an unnecessary detour that increases the cost by 4. (b) if the direction of the highways is changed, the conflict can also be avoided, and the path cost is further reduced by 3. (c) w=2 is insufficient to guide the agents toward a conflict-free path, and conflicts will still occur at the yellow cell.

fined direction through human-designed highways, thereby directly avoiding head-to-head corridor conflicts. Typically, there are two ways to set up highways [18]. Firstly, termed *strict-limited highways*, restricts agents' movements to a single direction within the corridor. The second, *soft-limited highways(w)*, employs heuristic value inflation for directions opposite to the highways to encourage agents to follow the designed direction. Here, w denotes the inflation factor, which adjusts the encouragement intensity to flexibly balance runtime and solution quality across different instances. As $w \to \infty$, *soft-limited highways* approximate the behavior of *strict-limited highways*. Highways can already handle corridor conflicts well, but manually designed guidance still has certain shortcomings in some cases: (1) for *strict-limited highways*, agents may take unnecessary detours under strict constraints (Fig. 2 (a)). Additionally, the fixed design approach lacks adaptability, as better direction designs often exist for certain instances (Fig. 2(b)), and (2) for *soft-limited highways*, the selection of w is challenging. An insufficient value of w may not provide enough incentive for agents to follow the designated highways, potentially leading to inefficient conflict resolution and timeouts (Fig. 2(c)). Conversely, an overly large w can lead to similar problems as observed with *strict-limited highways*.

[19]–[21] have studied how to generate guidance automatically. The central idea of their methods is to iteratively execute multiple random single-agent pathfinding before centralized planning to capture congestion information on the map and then transform it into movement costs or edge directions based on a human-designed function for the MAPF algorithm to guide its search. Although these methods can automatically generate corresponding guidance based on the structure of different instances, the flaw of such methods is quite apparent, that is, the choice of the number of iterations is quite troubling. An insufficient iteration count may lead to inadequate capture of problem structure by the generated guidance, thereby introducing excessive randomness that exacerbates planning complexity. Conversely, an excessively high number of iterations may not be worth the effort, as the time invested in offline optimization to generate guidance for a given instance, whose solving difficulty is unknown, may exceed the time required to solve it directly.

Based on the preceding analysis, we propose a new technique, *Reversible Lanes*, within the CBS framework to address corridor conflicts online. The implementation of our algorithm is straightforward. First, we detect corridor conflicts and then apply a special directional obstacle constraint to force the agent to use bypasses for direct conflict avoidance. In contrast to *corridor reasoning techniques*, which is prone to timeouts as the density of agents increases, we consider omitting the elaborate reasoning process and instead directly use bypasses to trade for runtime. Compared with *strict-limited highways*, which can be seen as imposing the same directional obstacle constraints on each agent before planning and quickly returns solutions but with poor quality, we, on the other hand, consider imposing constraints online during the search process to trade for the solution quality. Therefore, our algorithm can be regarded as a simplified variant of *corridor reasoning techniques*, or as *strict-limited highways* with online adaptive capabilities, similar to reversible lanes. We aim to find a balance between the above two algorithms to achieve better performance on certain instances. Theoretically, *soft-limited highways(w)* can also be regarded as an algorithm situated between *corridor reasoning techniques* and *strict-limited highways*, and owing to the existence of w, compared with our algorithm, it can freely balance runtime and solution quality, thus having stronger flexibility. However, this sensitivity to w also means that its performance is highly dependent on parameter selection. Moreover, the two approaches are not mutually exclusive and could potentially complement each other. Experimental results also show that *soft-limited highways(w)* can at most outperform our algorithm in one aspect by adjusting w, while in some cases, our algorithm performs better in both runtime and solution quality.

II. Preliminaries

In this section, we formalize the definition of MAPF and introduce the basic algorithm CBS.

A. Multi-Agent Path Finding (MAPF)

In this paper, we follow the standard MAPF defined in [1] that (1) considers vertex and edge conflicts, (2) adopts the disappear-at-target assumption, and (3) optimizes the sum of costs. Formally, a MAPF problem is defined by a graph $G = (V, E)$ and a set of n agents $\{a_1, ..., a_n\}$. Each agent a_i has a start vertex $s_i \in V$ and a target vertex $g_i \in V$. Time is assumed to be discretized, and in every time step, each agent is situated in one of the graph vertices and can perform a single action with unit cost, which includes moving to an adjacent vertex or staying at its current vertex. A path p_i for agent a_i is a sequence of vertices that are adjacent or identical, starting from the start vertex s_i and ending at the target vertex g_i. Agents immediately disappear after reaching

their target. A conflict between paths of agents a_i and a_j is either a vertex conflict (a_i, a_j, v, t), that is, agents a_i and a_j occupy the same vertex $v \in V$ at the same time step t, or an edge conflict (a_i, a_j, u, v, t), that is, agents a_i and a_j traverse the same edge $(u, v) \in E$ in opposite directions at the time step. Our task is to find a set of conflict-free paths that move all agents from their start vertices to their target vertices while minimizing the sum of their individual path costs.

