한국정밀공학회지 제 43 권 제 2 호 pp. 123-131 February 2026 / 123
J. Korean Soc. Precis. Eng., Vol. 43, No. 2, pp. 123-131 http://doi.org/10.7736/JKSPE.025.044
ISSN 1225-9071 (Print) / 2287-8769 (Online)

혼잡한 환경에서 에이전트 병합 및 분할을 이용한 다중 에이전트 경로 탐색 개선에 관한 연구

A Study on Improving Multi-agent Path Finding in Congested Environments Using Agent Merging and Splitting

유서현[1], 임성택[1], 강효재[1], 정찬희[2], 한대희[1], 강민성[3,#]
SeoHyun Yoo[1], SeongTaek Im[1], HyoJae Kang[1], ChanHee Jeong[2], DaeHee Han[1], and Min-Sung Kang[3,#]

1 한양대학교 융합로봇시스템학과 (Department of Interdisciplinary Robot Engineering Systems, Hanyang University)
2 한양대학교 로봇공학과 (Department of Robot Engineering, Hanyang University)
3 한양대학교 스마트융합공학부 (School of Smart Convergence Engineering, Hanyang University)
Corresponding Author / E-mail: wowmecha@hanyang.ac.kr, Tel: +82-31-400-5961
ORCID: 0000-0002-8459-5843

KEYWORDS: Multi-agent path finding (다중 에이전트 경로 탐색), Collision based search (충돌 기반 탐색), Path planning (경로 탐색), Robot navigation (로봇 네비게이션)

The rising demand for robots in warehouses has highlighted the need for efficient multi-robot algorithms. In response, researchers have focused on Multi-Agent Path Finding (MAPF), which enables multiple agents to calculate conflict-free paths to their individual goals. However, the computation time of conflict-based MAPF algorithms significantly increases as the number of conflicts rises, a common challenge in warehouse environments with narrow passages or corridors. To tackle this issue, this study introduces a new type of conflict called "Overlap Conflict." Overlap Conflicts occur when an agent stops, causing chain conflicts among subsequent agents traveling in the same direction. When an Overlap Conflict arises, the affected agents are dynamically merged into a single group, shifting the conflicts from an individual level to a group level. If the merged agents find themselves with unreachable goals, they are split back into individual agents to continue calculating paths to their respective destinations. This approach effectively reduces computation time in congested environments, particularly in narrow corridors where alternative routes exist.

Manuscript received: March 24, 2025 / Revised: October 23, 2025 / Accepted: October 30, 2025
This paper was presented at KSPE Autumn Conference in 2024

1. Introduction

As global robot supply chains become increasingly complex and consumer demand diversifies, relying solely on traditional multi-agent distribution methods are no longer sufficient to guarantee the efficiency of logistics operations. In response, the adoption of robot-assisted warehouses, as shown in Fig. 1, is gaining importance. In line with this trend, research of Multi-Agent Path Finding (MAPF) is being vigorously conducted. This algorithm aims to find non-conflicting paths for multiple agents to reach their respective destinations within shared environments.

MAPF has been actively studied from various perspectives. Algorithms that merge agents [1,2] or assign priorities among agents [3-5] have attempted to reduce computation time. However, assigning priorities to agents has the drawback of not guaranteeing the optimality of the solution. Other algorithms have aimed to find an optimal solution by resolving conflicts between agents, but the computation time swells exponentially as the number of conflicts grows [6-9]. To address this, some algorithms have defined additional conflict types or proposed new constraints to reduce

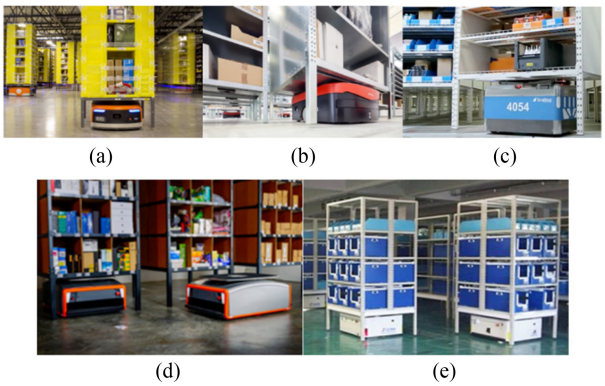


Fig. 1 Logistics companies using robots [10] (a) Amazon Robotics, (b) Swisslog, (c) Scallong, (d) Grey Orange, and (e) Hitachi (Adapted from Ref. 10 on the basis of OA)

computation time and explore suboptimal solutions [11-15]. However, even these approaches have not sufficiently mitigated the exponential increase in computation time caused by conflicts in congested environments.

In warehouses, corridors or passageways are often designed to be narrow to maximize space utilization, which frequently forces multiple agents to share the same route. When a leading agent unexpectedly stops in such restricted spaces, it can trigger a chain of conflicts among following agents. These chain conflicts are considerably more complex than isolated conflicts, leading to significant delays in re-planning routes, particularly in congested environments. To address this challenge, we propose a new algorithm, OverlapCBS, which introduces a new type of conflict, termed the Overlap Conflict, specifically designed to handle these chain conflicts. OverlapCBS reduces computation time by dynamically merging and splitting agents based on their paths when conflicts occur, thereby efficiently resolving conflicts in such environments. This approach aims to overcome the limitations of previous methods and improve overall efficiency in warehouse logistics.

