Finding All Optimal Solutions in Multi-Agent Path Finding

Shahar Bardugo[1], **Daniel Koyfman**[1], **Dor Atzmon**[2]

[1]Ben-Gurion University of the Negev, Israel
[2]Bar-Ilan University, Israel
bshahar@post.bgu.ac.il, koyfdan@post.bgu.ac.il, dor.atzmon@biu.ac.il

Abstract

The *Multi-Agent Path Finding* problem (MAPF) aims to find conflict-free paths for a group of agents, leading each agent to its respective goal. MAPF is applicable in navigating autonomous robots and vehicles to their destination. In this paper, we study the requirement of finding all optimal solutions in MAPF. We discuss the representation of all optimal solutions, propose four algorithms for finding them, and perform an extensive empirical evaluation of the proposed algorithms.

1 Introduction

The task in *Multi-Agent Path Finding* (MAPF) (Stern et al. 2019) is to successfully navigate a group of agents to their destinations by calculating a path for each agent such that the agents do not conflict (i.e., collide). MAPF derives from various real-world applications, from indoor applications like navigating robots in automated warehouses (Hönig et al. 2019; Li et al. 2020) to planning outdoor routes for autonomous vehicles (e.g., cars, trains, or drones) (Atzmon, Diei, and Rave 2019; Li et al. 2019a). Finding a single optimal solution for common objective functions is NP-hard (Surynek 2010; Yu and LaValle 2013b). Nevertheless, efficient optimal algorithms excel at doing so for many agents (Gange, Harabor, and Stuckey 2019; Zhang et al. 2020; Li et al. 2021; Lam et al. 2022; Shen et al. 2023).

This paper aims to find *all* optimal solutions in MAPF. In dynamic environments, such as an automated warehouse, the solution can become invalid in response to initial partial knowledge or unexpected changes, such as newly introduced obstacles or agent failures. For such a case, having all optimal solutions allows flexibility by providing alternative paths that can be quickly adapted (Siegmund, Ng, and Deb 2012; Isermann 1977). In other cases, an external decision-maker may need to choose a solution from all optimal solutions when some problem constraints cannot be encoded due to privacy issues (Byers and Waterman 1984; Arthur et al. 1997; Mahadevan and Schilling 2003). Such a case exists, for instance, when there is another objective that cannot be revealed, e.g., energy consumption, collision risk, or financial considerations. Another example is when the selected paths must remain confidential, e.g., the paths of airplanes at the airport. Moreover, finding all optimal solutions to a combinatorial problem, such as MAPF, is a fundamental objective in understanding the structure and complexity of the problem space. This paper is the first to consider this task in MAPF and opens many possible directions for future work.

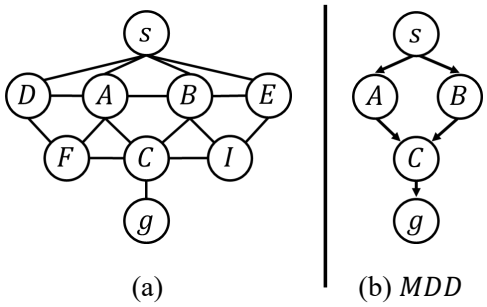

Figure 1: (a) A problem instance; and (b) its MDD.

The contributions of this paper are threefold, as follows. (1) We discuss the representation of all optimal MAPF solutions and present three ways for representing all optimal solutions. (2) For finding all optimal solutions, we propose two naive algorithms, one based on A* (Hart, Nilsson, and Raphael 1968) (A^*_{AS}) and the other is a reduction-based algorithm (Surynek et al. 2016) (RED_{AS}), and two novel algorithms based on CBS (Sharon et al. 2015) (CBS_{AS} and CBS-M_{AS}). (3) We perform expensive experiments with the four algorithms on nine benchmark maps, and analyze the results. We also measure and discuss the impact of the number of agents on the number of optimal solutions.

