Multi-Agent Corridor Reasoning for Multi-Agent Path Finding

Yiran Ni[1], **Deshi Ye**[1*]

[1]College of Computer Science and Technology, Zhejiang University,Hangzhou, China
22321163@zju.edu.cn, yedeshi@zju.edu.cn

Abstract

The Multi-Agent Path Finding (MAPF) problem is a computationally challenging task that involves coordinating collision-free trajectories for multiple cooperative agents. Although existing methods address corridor symmetry, where agents encounter repeated bidirectional conflicts in constrained environments, they typically focus exclusively on pairwise agent interactions. Our observations reveal that such pairwise symmetry frequently arises when multiple agents traverse shared corridors, necessitating repeated applications of the corridor reasoning technology over extended durations. To overcome this limitation, we propose a multi-agent corridor reasoning (MAC) technology capable of resolving group-level corridor symmetry in a single optimization step. Our theoretical analysis demonstrates that this technology preserves the completeness and optimality guarantees of Conflict-Based Search (CBS). By integrating MAC technology with CBSH-RTC, we developed CBSH-MACRT, which significantly outperforms state-of-the-art algorithms (CBSH-RTC and CBSH with mutex propagation) on standardized MAPF benchmarks, improving success rates by 8–40% and cutting runtimes by 14-67%.

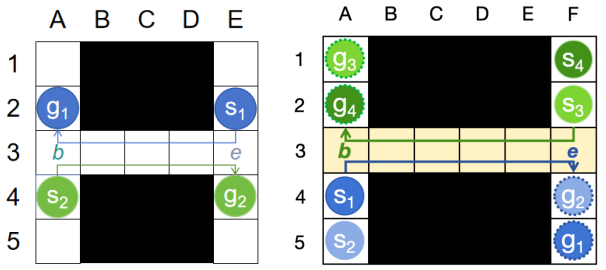
Introduction

Multi-Agent Path Finding (MAPF) represents a critical interdisciplinary challenge spanning artificial intelligence, operations research, and robotics. The problem aims to compute collision-free trajectories for all agents from their respective start positions to designated goals while minimizing the total path cost, typically measured by cumulative travel time. This formulation abstracts numerous real-world applications including automated warehouse management (Li et al. 2021b), autonomous intersection management (Dresner and Stone 2008) , drone swarm coordination (Hönig et al. 2018), and video game character control (Silver 2005). Multiple autonomous agents (e.g., robots, vehicles) must efficiently and safely navigate through shared environments to complete coordinated tasks in such scenarios. The inherent complexity of simultaneously optimizing individual trajectories while preventing spatiotemporal conflicts poses a fundamental challenge to system performance and scalability in practical implementations.

*Corresponding author

The computational complexity of the Multi-Agent Path Finding problem arises from the spatiotemporal interdependencies between agents: each agent's trajectory selection impacts not only its efficiency but also introduces potential conflicts with other agents, including collisions and deadlocks. Furthermore, challenges inherent to real-world applications – such as dynamic obstacles, real-time constraints, and heterogeneous agent characteristics (e.g., speed and size disparities) – exacerbate the problem's complexity. Consequently, developing efficient and robust MAPF algorithms is critical for enhancing the autonomy and practical deployment of multi-agent systems. Contemporary research has yielded three principal algorithmic categories:

- Optimal solvers: Conflict-Based Search (CBS) (Sharon et al. 2015) and its enhanced variants, including CBSH-RTC (Li et al. 2021a) and CBS with mutex propagation (Zhang et al. 2022);
- Bounded-suboptimal methods: Enhanced CBS (ECBS) (Barer et al. 2014) and its improved iteration, EECBS (Li, Ruml, and Koenig 2021);
- Unbounded-suboptimal approaches: Heuristic techniques such as MAPF-LNS2 (Li et al. 2022) and hybrid frameworks integrating reinforcement learning with large neighborhood search (Wang et al. 2025).



(a) an example of a corridor conflict. (b) an example of a multi-agent corridor conflict.

Figure 1. Examples of a pairwise corridor conflict and a multi-agent corridor conflict.

In prior research, Li et al. (2021a) identified a recurrent challenge in MAPF where two agents moving bidirectionally within a narrow corridor exhibit persistent conflicts due

to corridor symmetry–a phenomenon where agents traverse opposing directions in constrained spaces, resulting in repeated collisions (Figure 1 (a)). While Li et al. (2021a) resolved such pairwise conflicts through targeted range constraints without compromising algorithmic optimality, their approach proves ineffective for multi-agent corridor traversal scenarios. As exemplified in Figure 1 (b), resolving group-level conflicts necessitates iterative pairwise corridor reasoning operations.