2. Theoretical Background

2.1 Problem Definition

This study refines the Conflict-Based Search (CBS) algorithm [6] as its fundamental framework. The MAPF problem [16,17] is represented by a graph $G = (V, E)$ and a set of k agents. V represents locations, each of which can be exclusively occupied by a single agent and E stated the connections between two vertices. Each agent is assigned a *Task,* defined as a tuple $\{start_i, goal_i\}$ consisting of a unique start state and goal state, where an agent a_i has starti $start_i \in V$ and $goal_i \in V$. Time is assumed to be discretized, and at each timestep, an agent can take a single action: either *moving* or *waiting*. The solution to the MAPF problem consists of a set of conflict-free paths, denoted as $\Pi = \{\pi_1, \pi_2, ..., \pi_k\}$, where k means the number of agents and π_k denotes the path of a_k. Each path is associated with a sequence of tuples $\{vertex, timestep\}$. In the MAPF problem, there are two types of conflicts: *vertex conflicts* and *edge conflicts*. *Vertex conflicts* occur when two agents a_i and a_j occupy the same vertex v simultaneously, which is formally defined as $\{a_i, a_j, v, t\}$. *Edge conflicts* occur when two agents a_i and a_j swap vertices through the same edge, at the same time t, which is denoted as $\{a_i, a_j, v_i, v_j, t\}$. All paths satisfy the following conditions: 1. every agent starts at its predefined initial state and reaches its goal state, 2. no *vertex conflicts* or *edge conflicts* exist among the paths. In this study, we utilize the sum of costs as the objective function to evaluate solutions and aim to minimize it.

2.2 Conflict Based Search Algorithm (CBS)

CBS searches for paths for each agent and detects conflicts between agents' paths. When a conflict occurs, it generates constraints to resolve the conflict and modifies the paths accordingly. A *vertex constraint* prohibits agent a_i from occupying vertex v at time t, denoted as $\{a_i, v, t\}$. An *edge constraint* prohibits agent a_i from moving from vertex v_i to v_j at time t, denoted as $\{a_i, v_i, v_j, t\}$. CBS operates on two search levels. The high-level is responsible for identifying and resolving conflicts between agents by comparing their individual paths. This process is managed using a binary tree structure known as the Constraint Tree (CT), in which each node consists of three key components.

1) *N.constraints*
   A set of constraints, each of which illustrates constraints for a single agent.
2) *N. solution*
   A set of paths, each consisting of the individual agent's path that complies with the constraints applied to that node.
3) *N. cost*
   The total cost is calculated using an objective function. The high-level search utilizes this cost as a metric to prioritize nodes, aiming to find a solution that minimizes the chosen cost while avoiding conflicts.

The root node of the CT starts with an empty set of constraints. When a conflict is detected in the *N. solution*, the high-level search generates a new constraint and two child nodes, each inheriting the constraints of its parent and incorporating the newly generated constraint, respectively. These child nodes resolve the conflict by enforcing different constraints on each agent involved. The low-level search computes paths for individual agents while adhering to the given constraints. In this study, we apply the A^* algorithm as the fundamental search method for the low-level search. The CBS

algorithm continues this process until a conflict-free *N. solution* is found, resulting in a goal node in the CT that contains a valid set of paths for all agents.

3. Methodology

3.1 Overlap Conflict Definition

To effectively address scenarios in congested environments, this paper proposes a new type of conflict, termed the Overlap Conflict. This concept enables the efficient handling of chain conflicts that occur when multiple agents move through the same narrow corridors in the same direction. The traditional vertex conflict is further categorized into Single Vertex Conflict and Overlap Conflict, based on whether the paths of two agents continue to overlap following an initial conflict. This study introduces three types of conflicts: Single Vertex Conflict, Overlap Conflict, and Edge Conflict. The Edge Conflict is defined based on CBS, while the definitions of Single Vertex Conflict and Overlap Conflict are provided below.

1) Single Vertex Conflict

A Single Vertex Conflict is defined as a tuple $\{a_i, a_j, v, t, S\}$. It represents a conflict that occurs when agents a_i and a_j occupy vertex v at the same time t. S is a boolean variable used to distinguish it from an Overlap Conflict. For instance, Fig. 2(a) illustrates a Single Vertex Conflict where agents a_1 and a_2 occupy vertex $B2$ at the same time t_0. It is represented as the tuple $\{a_1, a_2, B2, t_0, False\}$.

2) Overlap Conflict

An Overlap Conflict is defined as a tuple $\{a_i, a_j, v, t, O\}$. It represents a conflict that occurs when agents a_i and a_j occupy vertex v at the same time t_0. O is a boolean variable used to distinguish it from a Single Vertex Conflict. For instance, Fig. 2(b) illustrates an Overlap conflict that occurs when agents. a_1 and a_2 successively occupy vertex $B2$ at times t_0. It is repsresented as the tuple $\{a_1, a_2, B2, t_0, True\}$.

