Conflict-Based Search for Multi-Agent Path Finding Among Movable Obstacles

Shaoli Hu[1,2], Shizhe Zhao[1], and Zhongqiang Ren[1⋆]

[1] Shanghai Jiao Tong University, Shanghai, China
[2] The Hong Kong University of Science and Technology, Hong Kong, China
zhongqiang.ren@sjtu.edu.cn

Abstract. This paper investigates Multi-Agent Path Finding Among Movable Obstacles (M-PAMO), which seeks collision-free paths for multiple agents from their start to goal locations among static and movable obstacles. M-PAMO arises in logistics and warehouses where mobile robots are among unexpected movable objects. Although Multi-Agent Path Finding (MAPF) and single-agent Path planning Among Movable Obstacles (PAMO) were both studied, M-PAMO remains underexplored. Movable obstacles lead to new fundamental challenges as the state space, which includes both agents and movable obstacles, grows exponentially with respect to the number of agents and movable obstacles. In particular, movable obstacles often closely couple agents together spatially and temporally. This paper makes a first attempt to adapt and fuse the popular Conflict-Based Search (CBS) for MAPF, and a recent single-agent PAMO planner called PAMO*, together to address M-PAMO. We compare their performance with up to 20 agents and hundreds of movable obstacles, and show the pros and cons of these approaches. Finally, we deploy our planner on real robots among movable obstacles in a lab setting, where the robots can push the movable obstacles in correct orders to reach their respective goals, which is otherwise hard to achieve with existing methods.

Keywords: Multi-Agent Path Finding, Path Planning, Movable Obstacles

1 Introduction

Multi-Agent Path Finding (MAPF) seeks collision-free paths for multiple agents among static obstacles from their start to goal locations while minimizing their arrival times. This paper considers a variant called Multi-agent Path finding Among Movable Obstacles (M-PAMO), where some of the obstacles can be pushed by the agents. When there are no movable obstacles, M-PAMO becomes MAPF. MAPF is NP-hard [1] and so is M-PAMO. When there is only one agent,

⋆ This work was supported by the Natural Science Foundation of Shanghai under Grant 24ZR1435900.
⋆ Corresponding author.

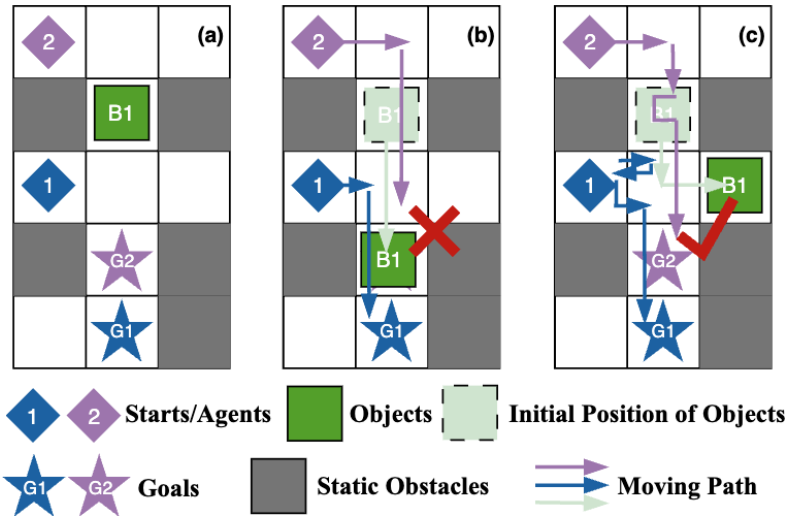


Fig. 1. An example of M-PAMO. (a) shows the problem instance. (b) shows that if agent 1 follows its shortest path and reaches its goal at time $t = 3$, there is no way for agent 2 to reach its goal due to the movable obstacle $B1$, since it cannot push $B1$ onto agent 1 at time $t = 4$. (c) shows the solution returned by our CBS-MOH and CBS-MOL, where agent 1 has to wait for 2 steps at first and then pushes $B1$ to the right so that both agents can then reach their goals.

M-PAMO becomes the Path planning Among Movable Obstacles (PAMO) [2–4]. M-PAMO arises in logistics and warehouses where the passages are blocked by unexpected movable items or shelves [5, 6].

Although MAPF [1, 7] and PAMO [2–4] were both studied, we are not aware of any study on M-PAMO. The introduction of movable obstacles (referred to as objects or boxes hereafter) into MAPF leads to new fundamental challenges, and makes some of the popular techniques for MAPF less effective. The movable obstacles lead to a much larger state space that grows exponentially with respect to the number of both the agents and the objects. In particular, these objects often closely couple the agents together spatially and temporally (Fig. 1): for example, an object that was pushed to location v by agent i at time t may block the only path of agent j at time t' that is much later than t, forcing the planner to consider different ways of interaction among agents i, j and the objects, to find this only solution.

As a first attempt, this paper focuses on Conflict-Based Search (CBS) [7], a popular method for MAPF, and investigates how to extend it to handle M-PAMO. CBS represents the workspace as a graph and employs a two-level search to handle MAPF, where the low-level addresses single-agent planning and the high-level resolves agent-agent conflicts. CBS begins by planning an individually optimal path for each agent while ignoring others, and then detects collisions among the agents along their paths. For a conflict between agents i, j, the high-level of CBS resolves it by adding constraints to either i or j and invokes the

low-level to replan agent i's or j's path to satisfy the added constraints. CBS repeats this process, alternating between high-level and low-level, until a set of conflict-free paths is found.