B. Conflict-Based Search (CBS)

Conflict-Based Search (CBS) [6] is a two-level optimal search algorithm for MAPF. At the low level, CBS invokes the space-time A^* to find the shortest path for each agent under the constraints added by the high level. At the high level, CBS performs a best-first search on a binary constraint tree (CT). Each CT node *N* contains:

(1) A set of constraints (*N.constraints*), where a constraint is either a vertex constraint (a_i, v, t), which prohibits an agent from occupying vertex v at timestep t, or an edge constraint, which prohibits an agent from traversing the edge (u, v) at timestep t.

(2) A solution (*N.solution*), which consists of a set of paths for each agent, that satisfies *N.constraints*.

(3) the total cost (*N.cost*), that is, the sum of the path costs of all agents.

When expanding a CT node *N*, CBS checks for conflict in *N.solution*. If there are none, *N.solution* is the optimal solution and CBS terminates. Otherwise, CBS chooses one of the conflicts and resolves it by splitting *N* into two child CT nodes. In each child CT node, an additional constraint is imposed to prohibit one agent involved in the conflict from using the conflicting vertex or edge at the specific conflicting timestep. The low-level search is then invoked to compute the shortest path for this agent that adheres to the new constraint, while the paths of all other agents remain unchanged. If the low-level search fails to find a valid path satisfying the constraints, this child CT node is pruned as it cannot yield a feasible solution. Through iterative conflict detection, constraint imposition, and low-level search invocations, CBS incrementally expands the CT until a node with conflict-free paths is identified, which are then considered the optimal solution.

III. CBS + Reversible Lanes

In this section, we will introduce our algorithm *CBS +Reversible Lanes* in detail. Algorithm 1 presents the pseudo-code of it. The part highlighted in blue, which includes detecting corridor conflicts [Line 10], adding directional obstacle constraints [Line 13], and determining the existence of a feasible path[Line 15], represents the aspects where our algorithm differs from CBS. We will explain [Line 10], [Line 13], and [Line 15] in section III-A, III-B, and III-C, respectively. Finally, in section III-D, we will analyze the relationships between our algorithm and *corridor reasoning techniques* as well as *highways*, and thereby illustrate the feasibility of our algorithm.

Algorithm 1: High level of CBS+Reversible Lanes

```
   Input: MAPF instance
1  Root.constraints = ∅
2  Root.solution = find paths by the low level()
3  Root.cost = SIC(Root.solution)
4  insert Root to OPEN
5  while OPEN not empty do
6      P ← best node from OPEN
       Validate the paths in P and find all conflicts
7      if P has no conflict then
8          return P.solution // P is goal
9      end
10     C ← conflict(a_i, a_j, i, d_i, d_j) in P
       // detect corridor conflict
11     foreach agent a_i in C do
12         A ← new node
13         A.constraints ← P.constraints + (a_i, i, d_i)
           // add directional obstacle
              constraint
14         A.solution ← P.solution
15         Update A.solution by invoke low level(a_i)
           // incorporate a checking
              mechanism
16         A.cost ← SIC(A.solution)
17         if A.cost < ∞ then
18             Insert A to OPEN
19         end
20     end
21 end
```

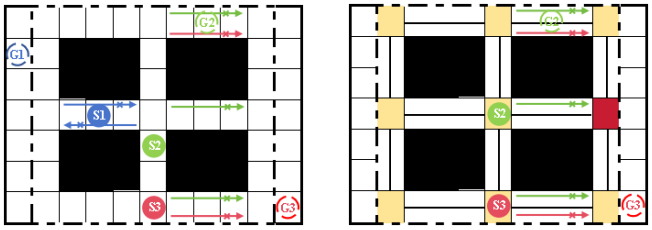
A. Detecting Corridor Conflicts

A corridor *C* is composed of a series of continuous vertices in a graph $G = (V, E)$, where each vertex in *C* has a degree of 2. We say a vertex/edge conflict is classified as a corridor conflict iff: (1) the conflict occurs within the corridor, which is determined by verifying that the degree of the conflicting vertex(or an endpoint of the conflicting edge) is 2; (2) the conflicting agents are moving in opposite directions at the time of the conflict. This is assessed using a function $JudgeMovement(path, t)$, which takes an agent's *path* and a time step *t* as inputs and returns the movement direction of the agent, with possible directions including *up*, *down*, *left*, *right*, and *wait*. Therefore, when the outputs of $JudgeMovement(path, t)$ for the two agents are opposite (e.g., one outputs "left" and the other outputs "right"), (2) is satisfied.