3.2 Overlap Constraint Definition

To resolve conflicts between agents, appropriate constraints for each conflict type are required. The Single Vertex Constraint and Edge Constraint follow the same definitions as those in CBS. The Overlap Constraint is defined as a tuple $\{a_i, a_j, t\}$ where a_i is designated as the head agent, a_j as the tail agent, treating the two agents as a single merged agent set starting from time t. A merged agent set moves together while maintaining a one-timestep distance, where timestep distance refers to the minimum number of timesteps required for an agent to move from its current vertex to another agent's vertex. For instance, as illustrated in Figs. 3(a) and 3(b), if two agents are located in adjacent vertices, they have a one-timestep distance. In contrast, as shown in Fig. 3(b), the agents have a 2-timestep distance, as agent 1 takes two-timesteps to reach the vertex where Agent 2 is located.

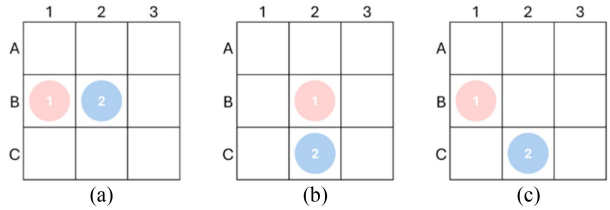


Fig. 2 Types of conflicts utilized in this paper. (a) Single Vertex Conflict: A conflict occurs at a single vertex, specifically at vertex $B2$ and (b) Overlap Conflict: A conflict spans multiple vertices. In this example, the two agents occupy vertex $B2$ and $B3$ simultaneously

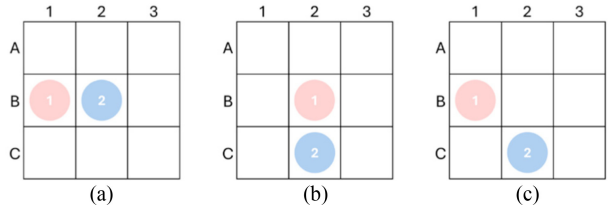


Fig. 3 Types of conflicts utilized in this paper. (a) Single Vertex Conflict: A conflict occurs at a single vertex, specifically at vertex $B2$ and (b) Overlap Conflict: A conflict spans multiple vertices. In this example, the two agents occupy vertex $B2$ and $B3$ simultaneously

Each agent can have at most one head agent and at most one tail agent. For instance, consider a merged set of three agents, as depicted in Fig. 3(a). Assuming that Agent 1 is positioned at the front in the direction of movement, Agent 1 has no head agent and designates Agent 2 as its tail agent. Agent 2 has Agent 1 as its head agent and Agent 3 as its tail agent, while Agent 3 has Agent 2 as its head agent and no tail agent. In contrast, configurations where an agent is associated with more than one head agent (as shown in Fig. 3(b)) or more than one tail agent (as shown in Fig. 3(c)) are not permitted.

3.3 State Definition

A state is defined as a tuple $\{vertex, time, head, tail\}$, where each component represents the current status of an agent and its dependency relationships with other agents. Specifically, *vertex* denotes the graph vertex at which the agent is located at *time*; *time* is the discrete timestep corresponding to that state, indicating when the agent occupies the vertex; *head* represents the head agent that the current agent is following, thereby highlighting the dependency relationship; and *tail* denotes the tail

agent that is following the current agent, indicating that the current agent acts as a leader. This structure not only captures the agent's current position and time but also explicitly defines the dependency relationships between agents.

3.4 High Level Search of OverlapCBS

Algorithm 1 describes the high-level search process in OverlapCBS. The basic flow of the algorithm follows that of CBS (Lines 1-8). However, OverlapCBS generates two distinct constraints for each left and right nodes, denoted as C_l and (Line 10), to resolve a detected conflict. These constraints are defined for the conflicting agents or agent sets and may incorporate constraints for multiple agents, depending on the situation. Based on the generated C_l and C_r, new CT nodes are created, adding the corresponding constraints to each node (Line 14). Subsequently, the low-level solver recalculates the paths for the individual agents subject to these constraints. Once the paths for all agents in the constraint lists are recalculated, the cost of the solution is computed, and the success of the solution is evaluated (Line 18). If a valid solution is found, the corresponding node is added to the OPEN set (Line 19). This process repeats until a node is retrieved from the OPEN set whose paths are conflict-free. In that case, the algorithm returns the solution (Line 8) and terminates.