We propose two variants of CBS for M-PAMO, where the first one CBS-MOH (Movable Obstacles on High-level) addresses the objects only at its high-level, while the second one CBS-MOL also modifies the low-level to handle the objects. Specifically, CBS-MOH ignores objects at its low-level and can leverage any existing single-agent planners used in CBS. CBS-MOH handles objects by introducing new types of conflicts and constraints into the search, and adds those constraints to the agents when conflicts of agent-agent, agent-object, object-object and object-environment are detected. In contrast, CBS-MOL further considers the objects at its low-level, where each agent's path is always planned among all objects. For this purpose, we extend our prior work for (single-agent) PAMO called PAMO* [2] to plan in space-time, and name the resulting planner ST-PAMO*, which is used as the low-level of CBS-MOL.

Although CBS is guaranteed to find an optimal solution for MAPF, we show that our CBS-MOH and CBS-MOL are not guaranteed to find optimal solutions for M-PAMO by providing intuitive counter-examples. The development of CBS-MOH and CBS-MOL and the examples show the algorithmic challenge in M-PAMO, which provides clues for future research. We compare two proposed algorithms in various maps with up to 20 agents and hundreds of objects, which demonstrate the reasonable scalability of our methods, in terms of both the number of agents and the number of movable obstacles.

Finally, we also deploy our planner on real robots among movable obstacles in a lab setting. In this real-robot demo, the robots have to push the movable obstacles in certain order with careful collaboration to navigate towards their respective goal, which is otherwise hard to achieve either with the existing MAPF methods that ignore movable obstacles, or with the existing single-agent PAMO planner that ignores the robot-robot interaction and collaboration.

1.1 Related Work

MAPF algorithms fall on a spectrum from coupled [8] to decoupled [9], trading off completeness and optimality for scalability. In the middle of this spectrum lies the popular dynamically coupled methods such as the popular CBS [7], which has been extended and improved in many ways [10–12]. Some recent work considers modifiable environments where the shelves in the warehouse can be rearranged when needed before planning paths for the agents [5] or when some agents have no tasks [6], which is different from M-PAMO.

PAMO was formulated and studied in different ways, such as in a grid [3], with polygonal obstacles [13], known environments [4], and unknown environments [14], using search [4], sampling [15], and learning [16] methods. However, none of them provides completeness or solution quality guarantees. The difficulty in PAMO is that the state space includes not only the robot's position but also the objects' positions, and the dimensionality of the state space grows exponentially as the number of objects increases. Our prior work [2] presents an

efficient approach called PAMO* with completeness and optimality guarantees, which runs A*-like search while leveraging the fact that most of the objects are intact during planning, which thus helps limit the actual state space explored during planning.

2 Problem Statement

Let the index set $I_A = \{1, 2, \ldots, A\}$ denote a set of A agents. All agents move in a workspace represented as a finite grid-like graph $G = (V, E)$, where G is a 2D occupancy grid that represents the workspace, where each cell $v \in V$ has coordinates $v = (x, y) \in \mathbb{Z}^+$ indicating the column and row indices of the cell in the grid. The time dimension is discretized into time steps. At any cell $v = (x, y)$, the possible actions of an agent are to wait in place or to move in one of the four cardinal directions and arrive at one of the neighboring cells $\{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$. Each action takes a time step. Let $N(v)$ denote the set of neighboring cells of $v \in V$. Each edge $e = (v_1, v_2) \in E$ indicates the movement from cell v_1 to another cell v_2.

At any time, all cells in V are partitioned into three subsets: free space V_{free}, static obstacles V_{so}, and movable obstacles (i.e., objects, or boxes) V_{mo}, i.e., $V = V_{free} \cup V_{so} \cup V_{mo}$. Since the boxes can be pushed as explained later, the sets V_{mo} and V_{free} change over time. The boxes are numbered by the index set $I_B = \{1, 2, \cdots, |V_{mo}|\}$. Each cell $v \in V_{free}$ is obstacle-free and can be occupied by an agent. Each cell $v \in V_{so}$ represents a static obstacle and cannot be occupied by any agent at any time. Each cell $v \in V_{mo}$ indicates a box. When an agent is adjacent to a box $v \in V_{mo}$ and moves to cell v, the box at v is pushed in the same direction as the agent's movement simultaneously. Specifically, a box $v \in V_{mo}$ can be pushed by an agent to an adjacent cell if all of the following conditions hold: (i) the current cell v_c occupied by an agent $i \in I_A$ is next to v; (ii) the agent $i \in I_A$ takes an action to reach v; and (iii) the box can be pushed into a neighboring cell $v' = v + (v_c - v)$ that is in free space (i.e., $v' \in V_{free}$) and is not occupied by another agent $j \in I_A, j \neq i$. Note that, Condition (iii) ensures a box cannot be pushed to v if v is occupied by another box or agent. When a box is pushed, the agent's action is also referred to as a push.

We use a superscript $i \in I_A$ over a variable to show the agent to which the variable belongs (e.g. $v^i \in V$ is a vertex related to agent i). Let $v_s^i, v_g^i \in V$ respectively denote the start and goal location of agent i. All agents share a global clock and start the paths at time $t = 0$. Each action of an agent, either wait, move or push, requires one unit of time. Let $\pi^i(v_1^i, v_\ell^i)$ be a path from v_1^i to v_ℓ^i via a sequence of vertices $(v_1^i, v_2^i, \ldots, v_\ell^i)$ in G. Each pair of adjacent vertices in π^i corresponds to an action of the agent. Let $g(\pi^i(v_1^i, v_\ell^i))$ denote the cost value of the path, which is the arrival time at v_ℓ^i along the path. Here, $g(\pi^i(v_1^i, v_\ell^i))$ is equal to the number of actions along the path, i.e., $g(\pi^i(v_1^i, v_\ell^i)) = \Sigma_{j=1,2,\ldots,\ell-1} cost(v_j^i, v_{j+1}^i)$, with $cost(v_j^i, v_{j+1}^i) = 1$. We denote $\pi^i(v_1^i, v_\ell^i)$ simply as π^i, and $g(\pi^i)$ as g^i when there is no confusion.