B. Adding Directional Obstacle Constraints

In this paper, we introduce the directional obstacle constraints to resolve corridor conflicts. We define a corridor conflict to be the tuple (a_i, a_j, i, d_i, d_j) where two agents traverse the corridor indexed as i in opposite directions (a_i traverses corridor i in the direction of d_i while a_j traverses corridor i in the direction of d_j). A directional obstacle constraint is defined as (a_i, i, d_i), where agent a_i

is prohibited from searching the path in corridor i along the direction d_i. The corridor index i can be defined according to the map itself, and the direction d_i includes: *up*, *down*, *left*, and *right*.



(a) the original map (b) the simplified map

Fig. 3: An example of the checking mechanism. The area within the dashed-line is the corridor area, and the remaining parts are the boundary area. Different agents are represented by different colors, and the crossed arrows indicate the directional obstacle constraints added to each of them. In (b), the corridor area is simplified to be composed of only 3 × 3 vertices, represented by yellow and red cells.

C. Determining the Existence of a Valid Path

Given that the directional obstacle constraint we impose is permanent and the space-time A^* continues searching until a valid path is found, it is necessary to introduce a checking mechanism to determine the existence of a valid path. We integrated this mechanism into the search process of space-time A^*. We define $PathExist(v)$ function to determine whether a valid path exists under directional obstacle constraints, and its specific implementation will be explained in the next paragraph. Whenever we prepare to expand a node in A^*, we first check whether the agent can remain at the vertex v until all finite time constraints (i.e., vertex constraints and edge constraints) expire. If not, we proceed with standard node expansion; if yes, we use $PathExist(v)$ for judgment. If it returns *True*, indicating that a path exists, subsequent processing follows the normal A^* procedure. If *False*, we discard the node directly. This process continues until the goal vertex is reached or until A^* exits due to an empty $OPEN$ (indicating no solution). Although this process may seem time-consuming, in practice, the start node can often be directly fed into $PathExist(v)$. In this case, regardless of the outcome of this initial check, the mechanism is not required for subsequent steps of the A^*, thus introducing only minimal additional overhead.

The implementation of $PathExist(v)$ function is straightforward. We can directly execute a standard A^* under the directional obstacle constraints from v to the goal vertex. If a valid path is found, return *True*; otherwise, return *False*. However, leveraging the layout characteristics of warehouse maps and our focus solely on path existence, this process can be streamlined. We first partition the map into corridor area and boundary area and simplify the representation of the corridor area. Subsequently, the node that is closest to v is reachable and belongs to the simplified corridor area is determined as the new start vertex. If no such vertex exists, we immediately return *False*. Otherwise, we perform a Breadth-First Search (BFS) from this new start vertex. (1) If the agent's goal vertex lies within the corridor area, during each expansion from node N to node N_i, check whether the goal vertex falls within the coordinate range defined by N and N_i. If yes, return *True*; if not, after all expansions are completed, return *False*. (2) If the agent's goal vertex lies within the boundary area, check whether the BFS can expand to the intersection of the boundary area and the corridor area. If so, return *True*; otherwise, return *False*.

Fig. 3 illustrates an example of the checking mechanism. We assume $S1$, $S2$, and $S3$ are start vertices of the three agents and can all be directly input into $PathExist(v)$. In Fig. 3(a), under the constraints of directional obstacles, agent a_1 cannot find a new starting vertex that meets the requirements and therefore directly returns *False*. Fig. 3(b) shows the new starting vertices that meet the requirements for $a2$ and $a3$. For agent $a2$, the BFS expansion never reaches $G2$ after all expansions, confirming no valid path exists from $S2$ to $G2$. In contrast, agent $a3$'s BFS can successfully reach the intersection (red cell in Fig. 3(b)) between the corridor area and the boundary area where $G3$ is located, confirming the existence of a path from $S3$ to $G3$.

D. Feasibility Analysis

Connection with *corridor reasoning techniques*. The key difference between *corridor reasoning techniques* and *Reversible Lanes* lies in the implementing constraints for resolving corridor conflicts: the former uses computed range constraints to achieve optimal conflict resolution, whereas the latter directly imposes directional obstacle constraints to approximate a feasible solution. If we consider a single execution of A^* as one operation, then each high-level expansion in *corridor reasoning techniques* requires 6 operations: 4 for calculating range constraints and 2 for computing the new path. In contrast, *Reversible Lanes*, by omitting the complex reasoning process, requires only 2 operations per expansion, effectively tripling computational speed. However, this is not absolute, as directly using bypasses may occasionally lead to encountering more conflicts later that slow down the solving process. Nevertheless, experimental results show that the impact of such cases is negligible, as the overhead from additional conflicts rarely outweighs the time saved by eliminating the reasoning overhead.