Algorithm 1: High Level Search of OverlapCBS

```
   Input: MAPF instance
   Output: Solution to the MAPF problem
1  Root. constraints ← Ø
2  Root. solution ← find individual paths by the low level( );
3  Root. cost ← get_cost(Root. solution);
4  insert Root to OPEN
5  while OPEN is not empty do
6    P ← best node from OPEN // lowest solution cost
7    if P has no conflict then
8      return P. solution //P is the goal
9    C_f ← get_first_conflict(P);
10   C_l, C_r ← create_constraints_from_conflict(C_f);
11   foreach constraints list C_i in {C_l, C_r} do
12   A ← new node;
13   foreach agent a_i in C_i do
14   A. constraints ← P. constraints + C_i {a_i};
15  A. solution ← P. solution;
16     Update A. solution (invoking low level(a_i));
17  A. cost ← get_cost (A. solution);
18     if A. cost < ∞ then
19       Insert A to OPEN;
```

3.4.1 Find First Conflict

It is necessary to first generate a path for a single agent and then compare the paths between agents to identify conflicts. Algorithm 2, *get_first_conflict*(), identifies and returns a conflict from a CT node solution by comparing two agents' paths. For each timestep t from 0 to max_t, the algorithm checks the agent's positions, where cv (current vertex) is the vertex at t and nv (next vertex) is the adjacent vertex the agent moves to (Lines 5-8).

Algorithm 2: get_first_conflict( )

```
   Input: CT node N
   Output: Conflict
1  max_t = find_max_time (N. solution);
2  foreach agent a_i, a_j do
3   t = 1;
4     while t < max_t do
5     cv_i ← get_current_vertex(a_i, t);
6     cv_j ← get_current_vertex(a_j, t);
7     nv_i ← get_next_vertex(a_i, t);
8     nv_j ← get_next_vertex(a_j, t);
9       if cv_i == cv_j and nv_i ! = nv_j then
10        return {a_i, a_j, cv_i, t, 0}// single vertex conflict
11      else if cv_i == cv_j and nv_i = nv_j then
12        return {a_i, a_j, cv_i, t, 1}// overlap conflict
13      else if cv_i == nv_j and nv_i = cv_j then
14        return {a_i, a_j, cv_i, nv_i, t}// edge conflict
15 t++
```

3.4.2 Create Constraint from Conflict

Once conflicts among agents are identified, appropriate constraints must be generated to resolve them. For single-agent pair conflicts, constraints are generated as in CBS. However, in OverlapCBS, conflicts may occur between merged agent sets (e.g., Fig. 4), requiring constraints for the entire set without disconnecting its agents. All of the examples in Fig. 4 are designated the pink agent set as agent-set 1 and the blue agent set as agent-set 2.

1) Single Vertex Conflict between head agents

   Fig. 4(a) illustrates a scenario in which a single vertex conflict occurs between the agent-set 1 and the agent-set 2 at timestep 0. In this situation, constraints are generated without breaking the internal connections within the agent sets. When constraints are applied to agent-set 1, two constraints, {1, C_4, 1}, and {4, C_4, 2}, are imposed on agent 4, leading to the movement pattern shown in Fig. 5(b).

2) Overlap Conflict between head agents

   Fig. 4(b) illustrates a scenario where an overlap conflict

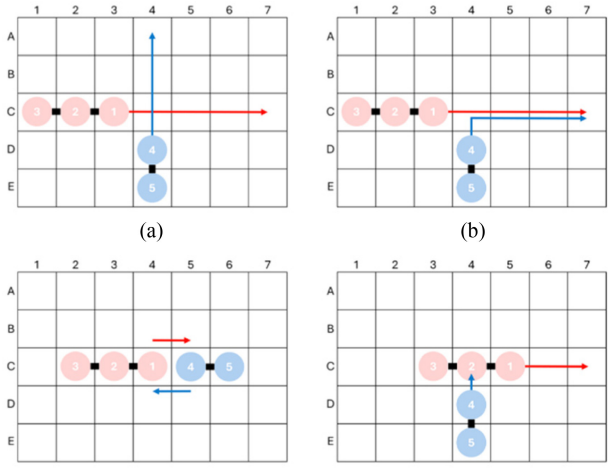


Fig. 4 Examples of conflict scenarios between agent sets: (a) Single vertex conflict, (b) Overlap conflict, (c) Edge conflict, and (d) single vertex conflict between a head agent and a middle agent

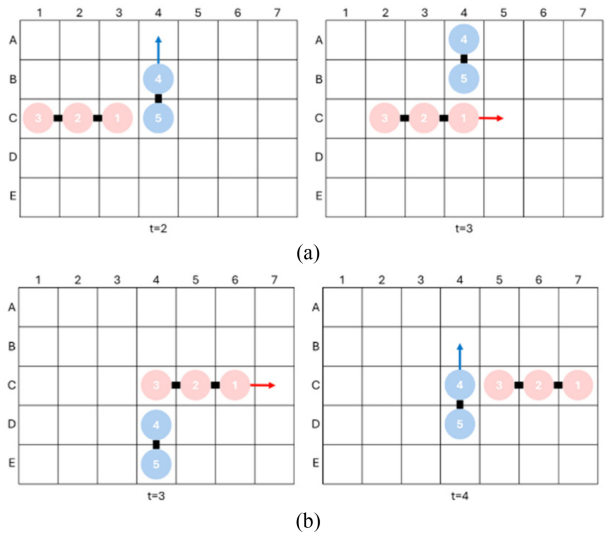


Fig. 5 Movement patterns resulting from single vertex conflict resolution. (a) When constraints are applied to agent-set 1 and (b) When constraints are applied to agent-set 2

occurs between the agent-set 1 and the agent-set 2. In this situation, constraints are generated without breaking the internal connections within the agent sets. When constraints are applied to agent-set 1, two single vertex constraints, {1, C_4, 1} and {1, C_4, 2}, along with one overlap constraints, {4, C_4, 1}, {4, C_4, 2}, and {4, C_4, 3}, along with one overlap constraint, {3, 4, 3}, are imposed on agent 4, resulting in the movement pattern shown in Fig. 6(b).