Any two agents (or two boxes) are in agent-agent conflict (or box-box conflict) if one of the following two cases happens. The first case is a vertex conflict where any two agents (or two boxes) occupy the same location at the same time. The second case is an edge conflict where any two agents (or two boxes) move through the same edge from opposite directions between times t and $t+1$ for some t. Similarly, there is an agent-box conflict if an agent and a box occupy the same vertex or move through the same edge at the same time. All of agent-agent (AA), box-box (BB), and agent-box (AB) conflicts (Fig. 2) are simply referred to as conflicts when there is no ambiguity.

Problem 1 (M-PAMO). Multi-Agent Path Finding Among Movable Obstacles (M-PAMO) seeks a set of conflict-free paths $\{\pi^i\}$ for all agents $i \in I_A$, while the sum of individual path costs $\sum_{i \in I_A} g(\pi^i)$ reaches the minimum.

When $V_{mo} = \emptyset$, M-PAMO becomes MAPF. When there is only one agent $I_A = \{1\}$, M-PAMO becomes PAMO.

3 Method

3.1 Review of Conflict-Based Search

CBS [7] is a two-level algorithm. The high-level maintains a Constraint Tree (CT). Each node $P = (\pi, g, \Omega)$ contains a joint path $\pi = (\pi^1, \pi^2, \ldots, \pi^A)$, a cost value $g = \sum_{i \in I_A} g(\pi^i)$, and a constraint set Ω. The form of each constraint is (i, v, t) or (i, e, t), which means agent i is prohibited from occupying vertex v or traversing edge e at time t. CBS constructs a CT with the root node $P_{root} = (\pi_o, g(\pi_o), \emptyset)$, where the joint path π_o is obtained by running the low-level (single-agent) planner (`LowLevelPlan`), such as A*, for every agent respectively with an empty set of constraints while ignoring any other agents. P_{root} is added to OPEN, a queue that prioritizes nodes based on their g-values from the minimum to the maximum.

In each iteration, a node $P = (\pi, g, \Omega)$ with the minimum g-value is selected from OPEN for expansion, where every pair of paths in π is checked for a vertex conflict (i, j, v, t) (and an edge conflict (i, j, e, t)). If no conflict is detected, π is conflict-free and is returned as a *solution* (i.e., a conflict-free joint path) and this solution is guaranteed to have the optimal cost. Otherwise, two constraints $\omega^i = (i, v, t)$ and $\omega^j = (j, v, t)$ are generated respectively, and two new constraint sets $\Omega \bigcup \{i, v, t\}$ and $\Omega \bigcup \{j, v, t\}$ are generated. Edge conflict is handled in a similar manner. For agent i in each of those two constraints (i, v, t) and the corresponding newly generated constraint set $\Omega' = \Omega \bigcup \{i, v, t\}$, the low-level planner is invoked to plan a minimum cost path π'^i of agent i subject to all constraints related to agent i in Ω'. The low-level planner typically runs A*-like search in a space-time graph with constraints marked as inaccessible vertices and edges in this space-time graph. After the low-level search, a new joint path π' is formed by first copying π and then replacing agent i's path π^i with π'^i. Finally, for each of the two generated constraints, a corresponding new node is

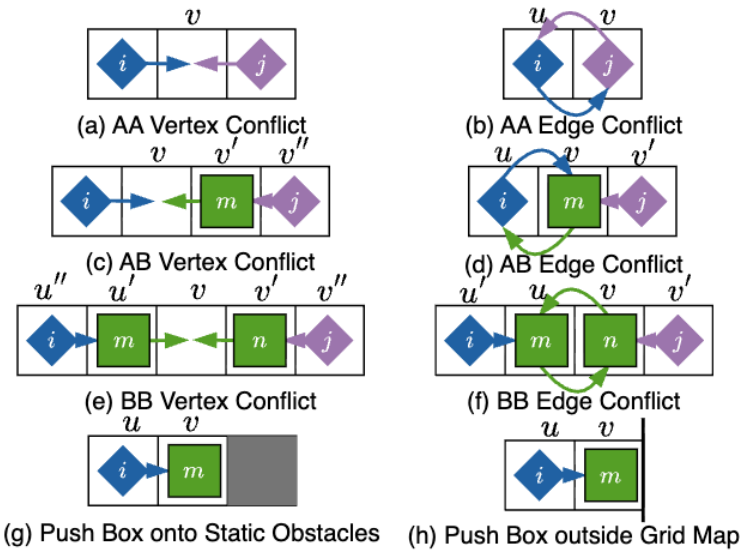


Fig. 2. Different types of conflicts in M-PAMO.

generated and added to OPEN for future expansion. CBS [7] is guaranteed to find a minimum cost solution for a solvable MAPF.

3.2 Resolving AB, BB and BPR Conflicts

We first present the conflict resolution approach for M-PAMO, which is used by both CBS-MOH and CBS-MOL. Among the three types of conflicts, AA conflicts are resolved in the same way as in CBS. We now focus on resolving AB and BB conflicts. Boxes cannot move themselves and can only be pushed by an agent. To prevent a box from occupying a vertex, we forbid the corresponding push action of the agent that moves the box by adding an edge constraint. Specifically, for an AB vertex conflict (i, m, v, t) as shown in Fig. 2 (c), where agent $i \in I_A$ and box $m \in I_B$ occupy the same vertex v at time t, the following two constraints are generated to resolve the conflict:

- Constraint (i, v, t) prevents agent i from occupying v at time t.
- Constraint $(j, (v'', v'), t)$ forbids agent $j \in I_A$ to move from v'' to v' $(v'', v' \in V_{free})$ between times $t-1$ and t, which pushes the box m from v' to v between times $t-1$ and t.

Note that, the second constraint $(j, (v'', v'), t)$ is imposed on the agent j that pushed box m instead of adding constraint on the box itself. We discuss in Sec. 3.5 why not add constraints onto the boxes.