To address multi-agent corridor conflicts efficiently, we propose CBSH-MACRT, an algorithm developed within the framework of state-of-the-art Multi-Agent Path Finding (MAPF) solvers – specifically, Conflict-Based Search (CBS) (Sharon et al. 2015) and its advanced variants CBSH-RTC (Li et al. 2021a). Our algorithm detects which agents have pairwise corridor conflicts in the same corridor, divides the agents into two sets according to their direction, and then resolves these corridor conflicts in one split by adding a set of range constraints. More importantly, we provide a rigorous theoretical proof that our technique does not violate the integrity and optimality of the CBS algorithm. We also evaluate the impact of multi-agent corridor reasoning techniques through extensive experimental comparisons, and the results show that symmetric reasoning techniques can significantly reduce CBS node expansion. In a major result, we show that our algorithm CBSH-MACRT significantly improves on CBSH-RTC (Li et al. 2021a) and Mutex Propagation (Zhang et al. 2022) in terms of both runtime and the percentage of instances solved within runtime limits.

This paper advances the state of the art through three major contributions:

- Multi-Agent Corridor Symmetry: We formalize multi-agent corridor symmetry in grid-based environments and propose unified range constraints to resolve group-level corridor conflicts in a single step.
- Optimality Preservation: We prove that our constraint-integration framework preserves the solution optimality guarantees of baseline algorithms.
- Empirical Validation: Evaluations on standard benchmarks demonstrate that our method achieves an 8-40% increase in success rate and 14–67% reduction in runtime compared to state-of-the-art Conflict-Based Search (CBS) variants.

Related Work

In this section, we will review some optimal algorithms for MAPF. The optimal MAPF solvers can be divided into four categories: A^*-based, conflict-based search (CBS)-based, and reduction-based.

Reduction-based

Reduction-based solvers differ from the three aforementioned search-based solver types. The Multi-Agent Path Finding (MAPF) problem can be reduced to well-established computational problems, including the Boolean Satisfiability Problem (SAT) (Surynek et al. 2016), Answer Set Programming (ASP) (Gómez, Hernández, and Baier 2020), flow networks, Integer Linear Programming (ILP) (Yu and LaValle 2016), Constraint Satisfaction Problem (CSP) (Wang et al. 2019), and Branch-and-Cut-and-Price (BCP) (Lam et al. 2022). Existing solvers for these problems can subsequently be applied to address MAPF. Reduction-based solvers are both optimal and complete. For small-scale MAPF instances characterized by dense obstacle configurations and high agent populations, these solvers demonstrate efficient performance.

A^*-based

A direct extension of A^* for Multi-Agent Path Finding (MAPF) involves joint state space search. This approach combines the positions of multiple agents into a joint state and applies A^* search within this unified state space, advancing all agents incrementally. However, since the joint state space grows exponentially with the number of agents, various optimization techniques have been developed to enhance A^* efficiency. These include independence detection (Standley 2010), operator decomposition (Standley 2010), partial expansion (Goldenberg et al. 2014), and subdimensional expansion (Wagner and Choset 2015).

CBS-based

The Conflict-Based Search (CBS) algorithm is the predominant optimal approach to solving MAPF problems. CBS employs a hierarchical framework comprising two layers: a high-level layer that identifies and resolves conflicts by generating constraints, and a low-level layer responsible for computing collision-free paths for individual agents. State-of-the-art optimal MAPF algorithms are predominantly derived from CBS variants. For instance, ICBS (Boyarski et al. 2015) accelerates search efficiency through conflict prioritization. CBSH (Felner et al. 2018) and CBSH2 (Li et al. 2019) enhance node prioritization by integrating heuristic values (h) derived from conflict graphs and dependency graphs, respectively, into the node cost (g). Additionally, Symmetry-Breaking CBS (Li et al. 2021a) and Mutual Exclusion Propagation CBS (Zhang et al. 2022) employ specialized constraints to eliminate symmetric conflicts, exponentially reducing the number of constraint tree (CT) nodes.

Problem Definition

The Multi-Agent Path Finding (MAPF) problem encompasses numerous variants. This work adopts the canonical formulation established in (Stern et al. 2019), which (1) considers vertex conflicts (simultaneous node occupancy) and edge conflicts (bidirectional traversal), (2) incorporates the stay-at-target terminal condition, and (3) optimizes the sum-of-costs metric. This variant serves as the foundation for comparative analysis due to its widespread adoption in theoretical and applied MAPF research.

The MAPF problem is formally defined by a graph $G = (V, E)$ and a set of m agents $\{a_1, a_2, \ldots, a_m\}$. Each agent a_i is assigned an initial vertex $s_i \in V$ and a goal vertex $g_i \in V$. Time is discretized into timesteps, during which agents may either transition to an adjacent vertex or remain stationary. A path p_i for agent a_i is a sequence of vertices $[v_0, v_1, \ldots, v_l]$ satisfying:

- $v_0 = s_i, v_l = g_i$;
- For all $0 \leq t < l$, $(v_t, v_{t+1}) \in E$ or $v_t = v_{t+1}$;
- Agents remain stationary at g_i after reaching it.

The objective is to compute a set of conflict-free paths $\{p_1, \ldots, p_m\}$ that minimizes the sum of individual path lengths $\sum_{i=1}^{m} |p_i|$, where $|p_i|$ denotes the timesteps required for a_i to reach g_i.