Connection with *strict-limited highways*. *Strict-limited highways* can be regarded as imposing the same directional obstacle constraints on each agent in every corridor. By contrast, *Reversible Lanes* dynamically determines agents' corridor movement directions based on path costs during the search process after each expansion of a CT node, functioning as adaptive *strict-limited highways*. Unlike methods that alter graph properties (e.g., by modifying edge weights or heuristic values), it imposes constraints on agents to enforce one-directional corridor traversal, ensuring that permissible movement directions within corridors are not prefixed but dynamically adjusted across different time intervals. This

adaptive, constraint-based mechanism motivates the naming of our approach as *Reversible Lanes*.

Lemma 1: For a given instance, let ϕ denote an arbitrary directional design of *strict-limited highways*, let Ψ_ϕ denote the set of all feasible solutions under this design, and let Ψ_{RL} denote the set of all feasible solutions obtained via *Reversible Lanes*. Then, $\Psi_\phi \subseteq \Psi_{RL}$.

Proof: To resolve corridor conflicts, *Reversible Lanes* enumerates all possible directional obstacle combinations. Since all such combinations are finite, and vertex constraints and edge constraints are also finite, it will terminate after a finite number of search steps. As *Reversible Lanes* initiates the search from a root node with empty constraints, while *strict-limited highways* starts from a root node with a fixed directional obstacle constraint combination ϕ, the former's search space entirely encompasses the latter's. Consequently, $\Psi_\phi \subseteq \Psi_{RL}$.

Theorem 1: If *strict-limited highways* is complete for a certain MAPF instance in warehouse maps, then *Reversible Lanes* also demonstrates completeness for this instance.

Proof: By Lemma 1, for *strict-limited highways*, as long as there exists a design method ϕ such that Ψ_ϕ is non-empty, then Ψ_{RL} must also be non-empty, i.e., a solution exists for *Reversible Lanes*.

Theorem 2: No manually designed *strict-limited highways* can outperform *Reversible Lanes* in terms of solution quality in warehouse maps.

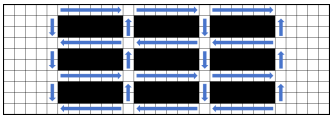
Proof: For a given instance, we assume Φ denotes the set of all feasible design methods for *strict-limited highways*. Let $cost_{min} = \min_{\phi \subseteq \Phi} \min_{\pi \subseteq \Psi_\phi} cost(\pi)$ be the minimum cost that can be found among all design methods. By Lemma 1, $\Psi_\phi \subseteq \Psi_{RL}$, and since *Reversible Lanes* performs best-first search in CT, therefore $\min_{\pi \subseteq \Psi_{RL}} cost(\pi) \leq \min_{\phi \subseteq \Phi} \min_{\pi \subseteq \Psi_\phi} cost(\pi) = cost_{min}$.

IV. Rapidly Constructing Constraints

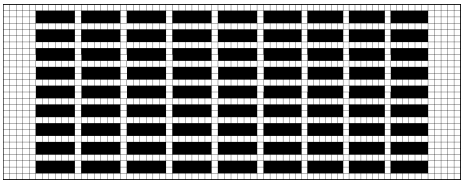
In this section, we propose two variants of *Reversible Lanes* that aim to construct directional obstacle constraints more quickly. Both variants attempt to establish consistent traversal directions inside narrow corridors at an early stage of the search, thereby providing a guidance effect similar to *strict-limited highways* and reducing the overall runtime.

The first variant modifies the conflict resolution strategy by prioritizing corridor conflicts over a chronological processing order. This approach, by first imposing directional obstacle constraints to improve the lower bound of a node in OPEN and thereby reducing the depth of the search tree, is similar to the Prioritizing Conflicts (PC) technique in [22]. The second variant builds upon the first variant, differing in that it adopts a two-stage framework, similar in spirit to the approaches in [19]–[21], in which a guidance is first established and subsequently used to guide the search. In the first stage, the algorithm follows the corridor-first conflict resolution strategy of the first variant until a node N whose solution contains no corridor conflicts is found. In the second stage, N is treated as a new root node, and the standard *Reversible Lanes* search is executed to resolve the remaining conflicts. If a feasible solution is obtained, the algorithm terminates; otherwise, it backtracks to the first stage to locate another corridor-conflict-free node. This process repeats until a solution is found. Obviously, Theorem 1 holds for both variants, whereas Theorem 2 applies only to the first variant, as the second variant returns the first feasible solution under a selected guidance instead of the minimum-cost solution over the entire *Reversible Lanes* search space.

Although both variants theoretically reduce the search space at the cost of further sacrificing solution quality, in some cases, their solving speed may instead decrease. A common drawback of both variants is that, after processing non-corridor conflicts, path changes may render some directional obstacle constraints redundant, thereby restricting agent movement. Moreover, new corridor conflicts may arise, resulting in a cycle of repeated constraint additions and path recalculations until a feasible solution is found which may complicate the search space and significantly reduce solving speed. In addition, for the second variant, N may lie on a less promising branch that does not lead quickly to a feasible solution, while a solution could exist in another part of the search tree that other algorithms would reach earlier. Fortunately, the experimental results indicate that these cases are rare, and the proposed variants effectively accelerate the algorithm with only a slight loss in solution quality.