An AB edge conflict $(i, m, (u, v), t)$ can be resolved in a similar manner as AB vertex conflict. As shown in Fig. 2 (d), agent $i \in I_A$ and box $m \in I_B$ swap their

locations over the same edge (u, v) (i.e., $u \in V_{free}$) at time t. The following two constraints are generated to resolve the conflict: Constraint $(i, (u, v), t)$ prevents agent i from u to v at time t, and constraint $(j, (v', v), t)$ forbids agent j to move from v' to v between times $t-1$ and t, which pushes the box m from v to u between times $t-1$ and t.

For a BB vertex conflict (m, n, v, t) as shown in Fig. 2 (e), where box $m \in I_B$ and box $n \in I_B$ occupy the same vertex v at time t, the following two constraints are generated to resolve the conflict: Constraint $(i, (u'', u'), t)$ forbids agent $i \in I_A$ to move from u'' to u' $(u'', u' \in V_{free})$ between times $t-1$ and t, which pushes the box m from u' to v between times $t-1$ and t. Constraint $(j, (v'', v'), t)$ forbids agent $j \in I_A$ to move from v'' to v' between times $t-1$ and t, which pushes the box m from v' to v between times $t-1$ and t. A BB edge conflict $(m, n, (u, v), t)$, as shown in Fig. 2 (f), can be resolved similarly by the following two constraints $(i, (u', u), t)$ and $(j, (v', v), t)$.

Finally, our approach also needs to check the situations where a box is pushed onto a static obstacle (Fig. 2 (g)) or pushed outside the grid map (Fig. 2 (h)). We refer to these two situations as *box pushing rules (BPR)*. If a push action of an agent i leads to violation of the BPR, a corresponding constraint (i, e, t) is generated to forbid agent i to use the edge e (that is, perform that push action) at time t. The violation of BPR yields only one constraint on agent i, which differs from the two constraints generated from an AA, AB or BB conflict. Such BPR checks are needed since the low-level of CBS-MOH (next section) does not consider any box and may return a path that pushes a box outside the map.

3.3 CBS-MOH: Handling Objects on the High-Level

The CBS-MOH algorithm is similar to CBS. The major difference lies in conflict detection (Alg. 1 Line 6). While CBS checks for AA conflicts along a joint path, CBS-MOH simulates the joint path of the boxes $\pi^{I_B} = \{\pi^m | m \in I_B\}$ based on the agents' joint path $\pi = \{\pi^i, i \in I_A\}$, and checks for conflicts and violation of BPR.

To check for conflicts (cft) or violation of BPR (bpr), as shown in Alg. 2, `SimAndConflictCheck` initializes an empty joint path for boxes π^{I_B} and then sets the initial boxes' positions $\pi^{I_B}(t = 0)$ as in V_{mo}. Then, it gets the maximum time t_{max} of the longest agent path $\pi^i \in \pi, i \in I_A$, which is used as the number of simulation steps. For each time step $t = 1, 2, \cdots, t_{max}$, the positions of the boxes at time t can be obtained based on the positions of the agents at times $t-1$ and t and the positions of the boxes at time $t-1$. Meanwhile, AA, AB, and BB conflicts as well as the violation of BPR are detected for each time step. If any conflict cft or bpr is detected by Alg. 2 `Check`, that cft or bpr is immediately returned to Alg. 1 for constraint generation (`GenerateConstraint`). Again, if a bpr is detected (i.e., cft =NULL, bpr $\neq$NULL), `GenerateConstraints` returns only one constraint (Alg. 1 Line 9). If neither cft nor bpr are found, `SimAndConflictCheck` returns NULL and terminates, indicating that the given joint path π of the agents is conflict-free.

Algorithm 1 High-level of CBS-MOH & CBS-MOL

$\Omega_0 \leftarrow \emptyset$
$\pi_o, g_o \leftarrow$ `LowLevelPlan`(Ω_0)
Add $P_{root} = (\pi_o, g(\pi_o), \emptyset)$ to OPEN
while OPEN not empty **do**
    $P_k = (\pi_k, g_k, \Omega_k) \leftarrow$ OPEN.pop()
    $cft, bpr \leftarrow$ `SimAndConflictCheck`(π_k)
    if cft and bpr is NULL **then**
        return π_k
    $\Omega \leftarrow$ `GenerateConstraints`(cft, bpr)
    for each $\omega^a \in \Omega$ **do**
        $\Omega'_k = \Omega_k \bigcup \{\omega^a\}$
        $\pi'_k, g'_k \leftarrow$ `LowLevelPlan`(a, Ω'_k)
        Add $P_{k'} = (\pi'_k, g'_k, \Omega'_k)$ to OPEN
return failure

Algorithm 2 `SimAndConflictCheck`

$\pi^{I_B} \leftarrow \emptyset$
$\pi^{I_B}(t=0) \leftarrow V_{mo}$
$t_{max} \leftarrow \max_{i \in I_A} g(\pi^i)$
for $t = 1, 2, \cdots, t_{max}$ **do**
    $\pi^{I_B}(t) \leftarrow$ `SimStep`$(\pi^{I_B}(t-1), \pi(t-1), \pi(t))$
    $cft, bpr \leftarrow$ `Check`$(\pi^{I_B}(t-1), \pi^{I_B}(t), \pi(t-1), \pi(t)))$
    if $cft \neq$ NULL **or** $bpr \neq$ NULL **then**
        return cft, bpr
return NULL

3.4 CBS-MOL: Handling Objects on the Low-Level

The high-level of CBS-MOL is the same as that of CBS-MOH. The biggest difference lies in their low-levels. We first extend PAMO* [2] to plan in the space-time and then use it as the low-level planner of CBS-MOL to handle constraints added by the high-level.