Definition 1. *(Conflict) A conflict is either a vertex conflict $\langle a_i, a_j, v, t\rangle$, which arises when agents a_i and a_j are at the same vertex $v \in V$ at the same timestep t, or an edge conflict $\langle a_i, a_j, u, v, t\rangle$, which arises when agents a_i and a_j traverse the same edge$(u, v) \in E$ in opposite directions at the same timestep t.*

In our experimental framework, the graph G is modeled as a 4-connected grid where the vertices represent traversable cells and the edges encode adjacency in the four cardinal directions: north, south, east and west. This design choice is driven by two considerations: (1) 4-connected grids are a standard representation in MAPF research due to their computational efficiency, and (2) they align with practical applications such as video-game navigation systems (Li et al. 2020) and warehouse robotic control systems (Dresner and Stone 2008), where agents operate in discretized planar environments.

Background

Conflict-Based Search

Conflict-Based Search (CBS) (Sharon et al. 2015) is a classic multi-agent path planning optimization algorithm that resolves conflicts between agents through hierarchical search. Its core idea is to decompose the problem into two stages: high-level conflict resolution and low-level path planning, gradually optimizing the path and eliminating conflicts. At the high level, CBS maintains a CT. Each CT node contains a conflict set, a constraint set, a path set, and a cost. The path of each agent must satisfy the constraints in the constraint set. Each agent's path may have some conflicts, which constitute the conflict set. The cost of a CT node is the sum of the time costs of all paths. At the low level, CBS invokes space-time A^* (Silver 2005) (i.e., A^* that searches in the space whose states are vertex-timestep pairs) to find a shortest path for a single agent that satisfies the constraints added by the high-level CT node.

Algorithm 1 presents the pseudo-code for the CBS algorithm. It first generates a CT root node and initializes it, then puts it into an empty priority queue $OPEN$ (Lines 1-2). CBS takes out the CT node N with the smallest cost value in the $OPEN$ queue. If node N does not have a conflict, the solution is found (Lines 4-7). Otherwise, select a conflict in $N.conflicts$ and generate two constraints to resolve the conflict (Lines 8-9). CBS will copy node N to generate two child nodes, then add the two constraints to the two child nodes respectively, and use the low-level A^* to update the path of the agent restricted by the new constraints. If the path does not exist, it will be pruned, otherwise, it will update the child node and insert it into $OPEN$ (Lines 10-20). Repeat the above process until a solution is found or $OPEN$ is empty.

Algorithm 1: CBS Algorithm

Require: A MAPF Instance(G,A)
Ensure: A minimum-cost solution of the MAPF instance.

```
1: Generate root CT node R.
2: Initlize(R);OPEN ← R;
3: while OPEN ≠ ∅ do
4:     N ← a CT node in OPEN with minimum cost.
5:     if N.conflicts = ∅ then return N.paths
6:     end if
7:     conf ← ChooseConfilcts(N)
8:     C1, C2 ← GenerateConstraints(conf)
9:     for i = 1, 2 do
10:        N' ← a copy of N
11:        N'.constraints ← N.constraints ∪ Ci
12:        aj ← the agent on which Ci is imposed.
13:        N'.paths[aj] ← LowlevelSearch(N, aj)
14:        if N'.paths[aj] does not exist then
15:            continue
16:        end if
17:        N'.conflicts ← all conflicts in N'.paths
18:        N'.cost ← SumCost(N'.paths)
19:        OPEN ← OPEN ∪ N'
20:    end for
21: end while
22: return "No Solution"
```

Definition 2. *(Constraint). A constraint is a spatio-temporal restriction introduced by CBS to resolve conflicts. A vertex constraint $\langle a_i, t, v\rangle$ prohibits agent a_i from occupying vertex v at timestep t, and an edge constraint $\left\langle a_i, t, v, v^{'} \right\rangle$ prohibits agent a_i from moving from vertex v to vertex $v^{'}$ between timesteps t and $t+1$.*

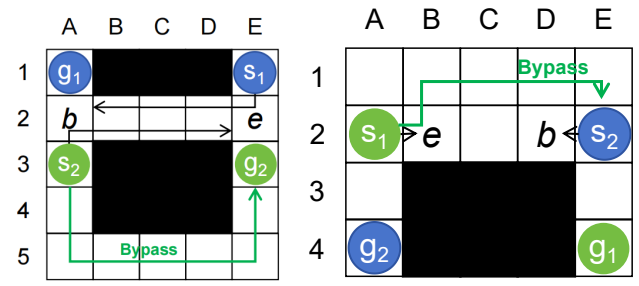
Corridor Symmetry

Definition 3. *(Corridor). A corridor $C = C_0 \cup \{e, b\}$ of graph $G = (V, E)$ is a chain of connected vertices $C_0 \in V$, each of degree 2, together with two endpoints $\{e, b\} \in V$ connected to C_0. The endpoint corresponding to the agent is the one with the farther Manhattan distance from the two endpoints.*

Definition 4. *(Corridor Conflict). Two agents are involved in a corridor conflict iff they traverse the same corridor in opposite directions and have one or more vertex or edge conflicts that occur inside the corridor.*

Definition 5. *(Pseudo Corridor Conflict). Two agents are involved in a pseudo-corridor conflict iff they move in opposite directions, have a vertex or edge conflict, and this conflict cannot be resolved by adding one wait action to either agent before the conflict timestep, as doing so inevitably leads to another vertex or edge conflict.*

Figure 2 (a) shows a corridor of length 4 made up of $C_0 = \{B2, C2, D2\}$, $b = A2$, and $e = E2$. A corridor



(a) an example of a corridor conflict.