3) Edge Conflict between head agents

Fig. 4(c) illustrates a scenario where an edge conflict occurs between the agent-set 1 and the agent-set 2. In this case, constraints are also generated without breaking the connections

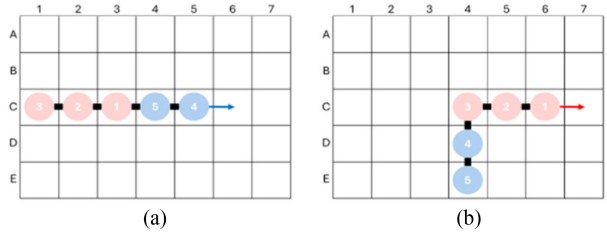


Fig. 6 Movement patterns resulting from overlap conflict resolution: (a) Constraints applied to agent-set. (b) Constraints applied to agent-set 2

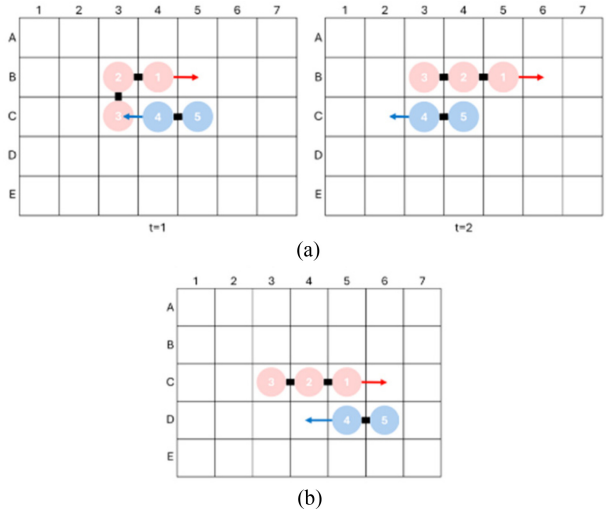


Fig. 7 Movement patterns resulting from edge conflict resolution. (a) Constraints on agent-set 1 and (b) Constraints on agent-set 2

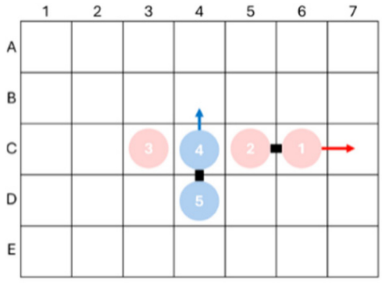


Fig. 8 A scenario where applying constraints to agent-set 1 breaks the internal connection of the agent-set 1, making it an invalid solution

within the agent sets. When constraints are applied to agent-set 1, and a vertex constraint {2, C_4, 1} is imposed on agent 2. This process involves sequentially inspecting conflicts for each agent in the agent set and generating constraints for each detected conflict. Applying such constraints to agent-set 1 results in the movement pattern shown in Fig. 7(a). Conversely, when constraints are applied to agent-set 2, and edge constraint {4, C_5, C_4, 1} is imposed on agent 4, and a vertex constraint {5, C_5, 1} is imposed on agent 5, resulting in the movement pattern shown in Fig. 7(b).

4) Single Vertex Conflict between head agent and middle agent
As shown in Fig. 4(d), a conflict can occur between the middle agent of the pink agent set (agent-set 1) and the head agent of the blue agent set (agent-set 2). In this case, to avoid breaking the connections within the agent sets, constraints are applied only to agent-set 2. When constraints are applied to agent-set 2, a single vertex constraint {4, C_4, 1} is imposed on agent 4, resulting in a movement pattern similar to that shown in Fig. 5(b). However, if constraints are applied to agent-set 1, it would break the connections within agent-set 1, as shown in Fig. 8. Therefore, no constraints are generated for agent-set 1 in this scenario.

3.5 Low Level Search for OverlapCBS

The algorithm 3 represents a modified low-level search process for CBS, incorporating the ability to follow a designated head agent whenever applicable. If no head agent exists to follow, the algorithm operates identically to the standard CBS low-level search. However, if a head agent is present, the agent prioritizes staying directly behind the head agent. If following directly is not possible, the agent attempts to move adjacent to the head agent. If neither of these options is feasible, the algorithm disconnects the head-tail relationship, effectively splitting the agent set.