(a) small warehouse map



(b) large warehouse map

Fig. 4: Experimental warehouse map. Black cells are obstacles, non-black cells are traversable

V. Experiments

The algorithms proposed in this paper are denoted as RL, with its two variants described in Section IV as RL-I and RL-II. *Corridor reasoning techniques* is denoted as CR, *strict-limited highways* and *soft-limited highways* are denoted as SHWY and HWY(w), with both highway designs adopting the *Crisscross*[1] from [23]. In this section, within the CBS framework, we first compare RL with SHWY and HWY(w) for each $w \in \{2, 2.5, 3\}$, and then we compare RL with

[1]Under the CC highways design method, within the corridor area, grid cells in even rows only allow agents to move rightward, grid cells in even columns only allow agents to move upward, and the design for odd rows and columns is the opposite(as shown by the blue arrow in Fig. 4(a)).

CR, thereby examining the trade-offs RL makes in solution quality and runtime against the two baseline algorithms. Finally, we evaluated the performance of the two variants, RL-I and RL-II, to assess the impact of rapidly constructing corridor constraints on the performance of RL.

The small warehouse map from [24], as shown in Fig. 4(a), is a 4-neighbor grid map of size 30×10 with 3×3 blocks in the middle. The size of the large map is 72×28, containing 9×9 blocks in the middle. For both warehouse maps, we generate 50 instances with random start and target vertices for each number of agents m. Implementations are written in Python 3.8.5 and evaluated on a 2.50 GHz Intel Core i5-12400F with 16 GB RAM. The time limits for the two maps are 30 seconds and 120 seconds, respectively.

Comparisons vs. HWY(w) and SHWY.

Fig. 5 shows the *success rates*(= number of instances solved by each algorithm within the runtime limit) for RL, SHWY, and HWY(w), for each $w \in \{2, 2.5, 3\}$. SHWY dominates other algorithms in all tested scenarios. For HWY(w), increasing the value of w encourages agents to more strictly follow the designated highways, which reduces the number of conflicts and results in a steadily rising success rate. RL outperforms all HWY(w) on smaller warehouse maps but positions itself between HWY(2) and HWY(3) on larger ones. Table I summarizes average runtimes and cost, where the runtime for unsolved instances is taken as the runtime limit. Compared with HWY(2), RL offers faster runtimes and superior solution quality. Although HWY(2.5), HWY(3), and SHWY also achieve significant speedups, this comes at the cost of solution quality, with greater speed improvements correlating with a decrease in solution quality. Overall, RL provides a more effective balance between runtime and solution quality, and no other algorithm in this experiment can dominate it in both aspects simultaneously.

An interesting observation is that: The success rates of RL on small warehouse are higher than those of HWY(3), while they decrease in large warehouses and converge to HWY(2.5). We assume this is due to the differences in agent density. For HWY(w), corridor conflicts between agents generally occur near the start and target vertices of the agents. This is because agents often initially move against the highways to reach a more suitable position before moving along the guidance, and as agents approach their destinations, they may deviate against the highways to directly reach their target vertices. In our experiment, the agent density on large warehouse is relatively low, and the distribution of the start and target vertices among agents is relatively sparse. Consequently, even if some agents move against the highways near their start or target vertices, the likelihood of conflicts with other agents moving along the guided direction remains low. In contrast, on smaller warehouse with a higher agent density, the probability of corridor conflicts will be somewhat higher.

Comparisons vs. CR.

As shown in Fig. 6, RL generally outperforms CR in terms of success rate, achieving comparable or superior results across different maps and agent densities. Table II, which

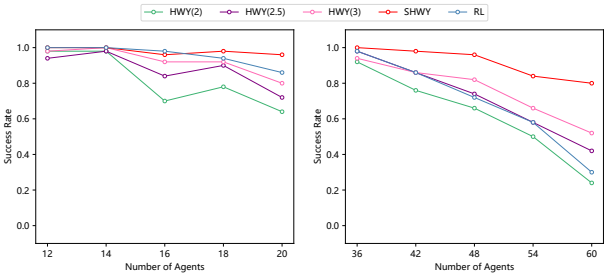


Fig. 5: Success rates for HWY(w), SHWY, and RL.

follows the same format as Table I, reports the average runtimes and solution cost. Compared with CR, RL can reduce the runtime to varying degrees across all scenarios. Notably, RL does not universally outperform CR. For example, on the large map with 60 agents, there are 2 instances where CR yields shorter runtimes and lower solution costs than RL. This corresponds to what we discussed in section III-D. Directly using the bypass may instead make the problem more complicated, not only increasing the cost but also losing the possibility of finding a solution faster.