(b) an example of a pseudo corridor conflict.

Figure 2. Two-agent MAPF instance with a corridor conflict and a pseudo corridor conflict.

symmetry occurs when two agents attempt to traverse a corridor in opposite directions at the same time. We refer to the corresponding conflict as a corridor conflict.

CBS may be forced to continue branching and exploring irrelevant and suboptimal resolutions of a corridor conflict to compute an optimal solution eventually. As the corridor length k increases, the number of expanded CT nodes grows exponentially as 2^{k+1}.

For pseudo-corridor conflict (as shown Figure 2 (b)), agent $a1$ and $a2$ conflict at point $C3$, and then two child nodes are generated. Both child nodes contain a wait operation, but a conflict still occurs after the wait ends. In this situation, CBS must branch again to find a conflict-free path (i.e., take a bypass).

Essentially, a pseudo corridor conflict functions like a corridor conflict limited to a single segment—comprising just its two endpoints. While these conflicts may appear less severe than traditional corridor conflicts since they avoid exponential growth in CT sizes, they can arise more frequently across various environments due to not being confined to structured corridors.

Resolving Corridor Conflicts

For the corridor symmetry problem, Li et al. (2021a) used a range constraint to solve it. A range constraint $\langle a_i, v, [t_l, t_r]\rangle$ prohibits agent a_i from occupying vertex v between timestep t_l and t_r.

Let $t_i(e)$ for $i = 1, 2$ be the earliest timestep when agent a_i can reach its exit endpoint e. Assume that agent a_i for $i = 1, 2$ has bypasses to reach its exit endpoint e without traversing corridor C, and the earliest timestep when it can reach vertex e using a bypass is $t_i^{'}(e)$. The bypass is defined as the path by which an agent reaches the endpoint without passing through the corridor. If the agent a_i does not have a bypass, $t_i^{'}(e)$ is infinite. If agents a_1 and a_2 have a corridor conflict in the corridor of length k, to resolve it, need to add respective range constraints for a_1 and a_2:

$$\left\langle a_1, b, [0, min(t_1^{'}(b) - 1, t_2(e) + k)]\right\rangle \quad (1)$$

$$\left\langle a_2, e, [0, min(t_2^{'}(e) - 1, t_1(b) + k)]\right\rangle \quad (2)$$

The preservation of Conflict-Based Search (CBS) optimality when incorporating such constraints is formally guaranteed by foundational theoretical work. Li et al. (2021a) introduced a pivotal definition and theorem demonstrating that constraint integration within CBS does not compromise its completeness or cost-optimality.

Definition 6. *(Mutually Disjunctive Constraints) (Li et al. 2021a). Two constraints for two agents a_i and a_j are mutually disjunctive iff any pair of conflict free paths of a_i and a_j satisfies at least one of the two constraints, i.e., there does not exist a pair of conflict-free paths that violates both constraints. Moreover, two sets of constraints are mutually disjunctive iff each constraint in one set is mutually disjunctive with each constraint in the other set.*

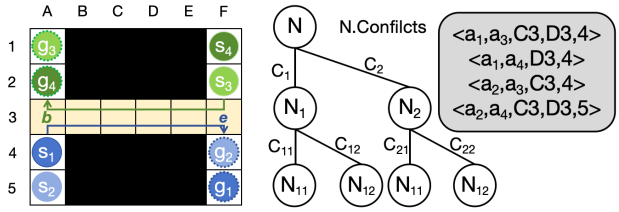
Theorem 1. *Using two constraint sets to split a CT node preserves the completeness and optimality of CBS if the two constraint sets are mutually disjunctive and each of them blocks at least one path in the plan of the CT node (Li et al. 2021a).*

Li et al. (2021a) proved that range constraints (1) and (2) are mutually disjunctive constraints.

Multi-Agent Corridor Conflicts

In this section, we extend corridor symmetry between two agents to multiple agents.

Definition 7. *(Multi-agent corridor conflict). Multiple agents are involved in a multi-agent corridor conflict iff they traverse the same corridor in opposite directions and have more vertex or edge conflicts that occur inside the corridor.*



(a) an example of a multi-agent corridor conflict.

(b) a high-level CT for solving the case in (a).

Figure 3. An example of a multi-agent corridor conflict and its CT .

Let us consider the example in Figure 3, (a) has four agents, s_i is the starting point of a_i, g_i is the goal point of a_i, in the middle is corridor C, and two endpoints e and b. They can be divided into two left and right agent sets A_l and A_r according to the endpoints. It can be seen that without adding constraints, a_1 will conflict with a_3 and a_4, and a_2 will conflict with a_3 and a_4.