PAMO* addresses the (single-agent) PAMO problem using A*-like search in a state space $S = G^{1+|V_{mo}|} = G \times G \times \cdots \times G$ that encodes the position of both the agent being planned and all the boxes. When generating successor states, PAMO* discards successors that are invalid based on the motion of the agent and the boxes. The search is guided by a heuristic that is based on the shortest path to the goal while ignoring all the boxes.

We extend PAMO* to ST-PAMO*, which plans in a time-augmented state space $S \times T = G^{1+|V_{mo}|} \times T$, where $T = \{1, 2, \cdots, t_h\}$ denotes the set of t_h total time steps. When generating the successors, in addition to validating agent-box motion as in PAMO*, ST-PAMO* also checks against the added constraints from the high-level search. If any of those constraints is violated, the successor is discarded. ST-PAMO* terminates when a path reaching the goal v_g^i is found. If any constraint ω is added at v_g^i, ST-PAMO* finds a path in $S \times T$ that reaches

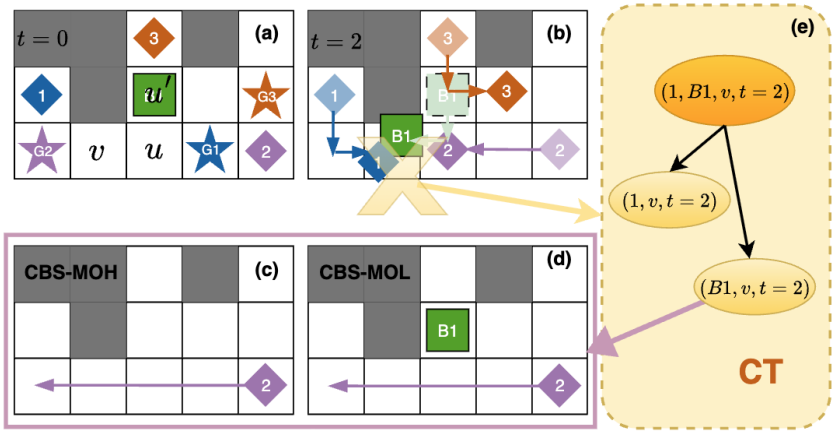


Fig. 3. An example where adding constraints onto the boxes fails. (a) shows the instance. (b) shows a conflict at t=2. (c) and (d) show the low-level planning: CBS-MOH ignores the box, while CBS-MOL still regards the box at its original position. (e) shows the constraint tree where the added constraint fails to resolve the conflicts and the search never terminates. More details can be found in Sec. 3.5.

v_g^i at a time step later than the time of any of those constraints ω, so that the agent can stay at v_g^i after the arrival.

In CBS-MOL, each invocation of ST-PAMO* always considers the initial positions of the boxes and is not aware of the boxes that are pushed by other agents. In other words, ST-PAMO* can only handle boxes implicitly by satisfying the constraints imposed on that agent. For this reason, the BPR checks are still useful in CBS-MOL, since it is possible that agent i pushes a box m at first which blocks the path of agent j, and agent j then pushes the box outside the map without being aware of it. Fig. 3 shows a similar situation.

3.5 Discussion and Analysis

Constraints on the Boxes Both our CBS-MOH and CBS-MOL add constraints on the agents to resolve AA, AB and BB conflicts, as opposed to adding constraints on the boxes. We provide an example in Fig. 3 to illustrate the consequence of adding constraints onto the boxes. Consider the instance in Fig. 3 (a), at $t = 2$, there is a conflict $(1, B1, v, t = 2)$ between agent 1 and box $B1$ at vertex v. Different from the proposed constraint generation as aforementioned in Sec. 3.2, an alternative way is to generate constraints $\omega^1 = (1, v, t = 2)$ on agent 1 for one branch, and $\omega^2 = (B1, v, t = 2)$ on the box $B1$ for the other branch. The constraint ω^2 means that, for any future nodes following this branch, no matter which agent is replanned, the low-level planner must ensure the box $B1$ is not pushed to v at time $t = 2$. However, for this instance, when replanning agent 2 after adding ω^2, the low-level planner of agent 2 is not aware that the box $B1$ is pushed to vertex u at time $t = 1$. Instead, the low-level planner of