In terms of solution quality, it is remarkable that RL shows only a minimal degradation compared to the optimal solutions found with CR, with the maximum cost gap across all instances being only 0.9%. This surprising result contradicts the expectation that simplified conflict resolution might severely compromise solution optimality. To investigate this, we analyzed the instances successfully solved by CR and focus on the conflict resolution behavior of the corridor. We classified a conflict resolution as a "bypass" when only one agent used the corridor involved in the conflict, a behavior that is directly consistent with our RL approach. Table III presents the bypass rates, which are above 70% in almost all scenarios. This may explain why RL suffers negligible quality loss, as despite CR's elaborate corridor reasoning, the practical outcome for most corridor conflicts is equivalent to RL's direct bypass strategy. Therefore, RL performs better in a comprehensive consideration of runtime and solution quality in the experiment. As for CR, the benefits of ensuring optimality are not cost-effective compared to the additional time overhead.

Comparisons vs. RL-I and RL-II.

To further assess the performance of our two variants, we

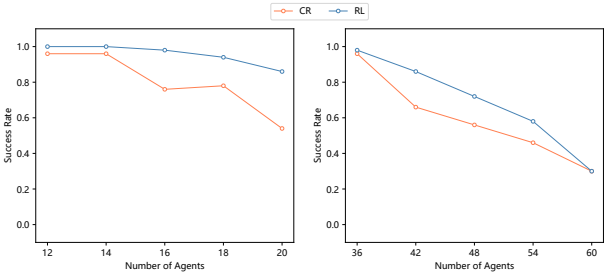


Fig. 6: Success rates for RL and CR.

TABLE I: Average runtimes in seconds and cost for HWY(w), SHWY, and RL. The Gap demonstrates the average gap measured between HWY(2) and other algorithms.

		HWY(2)		HWY(2.5)		HWY(3)		SHWY		RL	
Map	M	rt	cost	rt	cost	rt	cost	rt	cost	rt	cost
30×10	12	2.88	202.90	2.14	204.68	0.92	208.84	**0.16**	255.46	0.74	**199.08**
	14	1.06	234.58	1.08	236.68	0.55	240.54	**0.17**	295.76	1.18	**228.12**
	16	10.11	272.94	6.29	276.44	4.24	280.58	**1.77**	343.00	4.11	**267.54**
	18	8.90	314.14	5.02	317.14	3.02	322.50	**1.16**	386.70	4.12	**306.82**
	20	12.38	342.62	10.60	347.12	8.27	351.26	**2.32**	431.78	8.95	**338.16**
Gap				-23.9%	+1.1%	-54.7%	+2.7%	**-85.8%**	+25.4%	-40.1%	**-2.0%**
72×28	36	21.53	1390.98	12.56	1402.26	12.41	1429.84	**5.19**	1612.72	9.56	**1361.70**
	42	37.44	1605.82	25.84	1618.22	23.71	1648.64	**11.65**	1864.50	33.18	**1575.20**
	48	56.84	1858.64	45.28	1874.94	31.40	1907.44	**15.49**	2144.74	48.53	**1818.70**
	54	73.22	2049.16	68.33	2067.78	55.50	2104.40	**30.00**	2384.74	69.30	**2009.94**
	60	101.17	2309.00	81.25	2330.28	75.42	2372.44	**47.00**	2682.66	94.95	**2263.64**
Gap				-23.9%	+0.9%	-34.7%	+2.7%	**-66.0%**	+16.0%	-18.6%	**-2.0%**

TABLE II: Average runtimes in seconds and cost for CR and RL. The Gap demonstrates the average gap measured between CR and RL.

		CR		RL	
Map	M	rt	cost	rt	cost
30×10	12	3.35	**197.70**	**0.74**	199.08
	14	2.75	**226.84**	**1.18**	228.12
	16	10.60	**264.60**	**4.11**	267.54
	18	9.38	**304.12**	**4.12**	306.82
	20	16.61	**334.04**	**8.95**	338.16
Gap				**-59.7%**	+0.9%
72×28	36	13.60	**1361.06**	**9.56**	1361.70
	42	48.84	**1573.32**	**33.18**	1575.20
	48	69.34	**1816.60**	**48.53**	1818.70
	54	80.10	**2007.22**	**69.30**	2009.94
	60	99.11	**2260.24**	**94.95**	2263.64
Gap				**-21.9%**	+0.1%

TABLE III: Bypass rate of CR.