There are four conflicts on the root node N. After one split, corridor constraints C_1 and C_3 are added to a_1 and a_3, respectively, generating nodes N_1 and N_2. At this time, we are analyzing that there are still conflicts between N_1, a_2, and a_3, a_4, so we need to split the N_1 node to generate N_{11} and N_{12} nodes. Because CBS needs to choose the

smallest heuristic value every time, we must first choose to split N_2 before we can find the solution to N_{11}. Therefore, this process splits 3 times in total. Based on this example, we can continue to deduce the situation when more agents are involved in multi-agent corridor conflicts, and we can get Theorem 2.

Theorem 2. *We can define the agent set as A_l and A_r according to the endpoint corresponding to the agent. Let's set the size of set A_l to be M_1 and the size of set A_r to be M_2. The number of splits required to resolve multi-agent corridor conflicts is at most $2^{min(M_1,M_2)} - 1$.*

Identifying Multi-agent Corridor Conflicts

Algorithm 2 is the pseudocode for identifying and choosing multi-agent corridor conflicts. First, we need to traverse each conflict to see if it is a corridor conflict, then check each corridor conflict, draw an undirected graph, and regard the agents as nodes. If the conflict is a corridor conflict, they are connected by an edge, and a hash table $Hash$ is maintained, which is the mapping of the two agents to their conflicts (Lines 2-7). If there is no multi-agent corridor conflict, an empty set is returned (Lines 8-9). We also need to separate multiple Multi-agent Corridor Conflicts using union-find, and select the Multi-agent Corridor Conflicts involving the most agents and return them (Lines 10-17).

Algorithm 2: Multi-Agent Corridor Reasoning Function

Require: Constraint Node N.
Ensure: A Conflicts set $conf$.

```
1: function MultiAgentCorridorReasoning(N)
2:     for conf in N.conflicts do
3:         if c is corridor conflict then
4:             AddEdge(conf.a_i, conf.a_j);
5:             Hash(conf.a_i, conf.a_j) ← c
6:         end if
7:     end for
8:     if Hash = ∅ then return ∅;
9:     end if
10:    U ← UnionSearch();
11:    agentset ← agrmax_{u∈U} sizeof(u);
12:    for a_i, a_j in agentset do
13:        if Hash(a_i, a_j) exists then
14:            conf ← conf ∪ {Hash(a_i, a_j)};
15:        end if
16:    end for
17:    return conf;
18: end function
```

Resolving Multi-agent Corridor Conflicts

We consider a corridor of length k with endpoints b and e. Assume that the agent set from point b to point e is A_l, and the set in the opposite direction is called A_r. Moreover, any $a_i \in A_l$ has a corridor conflict with any $a_j \in A_r$, and the agents in A_l and A_r do not have conflicts with any other agents that are not in A_l and A_r. If we use pairwise corridor symmetry reasoning, solving these conflicts needs multiple splits. But this is less efficient, so we propose the multi-agent corridor symmetry reasoning, which can combine these constraints into one split. Next is how we add constraints.

We assume that each $a_i \in A_l (a_j \in A_r)$ has a bypass to reach e (or b) without traversing corridor C. Let t'_i (or t'_j) denote the earliest timestep at which a_i (or a_j) can reach e (or b) using the bypass. If a_i conflicts with a_j, then we add the constraint $\langle a_i, e, [0, min(t'_i(e) - 1, t_j(b) + k)]\rangle$ for a_i, and add the constraint $\langle a_j, b,\ [0, min(t'_j(b) - 1, t_i(e) + k)]\rangle$ for a_j. But a_i may have multiple conflicts for $i = 1, 2, \ldots, |(A_l)|$, we can get a constraint C_i for each $a_i \in A_l$ as follows:

$$C_i = \left\langle a_i, e, [0, max_{a_j \in A_r} min(t'_i(e) - 1, t_j(b) + k))]\right\rangle$$

similarly, we can obtain a constraint C_j for each $a_j \in A_r$:

$$C_j = \left\langle a_j, b, [0, max_{a_i \in A_l} min(t'_j(b) - 1, t_i(e) + k))]\right\rangle$$

We can get two constraint sets $\mathcal{C}_l = \{C_i | a_i \in A_l\}$ and $\mathcal{C}_r = \{C_j | a_j \in A_r\}$ by aggregating all the constraints C_i and C_j ($a_i \in A_l$, $a_j \in A_r$).

Multi-agent Corridor Reasoning Optimality

Based on Theorem 1, we know that if we want to prove that using range constraint set $\mathcal{C}_l$ and range constraint set $\mathcal{C}_r$ will not destroy the optimality of the CBS algorithm, it suffices to prove that $\mathcal{C}_l$ and $\mathcal{C}_r$ are mutually disjunctive.