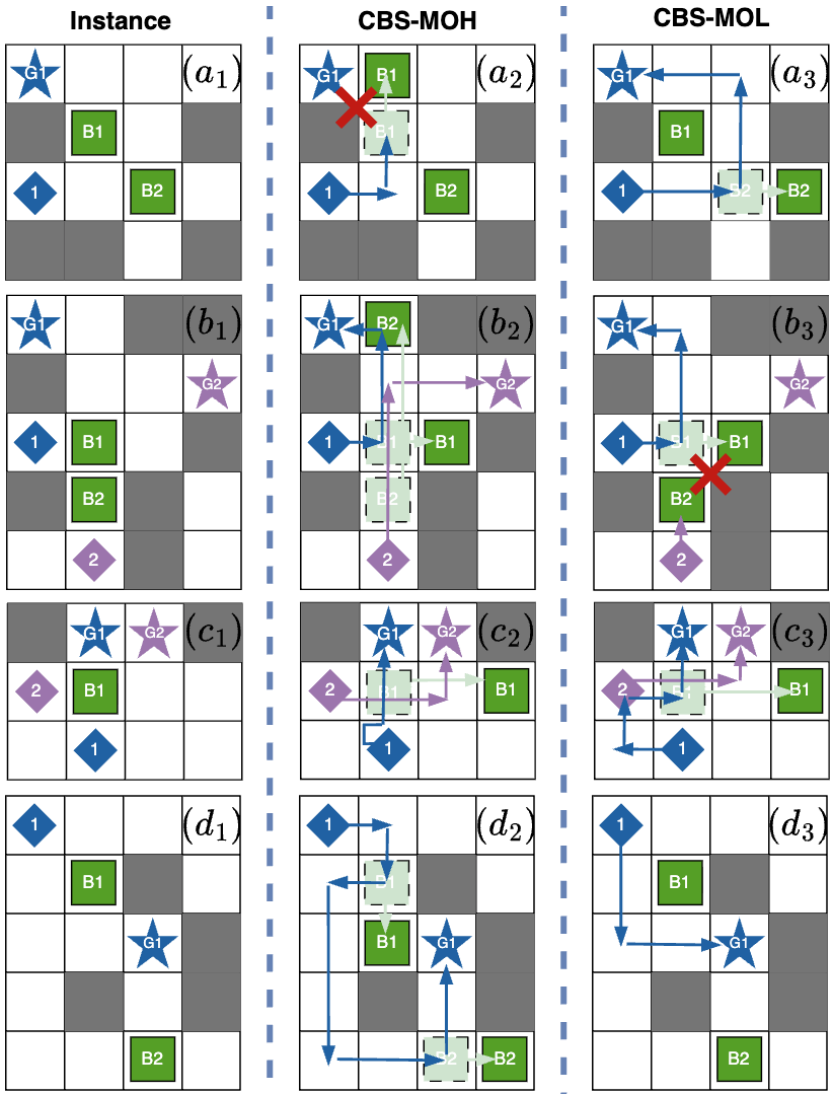


Fig. 4. M-PAMO examples. The first column shows 4 different instances. The 2nd and 3rd column show the planned results of CBS-MOH and CBS-MOL respectively. Row (a) shows a case where CBS-MOH fails while CBS-MOL succeeds. Row (b) shows a case where CBS-MOH succeeds while CBS-MOL fails. Row (c) shows a case where both planners succeed but find different paths, and CBS-MOL fails to find an optimal solution. Row (d) shows a case where CBS-MOH fails to find an optimal solution.

agent 2 still regards $B1$ at u' and the optimal path returned would still be the same. As a result, the high-level search would detect the same conflict again and the algorithm never terminates. Additional information, such as boxes that were

pushed by other agents at different times, needs to be passed to the low-level in an intelligent way, which is left as future work.

Loss of Completeness and Optimality The proposed CBS-MOH and CBS-MOL are neither complete nor solution optimal as examplified in Fig. 4. Fig. 4 (a_2) shows CBS-MOH fails to find a solution. Its low-level planner find a path that first pushes $B1$ to $v_{B1'}$ and then further pushes $B1'$ outside the map, which violates BPR. After adding the constraints, the low-level then finds a path that pushes $B1'$ from right to left, moving the box to its goal. Fig. 4 (b_3) shows a failure case for CBS-MOL. When planning for agent 2, since an agent cannot push two boxes simultaneously, ST-PAMO* fails to find a feasible path for agent 2. CBS-MOL cannot find the feasible solution that lets agent 1 first push $B1$ to $v_{B1'}$. Fig. 4 (c_2) shows CBS-MOH finds an optimal solution while CBS-MOL (c_3) is suboptimal. CBS-MOL's low-level planner for agent 1 does not know that $B1$ will be pushed by agent 2 and thus plans a path with a costly, unnecessary push action. In contrast, the high-level of CBS-MOH resolves the AA vertex conflict, so it finds a solution where agent 1 is constrained to wait for agent 2 to move $B1$, resulting in a lower total cost. Fig. 4 (d_1) shows a case where CBS-MOH is suboptimal, while CBS-MOL is optimal. CBS-MOH's low-level finds a shortest path without considering boxes. Fig. 4 (d_2) shows, in CBS-MOH, agent 1 pushes box $B1$ to $v_{B1'}$, resulting in a suboptimal path.

4 Experimental Results

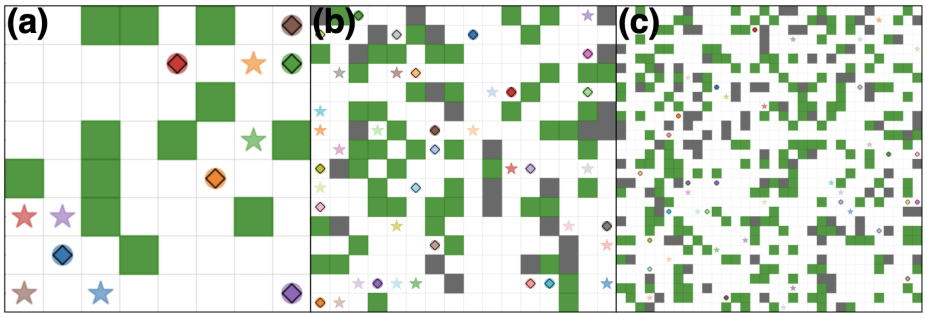


Fig. 5. Test instance examples. (a) An 8×8 empty grid with 6 agents, and 20% box density; (b) a 16×16 random grid with 20 agents, 10% static obstacles, and 20% box density; and (c) a 32×32 random grid with the same obstacle and box densities as (b) but also 20 agents.

We test CBS-MOH and CBS-MOL in grid maps: empty 8×8, 16×16 and random 16×16, 32×32, as shown in Fig. 5. The empty maps have no static obstacles while the random maps have 10% static obstacles. We place the boxes

V_{mo} randomly in the grid without overlapping with static obstacles or the agents' start and goal positions. For each map, we vary the percentage of boxes among 10% and 20%. We create 25 instances for each map, and the instances are not guaranteed to be feasible due to the randomly located objects. We set a runtime limit of 60 seconds for each instance. All tests are conducted on a MacBook laptop with an M2 CPU and 16GB RAM.