To determine the path of a merged agent set, the pathfinding process must proceed sequentially, starting with the head agent

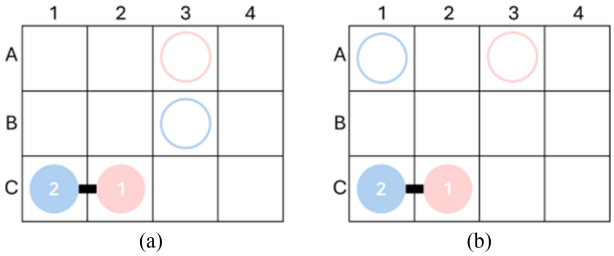


Fig. 9 Example scenarios in which agents within the same agent set move towards their respective goal locations

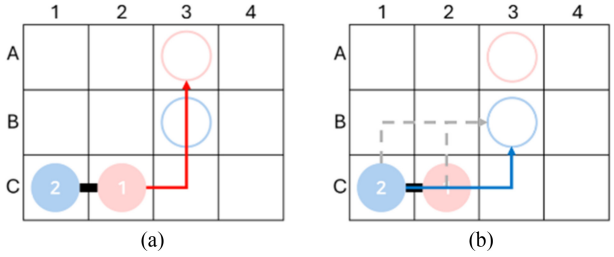


Fig. 10 Path planning process for a merged agent set. (a) The head agent plans its path to the goal first, (b) The tail agent plans its path to follow the head agent as closely as possible

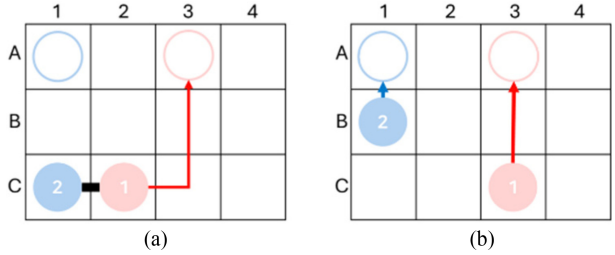
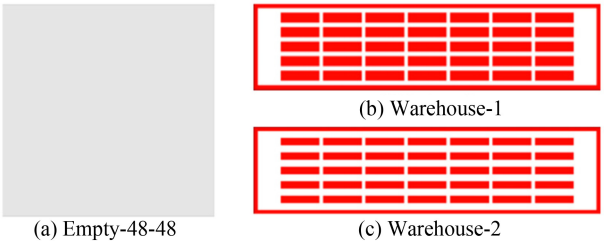


Fig. 11 Agent-set splitting due to increased cost for the tail agent

Algorithm 3: Low Level Search Algorithm

Input: start vertex *S*, goal vertex *G*, constraints *Cons*, solution *Sol*
Output: *Path* from *S* to *G* or failure

open ← {(*S*, 0 + *heuristic*(*S*))};
closed ← ϕ
g[*S*] ← 0;
head = -1;
initialize *bh*, *nh*;
while *OPEN is not empty* **do**
  if *bh is not null* **then**
    current ← *bh*;
  elseif *nh is not null* **then**
    current ← *nh*;
  else *bh is not null* **then**
    current ← *node in open with the lowest f*-score;
    head = -1;
  if *current* == *G* **then**
    return Path from *S* to *G*;
  Remove *current* from *open* and add *current* to *closed*;
  head = *update_head_form_constraint*(*Cons*)
  neighbor_list ← *get_neighbors*(*current*)
  foreach *state n in neighbor_list* **do**
    if *n in closed* **then**
      Continue;
  tentative_g ← *g*[*current*] + *cost*(*current*, *n*)
  if not *in open* **or** *tentative_g* < *g*[*n*] **then**
    g[*n*] ← *tentative_g*;
  f[*n*] ← *g*[*n*] + *heuristic*(*n*);
  parent[*n*] ← *current*;
    if *n not in open* **then**
      Add *n* to *open*;
    vertex$_h$ = *get_current_vertex*(*head, current.time*)
    if *n.vertex* == *vertex*$_h$ **then**
      bh = *n*
    else if *dist*(*n.vertex*, *vertex*$_h$) == 1 ***then***
  nh = *n*
return failure; // No valid path exists