Theorem 3. *For any path pair (p_i, p_j) of agents $a_i \in A_l$ and $a_j \in A_r$ with multi-agent corridor conflicts, if path p_i violates $\mathcal{C}_l$ while path p_j violates $\mathcal{C}_r$, then these paths must have one or more vertex or edge conflicts within the corridor.*

Proof. Consider two agent sets A_l and A_r that have a multi-agent corridor conflict, which originate from opposite ends of the corridor. Assume that path p_1 of agent $a_1 \in A_l$ violates constraint $\mathcal{C}_l$, path p_2 of agent $a_2 \in A_r$ violates constraint $\mathcal{C}_r$. Specifically, path p_1 only violates the constraint $\langle a_1, e, \max_{a_j \in A_r}(\min(t'_1(e) - 1, t_j(b) + k))\rangle$ in $\mathcal{C}_l$, and p_2 violates the constraint $\langle a_2, b, \max_{a_i \in A_l}(\min(t'_2(b) - 1, t_i(e) + k))\rangle$ in $\mathcal{C}_r$. Then path p_1 of agent a_1 visits endpoint b at timestep $t_1 \in [0, \max_{a_j \in A_r}(\min(t'_1(e) - 1, t_j(b) + k))]$ and path p_2 of agent a_2 visits endpoint e at timestep $t_2 \in [0, \max_{a_i \in A_l}(\min(t'_2(b) - 1, t_i(e) + k))]$. We aim to prove that paths p_1 and p_2 have one or more vertex or edge conflicts inside the corridor. Since

$$\begin{aligned} t_1 &\leq \max_{a_j \in A_r}(\min(t'_1(e) - 1, t_j(b) + k)) \\ &\leq t'_1(e) - 1 \leq t'_1(e) \end{aligned}$$

path p_1 traverses the corridor. Similarly, path p_2 traverses the corridor as well.

Since $t_1 \leq max_{a_j \in A_r}(t_j(b) + k)$, a_1 will arrive at point e earlier than the latest agent in A_r that arrives at point e. Similarly, a_2 will arrive at point b earlier than the latest agent in A_l that arrives at point b. Furthermore, because without adding constraints, each agent in A_l and A_r will have corridor conflicts with each other, thus a_1 will conflict with a_2. Therefore, the property holds. □

Theorem 3 states that if there is no conflict, agents a_i and a_j will not violate their corresponding constraints at the same time. This means that constraints c_i and c_j are mutually disjunctive, and $\mathcal{C}_l$ and $\mathcal{C}_r$ are mutually disjunctive.

CBSH-MACRT Algorithm Process

CBSH-MACRT is based on the CBSH-RCT algorithm, replacing pairwise corridor conflicts with multi-agent corridor conflicts. The algorithm 3 shows the pseudocode. Compared with the CBS, the CBSH-MACRT algorithm needs to calculate the heuristic value for the node (Lines 9-13), then prioritize the conflicts in the CT node N, and then detect which conflicts are paired symmetric conflicts. Finally, call the $MultiAgentCorridorReasoning(N)$ function to detect multi-agent corridor conflicts and select the one involving the most agents, and return its conflict set (Lines 14-16). If there is no corridor conflict, the conflict is selected according to the priority (Lines 17-19). Finally, constraints are added and child nodes are split (Lines 20-22) until a solution is found.

Algorithm 3: CBSH-MACRT Algorithm

Require: A MAPF Instance(G,A)
Ensure: A minimum-cost solution of the MAPF instance.

1: Generate root CT node R.
2: $Initlize(R)$;
3: $OPEN \leftarrow R$;
4: **while** $OPEN \neq \emptyset$ **do**
5: $N \leftarrow$ a CT node in $OPEN$ with minimum cost.
6: $OPEN \leftarrow OPEN \setminus \{R\}$;
7: **if** $N.conflicts = \emptyset$ **then return** $N.paths$;
8: **end if**
9: **if** the heuristic for N has not yet been computed **then**
10: $ComputerHeuristic(N)$;
11: $OPEN \leftarrow OPEN \cup N$;
12: **continue**;
13: **end if**
14: $ConflictPrioritization(N)$;
15: $SymmetryReasoning(N)$;
16: $conf \leftarrow MultiAgentCorridorReasoning(N)$;
17: **if** $conf \leftarrow \emptyset$ **then** $conf \leftarrow ChooseConfilcts(N)$;
18: **end if**
19: $C_1, C_2 \leftarrow GenerateConstraints(conf)$;
20: **for** $i = 1, 2$ **do**
21: *// The process of adding child nodes and constraints is the same as that of the CBS algorithm.*
22: **end for**
23: **end while**
24: **return** "No Solution"

Experiments

Benchmark

Our experimental framework employs 4-connected grid maps from the standardized MAPF benchmark suite (Stern et al. 2019). Because our reasoning technology is targeted at corridors, when we choose a map, we need to choose a map with more corridors. A total of four types of maps, namely maze maps, room maps, warehouse maps and game maps. For the maze map, we ran mazes with map sizes of 32 × 32 and 128 × 128. The maze map has many continuous and dense walls, so there are many pseudo corridor conflicts in the maze map. For room maps and game maps, the maps are divided into small areas, and when agents enter and exit small areas, many corridor conflicts will occur. There are many corridors in the warehouse map, which increases the number of multi-agent corridor conflicts. On these maps, the superiority of our algorithm can be better reflected.

Baselines

This section evaluates two state-of-the-art optimal MAPF solvers as benchmarks: CBSH-RTC and CBSH with Mutex Propagation (CBSH-MS). CBSH-RTC incorporates three symmetry reasoning techniques – Rectangle (R), Target (T), and Corridor (C) reasoning – to optimize conflict resolution. CBSH-MS, an enhanced variant of CBSH, employs mutex propagation to systematically identify cardinal and semi-cardinal symmetric conflicts and resolve them through dual vertex constraint sets.