Small warehouse					
M	12	14	16	18	20
Bypass rate	72.5%	74.7%	66.7%	73.9%	78.6%
Large warehouse					
M	36	42	48	54	60
Bypass rate	78.3%	74.7%	75.4%	74.0%	83.3%

increased the number of agents and plotted their success rates in Fig. 7. The results are clearly consistent with the theoretical expectations: compared with RL, both variants achieve higher success rates, with RL-II showing a better advantage over RL-I. Table IV presents a comparison of the average runtimes and solution costs for the three algorithms. Since the success rates of RL approach zero at high agent densities, only results for lower densities are given. Overall, both variants reduce the runtime of RL, and RL-II generally outperforms RL-I, with the exception that it is slightly slower at 18 agents. This behavior is consistent with the discussion in section IV, where we noted that the two-stage guidance may occasionally explore a less promising branch.

Regarding solution cost, both variants incur only a negligible increase compared to RL. This limited cost degradation can be attributed to the fact that corridor conflicts dominate agent interactions in the evaluated warehouse maps. Once consistent traversal directions are established, the overall traffic structure is largely determined, and the remaining conflicts become mostly local, inducing only minor path adjustments with limited impact on the total cost. Moreover, since the low-level search remains unchanged and continues to optimize individual paths under the given constraints, the solutions found by the two variants are typically close in cost to those of RL.

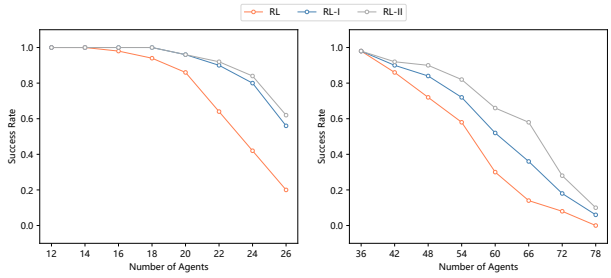


Fig. 7: Success rates for RL, RL-I, and RL-II.

In our experiments, RL-II consistently found solutions within the subtree rooted at the first corridor-conflict-free node N. Table IV also reports the percentage of the total runtime spent constructing this node across all successfully solved instances, which remains small across all settings, indicating that resolving corridor conflicts via directional obstacle constraints is efficient. Most runtime is instead spent resolving vertex and edge conflicts. Moreover, the short overhead of the first stage suggests that, beyond integrating RL with advanced CBS-based solvers [7], [10], the two-stage idea can also be combined with other MAPF solvers [25], where corridor guidance is first constructed within a CBS framework and then leveraged by a different solver.

VI. CONCLUSIONS

In this paper, we address frequent corridor conflicts in warehouse maps by introducing *Reversible Lanes*, a technique built on CBS that imposes directional obstacle con-

TABLE IV: Average runtimes in seconds and cost for RL, RL-I and RL-II. The Gap demonstrates the average gap measured between RL and RL-I/RL-II.

		RL		RL-I		RL-II	
Map	M	rt	cost	rt	cost	rt	cost
30×10	12	0.74	**199.08**	0.44	199.14	**0.34**(24%)	199.14
	14	1.18	**228.12**	0.44	228.36	**0.38**(32%)	228.44
	16	4.11	**267.54**	0.91	267.78	**0.81**(35%)	268.40
	18	4.12	**306.82**	**1.22**	307.28	1.62(28%)	307.34
	20	8.95	**338.16**	3.23	339.14	**2.53**(27%)	339.38
Gap				-65.0%	+0.1%	**-66.9%**	+0.2%
72×28	36	9.56	**1361.70**	8.00	**1361.70**	**5.73**(14%)	1361.74
	42	33.18	**1575.20**	21.69	1575.48	**18.18**(12%)	1575.74
	48	48.53	**1818.70**	34.22	1819.48	**23.16**(21%)	1819.76
	54	69.30	**2009.94**	54.08	2010.88	**34.56**(22%)	2011.22
	60	94.95	**2263.64**	77.78	2265.48	**60.75**(20%)	2266.14
Gap				-24.1%	+0.0%	**-44.6%**	+0.1%

straints to force agents toward bypasses for conflict resolution. Theoretical analysis proves its feasibility and experiments demonstrate that this approach accelerates search efficiency in warehouse environments by leveraging bypasses, all while maintaining near-optimal solution quality, showcasing a pragmatic balance between speed and performance without significantly sacrificing path optimality.

RL still has several limitations, and future work will therefore focus on three main directions: (1) treating *Reversible Lanes* as a general technique and exploring its integration with some more advanced MAPF solvers to improve scalability in large-agent scenarios, as discussed in section V, (2) adapting and refining RL to ensure its feasibility on a wider variety of map structures beyond warehouse environments, and (3) considering the lifelong MAPF problem[2] [26], where the adaptive guidance mechanism of RL can be more fully exploited, particularly the combination of Rolling-Horizon Collision Resolution (RHCR)[3] [27].

REFERENCES

[1] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. Kumar, *et al.*, "Multi-agent pathfinding: definitions, variants, and benchmarks," in *SoCS*, 2019, pp. 151–158.