(a) Empty-48-48 (b) Warehouse-1 (c) Warehouse-2

Fig. 12 Grid map for evaluation

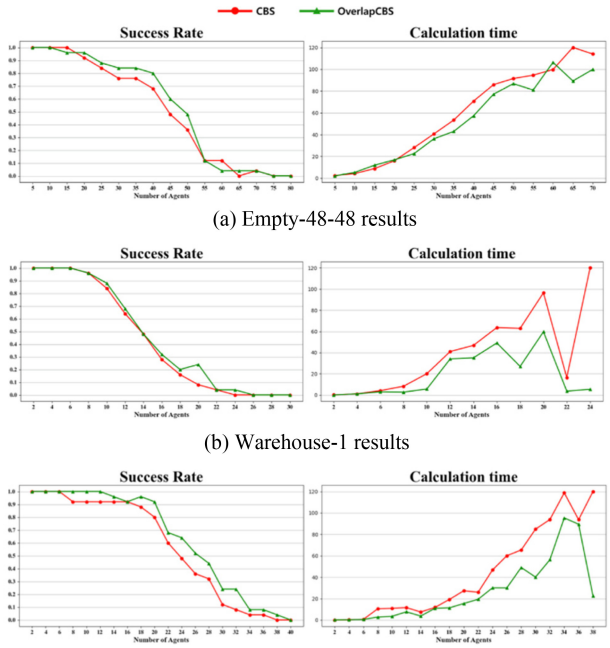
then moving to the tail agent. This can be illustrated using the scenario in Fig. 9(a). In the figure, agent 1 is the head agent, and agent 2 is the tail agent. First, the path for the head agent is calculated, as shown in Fig. 10(a). Next, the path for the tail agent is planned to follow the head agent's path as closely as possible, as depicted in Fig. 10(b). However, as shown in Fig. 9(b), there are cases where the tail agent cannot reach its goal location if it continues to follow the head agent. In such situations, as illustrated in Fig. 11, the agent set is split at the point where the tail agent's cost increases due to following the head agent. From that point onward, the tail agent independently plans its path as a single agent.

4. Experiments

4.1 Evaluation Setup

To evaluate the performance of the proposed algorithm, we conducted comparative experiments with the CBS algorithm using the three maps shown in Fig. 12. In each environment, 25 scenarios were generated by randomly assigning start and goal positions to the agents, and the experiments were conducted while progressively increasing the number of agents. The specific characteristics of each environment are as follows:

1) Empty-48-48
   This is a 48 × 48 grid map with no obstacles, representing an open space. Overlap conflicts are expected to occur infrequently, making it suitable for evaluating the algorithm under general conditions.
2) Warehouse-1
   This is a 90 × 18 warehouse map with a single narrow corridor. Due to its narrow pathways, Overlap conflicts are expected to occur frequently. It evaluates the algorithm in environments with frequent conflicts and no alternative routes.
3) Warehouse-2
   This is a 90 × 24 warehouse map with double corridors. Similar to Warehouse-1, Overlap conflicts are expected to occur frequently, but alternative routes are available, allowing evaluation under such conditions.



(a) Empty-48-48 results

(b) Warehouse-1 results

(c) Warehouse-2 results

Fig. 13 Evaluation results. (a) Empty-48-48, (b) Warehouse-1, and (c) Warehouse-2

The algorithm's performance is evaluated using two metrics: Success rate and Calculation time. Success rate represents the percentage of scenarios where pathfinding is completed within 120 seconds. Calculation time is the average time taken for scenarios where either the CBS or OverlapCBS algorithm succeeds, with failed cases assigned 120 seconds.

4.2 Evaluation Results and Analysis

Experimental results clearly show that OverlapCBS significantly outperforms traditional CBS. In the Empty-48-48 map (Fig. 12(a)), OverlapCBS achieves similar success rates as CBS when fewer than 30 agents are used; however, when the number of agents exceeds 30, OverlapCBS achieves a higher success rate and, with more than 40 agents, it computes solutions approximately 20% faster than CBS. In the Warehouse-1 map (Fig. 12(b)), which features a single narrow corridor, OverlapCBS exhibits a success rate that is 15% higher than that of CBS with 20 agents and maintains more stable computation times. In the Warehouse-2 map (Fig. 12(c)), characterized by double corridors, OverlapCBS achieves up to 30% higher success rates and completes computations about 33% faster than CBS in scenarios with 20 or more agents. These numerical improvements confirm that OverlapCBS is a superior solution for Multi-Agent Path Finding in congested environments.

5. Conclusion

The experimental results indicate that the proposed OverlapCBS algorithm outperforms traditional CBS in terms of both success rate and computation time, particularly in environments with narrow corridors and alternative routes. This improvement is primarily attributed to OverlapCBS's ability to effectively handle overlap conflicts by merging conflicting agents into unified sets. By shifting conflict detection from the individual agent level to the group level, OverlapCBS efficiently addresses chain conflicts and overcomes the limitations of previous methods in congested environments, thereby offering a robust solution for Multi-Agent Path Finding (MAPF) in complex scenarios such as warehouses. However, the algorithm shows relatively less improvement in settings without alternative routes (e.g., single narrow corridors) compared to environments such as Warehouse-2, where alternative paths are available. This limitation likely stems from the current inability to effectively resolve conflicts caused by agents moving in opposite directions under such conditions. Additionally, in open environments where overlap conflicts are infrequent, the potential impact of OverlapCBS is diminished.

To address these challenges, future work should focus on adapting additional conflict types, as defined in prior research [13], into the OverlapCBS framework. This integration aims to improve the algorithm's adaptability and performance across a broader range of scenarios, ultimately enhancing its robustness and real-world applicability for MAPF applications.