4.1 Success Rate

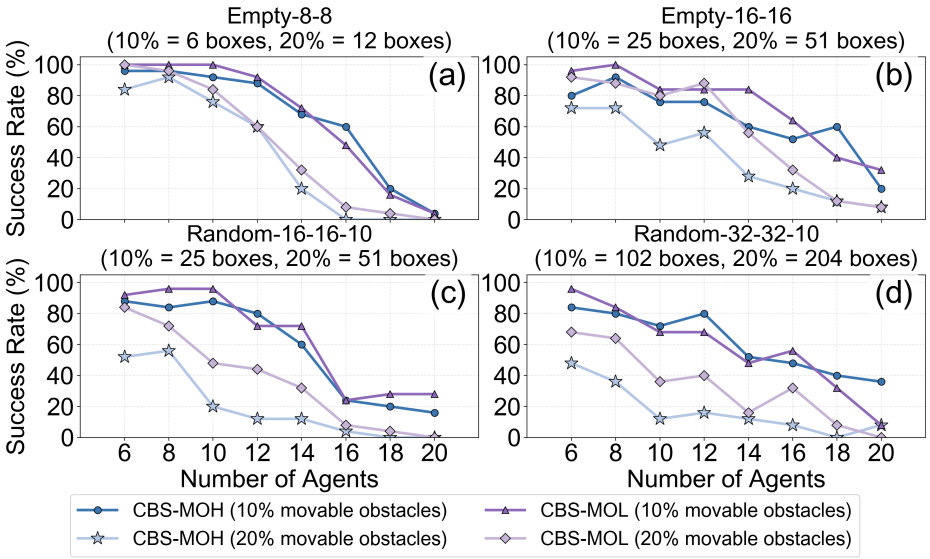


Fig. 6. Success rates of CBS-MOH and CBS-MOL under a varying number of boxes and a varying number of agents across different grid maps.

Fig. 6 shows the success rates of CBS-MOH and CBS-MOL for different numbers of agents up to 20 in four maps, with box density varied between 10% and 20%. For a fixed number of agents, both methods have lower success rates as box density increases. The success rates of CBS-MOH drop faster than CBS-MOL, and a possible reason is the low-level ST-PAMO* in CBS-MOL considers the boxes and can handle more boxes.

For a fixed map, increasing the number of agents also reduces success rates for both methods. At 10% box density, their performance is comparable. However, at 20% box density, CBS-MOL consistently achieves higher success rates than CBS-MOH. This is because CBS-MOH 's low-level planner ignores boxes, leading to more deadlocks in environments with high box density, whereas CBS-MOL 's low-level search (ST-PAMO*) explicitly accounts for boxes during planning and thereby avoids many such failures.

4.2 Expanded Nodes

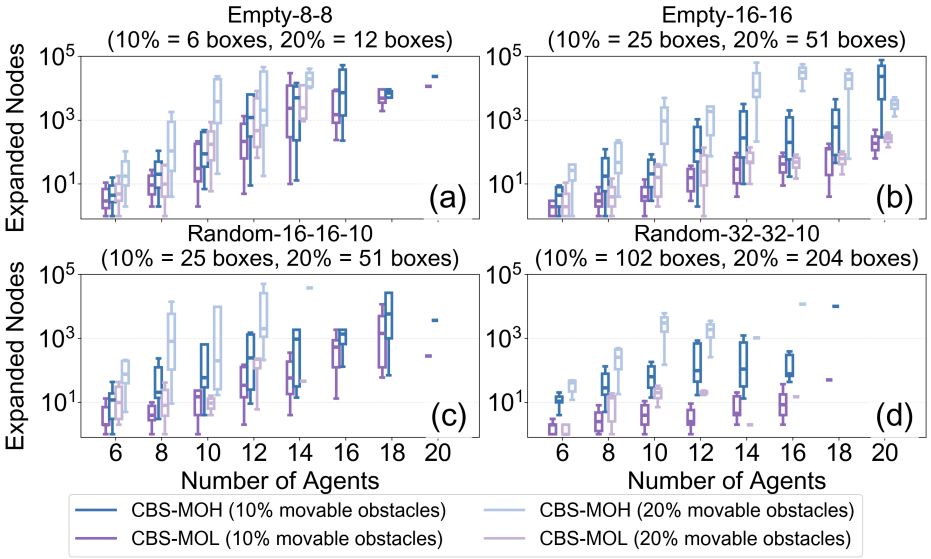


Fig. 7. Expansions of CBS-MOH and CBS-MOL under a varying number of boxes and a varying number of agents across different grid maps.

Fig. 7 reports the number of expanded nodes over instances commonly solved by both CBS-MOH and CBS-MOL, using box plots with medians as central lines. CBS-MOH consistently expands more nodes than CBS-MOL. This is because CBS-MOH employs ST-A* as its low-level planner, which ignores movable obstacles and thus runs faster, allowing more search iterations within the 60-second time limit. However, by neglecting objects during low-level planning, CBS-MOH often finds more conflicts at the high level, leading to a larger number of node expansions and thus higher search overhead.

To summarize, for a small number of boxes, CBS-MOH is recommended due to its fast running speed per iteration. For large number of boxes, CBS-MOL may be more advantageous as it considers boxes at low-level.

4.3 Real-Robot Validation

We deploy our planners on real robots (differential-drive LIMO robots) with a motion capture system. Fig. 8 shows a case where two agents plan paths with a single box in a compact environment. The purple robot (agent a_2) must push the box at first, since otherwise the problem is unsolvable. Our method is able to find a feasible solution and paths are executed by the robots for verification.