Experiment results

Our multi-agent corridor reasoning technology can be seamlessly integrated into the CBSH-RTC framework by substituting its corridor reasoning technology with our generalized approach. This adaptation yields an enhanced solver, designated CBSH-MACRT (Multi-Agent Corridor-R-T), which extends conflict resolution capabilities to multi-agent corridor symmetry scenarios.

Map	CBSH-RTC			CBSH-MACRT		
	Node	Cr	Cr(%)	Node	Cr	Cr(%)
maze	519	247	47.5%	**289**	**110**	**38%**
room	1349	668	49.5%	**546**	**169**	**31%**
empty	1610	0.15	0%	1610	0.15	0%
random	2602	43	1.6%	**2548**	**29**	**1%**

Table 1. Corridor conflict distribution for RTC and MACRT.

Success Rates Figure 4 compares the success rates of three algorithms – CBSH-RTC, CBSH-MS, and our proposed method – across eight benchmark maps, measured as the percentage of instances solved within a 100-second runtime limit. CBSH-MACRT performs better than CBSH-RTC and CBSH-MS in all tested maps. Across diverse benchmark maps, our proposed algorithm demonstrates substantial performance gains over both CBSH-RTC and CBSH-MS baselines, achieving peak success rate improvements of 40% and 52%, respectively. This is not only because maps have multi-agent corridor conflicts involving more agents, but another main reason is that the corridor reasoning technology of CBSH-RTC and the mutex propagation of CBSH-MS are both for pairs of agents, while our algorithm can solve multiple pairs of corridor conflicts by splitting the CT node once, which greatly improves the high-level operation efficiency of the algorithm.

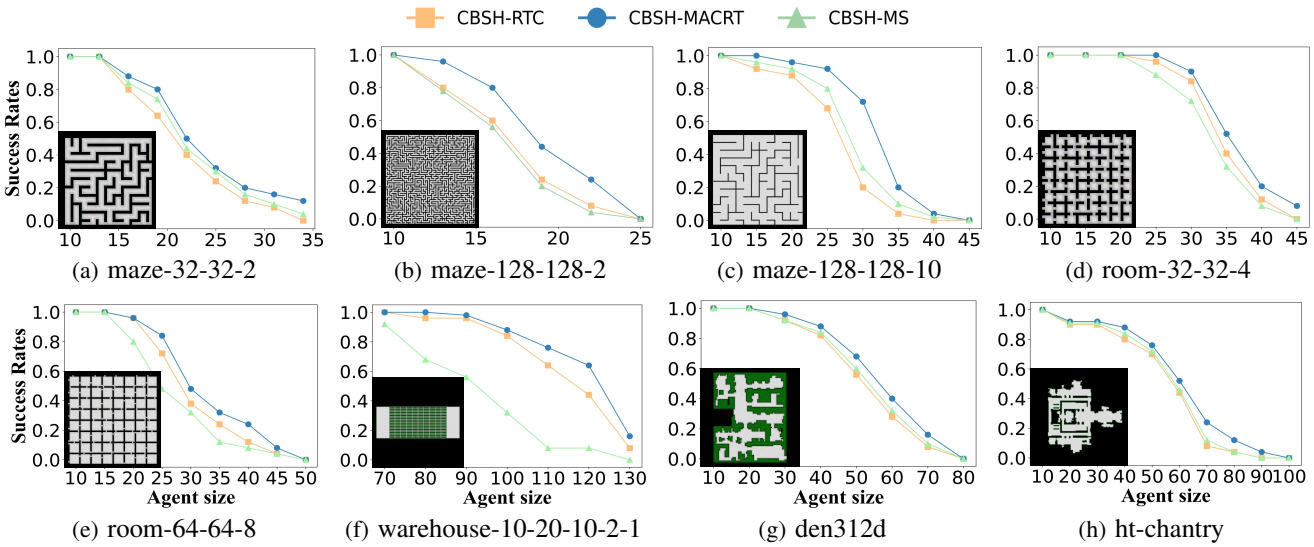


Figure 4. Expertments results on the MAPF benchmark. The relationship between the number of agents and the success rate of solving problems, the x-axis is the number of agents and the y-axis is the success rate and a runtime limit of 100 seconds.

Corridor conflict ratio Tables 1 provide complementary analyses of multi-agent corridor conflict resolution. "Node" represents the number of expanded CT nodes within the time limit. "Cr" and "Cr(%)" denote the number of resolved (multi-agent) corridor conflicts and their percentage of the total conflicts, respectively, throughout this work. Table 1 reports how often MACRT and RTC use (multi-agent) corridor reasoning technique to expand CT nodes, which also indicates how often corridor conflicts occur on different maps. Obviously, for mazes, rooms, and game maps, which have many corridors, the proportion of corridor conflicts is very large. Because our algorithm can effectively solve multiple corridor conflicts with one split, our algorithm can reduce the proportion of corridor conflicts by 9.4-18.6% compared to RTC. However, on maps with few or even no corridor conflicts, such as empty maps and random maps, our algorithm is not much different from the baseline. On the contrary, it will increase the time consumption due to detecting multi-agent corridor conflicts. Therefore, our algorithm will perform better in scenarios with more corridor conflicts.