REFERENCES

1. Sharon, G., Stern, R., Felner, A., Sturtevant, N., (2012), Meta-agent conflict-based search for optimal multi-agent path finding, Proceedings of the International Symposium on Combinatorial Search, 97-104.
2. Pei, M.-S., Liu, C.-L., (2023), Multi-agent path planning based on meta-agent conflict merge search algorithm, Proceedings of the 5th International Academic Exchange Conference on Science and Technology Innovation, 87-90.
3. Ma, H., Harabor, D., Stuckey, P. J., Li, J., Koenig, S., (2019), Searching with consistent prioritization for multi-agent path finding, Proceedings of the AAAI Conference on Artificial Intelligence, 7643-7650.
4. Kasaura, K., Nishimura, M., Yonetani, R., (2022), Prioritized safe interval path planning for multi-agent pathfinding with continuous time on 2D roadmaps, IEEE Robotics and Automation Letters, 7(4), 10494-10501.
5. Okumura, K., Machida, M., Défago, X., Tamura, Y., (2022), Priority inheritance with backtracking for iterative multi-agent path finding, Artificial Intelligence, 310, 103752.
6. Sharon, G., Stern, R., Felner, A., Sturtevant, N. R., (2015), Conflict-based search for optimal multi-agent pathfinding, Artificial Intelligence, 219, 40-66.
7. Andreychuk, A., Yakovlev, K., Surynek, P., Atzmon, D., Stern, R., (2022), Multi-agent pathfinding with continuous time, Artificial Intelligence, 305, 103662.
8. Felner, A., Li, J., Boyarski, E., Ma, H., Cohen, L., Kumar, T. S., Koenig, S., (2018), Adding heuristics to conflict-based search for multi-agent path finding, Proceedings of the International Conference on Automated Planning and Scheduling, 83-87.
9. Li, J., Felner, A., Boyarski, E., Ma, H., Koenig, S., (2019), Improved heuristics for multi-agent path finding with conflict-based search, IJCAI, 442-449.
10. Xie, L., Li, H., Thieme, N., (2018), From simulation to real-world robotic mobile fulfillment systems, arXiv preprint arXiv:1810.03643.
11. Li, J., Harabor, D., Stuckey, P. J., Ma, H., Koenig, S., (2019), Symmetry-breaking constraints for grid-based multi-agent path finding, Proceedings of the AAAI Conference on Artificial Intelligence, 6087-6095.
12. Li, J., Harabor, D., Stuckey, P. J., Ma, H., Gange, G., Koenig, S., (2021), Pairwise symmetry reasoning for multi-agent path finding search, Artificial Intelligence, 301, 103574.
13. Li, J., Harabor, D., Stuckey, P. J., Felner, A., Ma, H., Koenig, S., (2019), Disjoint splitting for multi-agent path finding with conflict-based search, Proceedings of the International Conference on Automated Planning and Scheduling, 279-283.
14. Boyarski, E., Felner, A., Sharon, G., Stern, R., (2015), Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding, Proceedings of the International Conference on Automated Planning and Scheduling, 47-51.
15. Li, J., Ruml, W., Koenig, S., (2021), Eecbs: A bounded-suboptimal search for multi-agent path finding, Proceedings of the AAAI Conference on Artificial Intelligence, 12353-12362.
16. Salzman, O., Stern, R., (2020), Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems, Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, 1711-1715.
17. Stern, R., Sturtevant, N., Felner, A., Koenig, S., Ma, H., Walker, T., Li, J., Atzmon, D., Cohen, L., Kumar, T., (2019), Multi-agent pathfinding: Definitions, variants, and benchmarks, Proceedings of the International Symposium on Combinatorial Search, 151-158.

SeoHyun Yoo
M.Sc. candidate in the Department of Interdisciplinary Robot Engineering Systems, Hanyang University. Her research interest includes multi-robot systems, path planning for multiple mobile robots or agents, task and motion planning.
E-mail: sh0430@hanyang.ac.kr

SeongTaek Im
M.Sc. candidate in the Department of Interdisciplinary Robot Engineering Systems, Hanyang University. Her research interest includes multi-robot systems, path planning for multiple mobile robots or agents, task and motion planning.
E-mail: st9051@hanyang.ac.kr

HyoJae Kang
Ph.D. candidate in the Department of Interdisciplinary Robot Engineering Systems, Hanyang University. His research interest includes mechanism design, control architectures and programming, gripper and other end-effectors, entertainment robotics.
E-mail: majaae5@hanyang.ac.kr

ChanHui Jung
Researcher in the Department of Robot Engineering, Hanyang University. His research interest includes precision engineering. multi-robot systems, path planning for multiple mobile robots or agents, task and motion planning.
E-mail: qhrejddlvltm@hanyang.ac.kr

Daehee Han
M.Sc. candidate in the Department of Interdisciplinary Robot Engineering Systems, Hanyang University. His research interest motion and path planning, integrated planning and control, AI-enabled robotics, reinforcement learning.
E-mail: hdh1645@hanyang.ac.kr

Min-Sung Kang
Associate professor in the School of Smart Convergence Engineering, Hanyang University. His research interest includes mechanism design, planning, control architectures and programming, embedded systems.
E-mail: wowmecha@hanyang.ac.kr