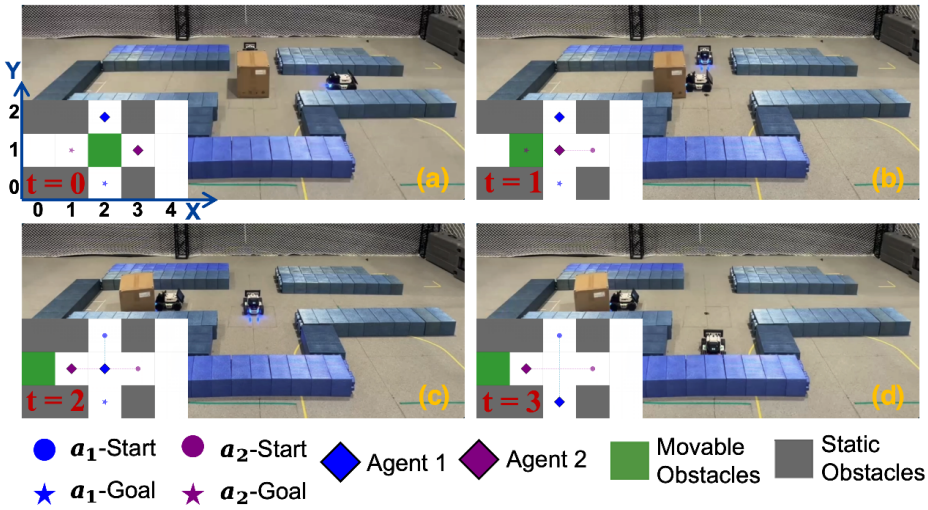


Fig. 8. An M-PAMO scenario involving two agents and a single movable obstacle (a box) for path finding. Agent a_1 starts at $(2, 2)$ with goal $(2, 0)$, while agent a_2 begins at $(3, 1)$ and aims to reach $(1, 1)$. The box initially occupies cell $(2, 1)$, obstructing the direct path between the agents' start and goal locations. Subsequently, agent a_2 pushes the box away from its original position, thereby clearing the obstruction for a_1. Both agents then successfully reach their respective goal locations.

5 Conclusion and Future Work

This paper formulates a new problem M-PAMO, proposes two approaches to solve it based on the popular CBS method, and compares their performance in various maps. The complex interaction between the boxes and the agents couples the agents and makes the problem computationally challenging and burdens the popular CBS approach.

For future work, one can improve CBS-based approaches by passing the information about pushed boxes to the low-level, and combining different constraints by adding strategies (e.g., onto agents, onto boxes) for better performance. Besides, one can also consider actions with different amount of duration where, for example, push actions take longer duration than regular move actions, and study M-PAMO with asynchronous actions [17–19].

References

1. Yu, J., LaValle, S.M.: Structure and intractability of optimal multi-robot path planning on graphs. In: Twenty-Seventh AAAI Conference on Artificial Intelligence (2013)
2. Ren, Z., Suvonov, B., Chen, G., He, B., Liao, Y., Fermuller, C., Zhang, J.: Search-based path planning in interactive environments among movable obstacles. In: 2025 IEEE International Conference on Robotics and Automation (ICRA). pp. 533–539. IEEE (2025)

3. Demaine, E.D., Demaine, M.L., O'Rourke, J.: Pushpush and push-1 are np-hard in 2d. In: Proceedings of the 12th Canadian Conference on Computational Geometry (2000)
4. Stilman, M., Kuffner, J.: Planning among movable obstacles with artificial constraints. The International Journal of Robotics Research 27(11-12), 1295–1307 (2008)
5. Zhang, Y., Fontaine, M.C., Bhatt, V., Nikolaidis, S., Li, J.: Multi-robot coordination and layout design for automated warehousing. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence. IJCAI '23 (2023)
6. Vainshtein, D., Sherma, Y., Solovey, K., Salzman, O.: Terraforming–environment manipulation during disruptions for multi-agent pickup and delivery. In: Proceedings of the International Symposium on Combinatorial Search. vol. 16, pp. 92–100 (2023)
7. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence 219, 40–66 (2015)
8. Standley, T.S.: Finding optimal solutions to cooperative pathfinding problems. In: Twenty-Fourth AAAI Conference on Artificial Intelligence (2010)
9. Silver, D.: Cooperative pathfinding. In: Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment. vol. 1, pp. 117–122 (2005)
10. Barer, M., Sharon, G., Stern, R., Felner, A.: Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In: Proceedings of the international symposium on combinatorial Search. vol. 5, pp. 19–27 (2014)
11. Ren, Z., Rathinam, S., Choset, H.: A conflict-based search framework for multi-objective multiagent path finding. IEEE Transactions on Automation Science and Engineering pp. 1–13 (2022)
12. Ren, Z., Rathinam, S., Choset, H.: CBSS: A new approach for multiagent combinatorial path finding. IEEE Transactions on Robotics 39(4), 2669–2683 (2023)
13. Wilfong, G.: Motion planning in the presence of movable obstacles. In: Proceedings of the fourth annual symposium on Computational geometry. pp. 279–288 (1988)
14. He, B., Chen, G., Wang, W., Zhang, J., Fermuller, C., Aloimonos, Y.: Interactive-far: Interactive, fast and adaptable routing for navigation among movable obstacles in complex unknown environments. arXiv preprint arXiv:2404.07447 (2024)
15. Van Den Berg, J., Stilman, M., Kuffner, J., Lin, M., Manocha, D.: Path planning among movable obstacles: a probabilistically complete approach. In: Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics. pp. 599–614. Springer (2010)
16. Xia, F., Shen, W.B., Li, C., Kasimbeg, P., Tchapmi, M.E., Toshev, A., Martín-Martín, R., Savarese, S.: Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments. IEEE Robotics and Automation Letters 5(2), 713–720 (2020)
17. Ren, Z., Rathinam, S., Choset, H.: Loosely synchronized search for multi-agent path finding with asynchronous actions. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 9714–9719 (2021)
18. Zhou, S., Zhao, S., Ren, Z.: Loosely synchronized rule-based planning for multi-agent path finding with asynchronous actions. In: Proceedings of the AAAI conference on artificial intelligence. vol. 39, pp. 14763–14770 (Apr 2025)
19. Wu, X., Zhao, S., Ren, Z.: Conflict-based search for multi agent path finding with asynchronous actions. In: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2026)