[2] H. Ma, J. Yang, L. Cohen, T. K. Kumar, and S. Koenig, "Feasibility study: moving non-homogeneous teams in congested video game environments," in *AIIDE*, 2017, pp. 270–272.

[3] R. Morris, C. S. Pasareanu, K. S. Luckow, W. Malik, H. Ma, T. K. S. Kumar, and S. Koenig, "Planning, scheduling and monitoring for airport surface operations," in *AAAI Workshop on Planning for Hybrid Systems*, 2016, pp. 608–614.

[4] J. Li, Z. Chen, Y. Zheng, S.-H. Chan, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, "Scalable rail planning and replanning: winning the 2020 flatland challenge," in *ICAPS*, 2021, pp. 477–485.

[5] S. Varambally, J. Li, and S. Koenig, "Which mapf model works best for automated warehousing?" in *SoCS*, 2022, pp. 190–198.

[6] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[2]In this paper, the reason we consider the disappear-at-target assumption is: (1) our focus is on head-to-head corridor conflicts, and (2) the lifelong MAPF problem is more practically relevant, in which the agent often immediately removes to the next target or an endpoint, where it does not block any other agents, after reaching their target vertex.

[3]*Reversible Lanes* performs better on small warehouse, while RHCR solves the lifelong MAPF problem by decomposing it into a sequence of windowed MAPF instances. The two approaches complement each other effectively, and additionally, the use of *Reversible Lanes* can help reduce deadlocks in RHCR.

[7] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *SoCS*, 2014, pp. 19–27.

[8] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *AAAI*, 2019, pp. 7643–7650.

[9] J. Li, A. Felner, E. Boyarski, H. Ma, and S. Koenig, "Improved heuristics for multi-agent path finding with conflict-based search," in *IJCAI*, 2019, pp. 442–449.

[10] J. Li, W. Ruml, and S. Koenig, "Eecbs: a bounded-suboptimal search for multi-agent path finding," in *AAAI*, 2021, pp. 12 353–12 362.

[11] Q. Xu, J. Li, S. Koenig, and H. Ma, "Multi-goal multi-agent pickup and delivery," in *IROS*, 2022, pp. 9964–9971.

[12] Y. Zhang, D. Harabor, P. Le Bodic, and P. J. Stuckey, "Efficient multi-agent path finding with turn actions," in *SoCS*, 2023, pp. 119–127.

[13] K. Brown, O. Peltzer, M. A. Sehr, M. Schwager, and M. J. Kochenderfer, "Optimal sequential task assignment and path finding for multi-agent robotic assembly planning," in *ICRA*, 2020, pp. 441–447.

[14] J. Li, G. Gange, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, "New techniques for pairwise symmetry breaking in multi-agent path finding," in *ICAPS*, 2020, pp. 193–201.

[15] J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Artificial Intelligence*, vol. 301, p. 103574, 2021.

[16] D. Atzmon, R. Stern, A. Felner, G. Wagner, R. Barták, and N.-F. Zhou, "Robust multi-agent path finding," in *SoCS*, 2018, pp. 2–9.

[17] L. Cohen, T. Uras, and S. Koenig, "Feasibility study: using highways for bounded-suboptimal multi-agent path finding," in *SoCS*, 2015, pp. 2–8.

[18] M.-F. Li and M. Sun, "The study of highway for lifelong multi-agent path finding," *arXiv preprint arXiv:2304.04217*, 2023.

[19] L. Cohen, T. Uras, T. K. S. Kumar, H. Xu, N. Ayanian, and S. Koenig, "Improved solvers for bounded-suboptimal multi-agent path finding," in *IJCAI*, 2016, pp. 3067–3074.

[20] S. D. Han and J. Yu, "Optimizing space utilization for more effective multi-robot path planning," in *ICRA*, 2022, pp. 10 709–10 715.

[21] Z. Chen, D. Harabor, J. Li, and P. J. Stuckey, "Traffic flow optimisation for lifelong multi-agent path finding," in *AAAI*, 2024, pp. 20 674–20 682.

[22] E. Boyarski, A. Felner, R. Stern, G. Sharon, O. Betzalel, D. Tolpin, and E. Shimony, "Icbs: the improved conflict-based search algorithm for multi-agent pathfinding," in *SoCS*, 2015, pp. 223–225.

[23] L. Cohen, "Efficient bounded-suboptimal multi-agent path finding and motion planning via improvements to focal search," Ph.D. dissertation, University of Southern California, 2020.

[24] J. Li, D. Harabor, P. J. Stuckey, A. Felner, H. Ma, and S. Koenig, "Disjoint splitting for conflict-based search for multi-agent path finding," in *ICAPS*, 2019, pp. 279–283.

[25] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, vol. 310, p. 103752, 2022.

[26] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *AAMAS*, 2017, pp. 837–845.

[27] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *AAAI*, 2021, pp. 11 272–11 281.