Runtimes and CT Nodes Table 2 compares the running time and CT size of the CBSH-RTC and CBSH-MACRT algorithms, which further demonstrates the superiority of the multi-agent corridor reasoning technique. In the Table 2, "m" denotes the number of agents involved in each problem instance. The "Ins" column indicates the count of instances successfully solved by both CBSH-RTC and CBSH-MACRT algorithms. Concurrently, the "Runtime(s)" and "CT Nodes" columns report the average solution time (in seconds) and the average number of nodes in the CT generated across these jointly solved instances, respectively. Table 2 benchmarks CBSH-MACRT against CBSH-RTC through runtime and CT node metrics on commonly solved instances. CBSH-MACRT demonstrates superior efficiency, reducing runtime by 14–67% and CT nodes by 14–78% on the four type maps, which means our algorithm greatly reduces running time and improves computing efficiency.

Conclusions

This paper presents a multi-agent corridor reasoning technique to address symmetry conflicts arising between multiple agents in CBS for MAPF. We rigorously demonstrate that this technique preserves CBS's solution optimality while significantly reducing high-level CT node splits. Experimental validation shows these reductions directly correlate with a 8-40% improvement in success rates and a 14–67% acceleration in planning times compared to baseline CBS implementations.

Map	m	Ins	Runtime(s)		CT Nodes	
			RTC	MACRT	RTC	MACRT
Maze	10	50	1.8	**1.3**	15.7	**8.2**
	20	46	22.5	**8.4**	267	**59**
	30	20	14.2	**12.8**	340	**232**
	40	4	24.9	**20.9**	577	**298**
Room	10	50	0.7	**0.6**	9	**6**
	20	46	6.7	**5.9**	188	**66.5**
	30	22	17.4	**8.6**	1269	**269.5**
	40	4	30.5	**20**	1284	**375**
Ware-	90	48	15	**12.6**	267	**229**
house	100	42	22.2	**16.3**	327	**249**
	110	32	26.2	**20.6**	428	**330**
	120	22	39.6	**29.1**	661	**446**
Den-	30	46	1.9	**1.2**	77.9	**38**
312d	40	42	9.7	**4.4**	586.5	**145**
	50	24	15.1	**7.7**	672	**346**
	60	10	38.1	**21.1**	2230	**683**

Table 2. Experiment Results on four maps.

References

Barer; Max; 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 International Symposium on Combinatorial Search*, volume 5, 19–27.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Betzalel, O.; Tolpin, D.; and Shimony, E. 2015. Icbs: The improved conflict-based search algorithm for multi-agent pathfinding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, 223–225.

Dresner, K.; and Stone, P. 2008. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence research*, 31: 591–656.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. S.; and Koenig, S. 2018. Adding heuristics to conflict-based search for multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 83–87.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced partial expansion A. *Journal of Artificial Intelligence Research*, 50: 141–187.

Gómez, R. N.; Hernández, C.; and Baier, J. A. 2020. Solving sum-of-costs multi-agent pathfinding with answer-set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9867–9874.

Hönig, W.; Preiss, J. A.; Kumar, T. S.; Sukhatme, G. S.; and Ayanian, N. 2018. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics*, 34: 856–869.

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2022. Branch-and-cut-and-price for multi-agent path finding. *Computers & Operations Research*, 144: 105809.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 10256–10265.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *Proceedings of International Joint Conference on Artificial Intelligence*, volume 2019, 442–449.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021a. Pairwise symmetry reasoning for multi-agent path finding search. *Artificial Intelligence*, 301: 103574.

Li, J.; Ruml, W.; and Koenig, S. 2021. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 35, 12353–12362.

Li, J.; Sun, K.; Ma, H.; Felner, A.; Kumar, T.; and Koenig, S. 2020. Moving agents in formation in congested environments. In *Proceedings of the International Symposium on Combinatorial Search*, volume 11, 131–132.

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

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219: 40–66.

Silver, D. 2005. Cooperative pathfinding. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, volume 1, 117–122.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 24, 173–178.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 151–158.

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *Proceedings of the twenty-second european conference on artificial intelligence*, 810–818.

Wagner, G.; and Choset, H. 2015. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219: 1–24.

Wang, J.; Li, J.; Ma, H.; Koenig, S.; and Kumar, S. 2019. A new constraint satisfaction perspective on multi-agent path finding: Preliminary results. In *18th International Conference on Autonomous Agents and Multi-Agent Systems*, 2253–2255.

Wang, Y.; Duhan, T.; Li, J.; and Sartoretti, G. 2025. LNS2+ RL: Combining multi-agent reinforcement learning with large neighborhood search in multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 23343–23350.

Yu, J.; and LaValle, S. M. 2016. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32: 1163–1177.

Zhang, H.; Li, J.; Surynek, P.; Kumar, T. S.; and Koenig, S. 2022. Multi-agent path finding with mutex propagation. *Artificial Intelligence*, 311: 103766.