2 Background and Related Work

Path Finding

A *path* in graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ from start vertex $s \in \mathcal{V}$ to goal vertex $g \in \mathcal{V}$ is a list of vertices such that the list starts with vertex s and ends with vertex g, and any two consecutive vertices in the list are traversable. Let $\pi(t)$ be the t-th vertex in path π. Formally, $\pi(0) = s$, $\pi(|\pi| - 1) = g$, and $\forall t : (\pi(t), \pi(t+1)) \in \mathcal{E}$. The cost $C(\pi)$ of path π equals the number of transitions performed in it ($= |\pi| - 1$). A *Path Finding* problem (PF) is defined by a tuple $\langle \mathcal{G}, s, g \rangle$ and requires such a path. The *optimal* solution (optimal path) to PF is the least-cost path among all PF solutions (paths).

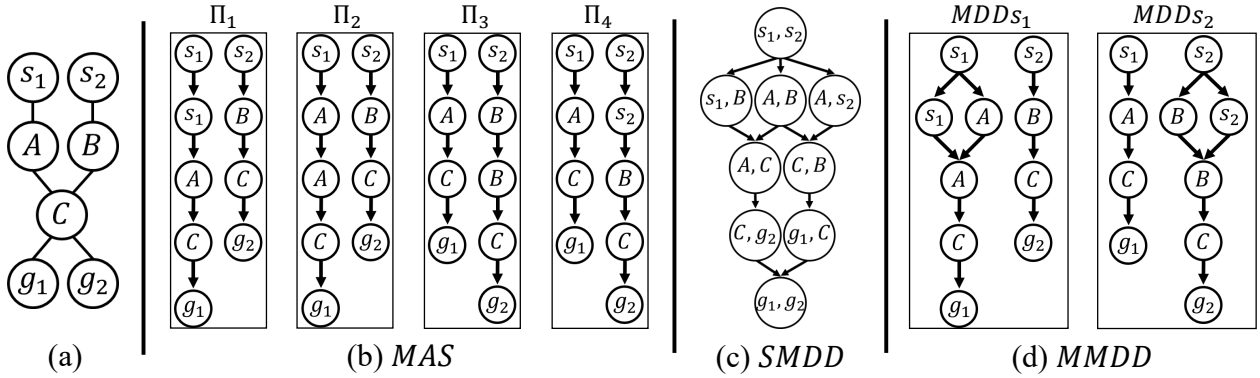

Figure 2: (a) Problem instance; and three ways to represent all its optimal solutions: (b) MAS; (c) $SMDD$; and (d) $MMDD$.

Finding the optimal path to PF can be efficiently done by A^* (Hart, Nilsson, and Raphael 1968), a well-known heuristic search algorithm. A* uses a heuristic function $h(v)$ that estimates the cost to reach the goal vertex g from any given vertex v. A heuristic function is *admissible* if it never overestimates the cost to the goal. Given an admissible heuristic function, A* is guaranteed to return the optimal path.

The *k-Shortest Paths* problem (KSP) (Yen 1971; Eppstein 1998) extends PF and requires the k least-cost paths, which may also contain non-optimal paths. The *K** algorithm (Aljazzar and Leue 2011) leverages A* and uses a detour function to quickly solve KSP. Katz et al. (2018) showed the benefits of using K* also in classical planning.

A *Multi-Value Decision Diagram* (MDD) (Srinivasan et al. 1990; Sharon et al. 2015) is a directed acyclic graph $MDD = (\mathcal{V}_{MDD}, \mathcal{E}_{MDD})$ capable of representing all optimal paths from vertex s to vertex g in graph $\mathcal{G}$ without duplicating vertices that are shared by multiple paths. An MDD is a subgraph of graph $\mathcal{G}$, i.e., $\mathcal{V}_{MDD} \subseteq \mathcal{V}$ and $\mathcal{E}_{MDD} \subseteq \mathcal{E}$. It has a single source vertex $s_{MDD} = s$ and a single sink vertex $g_{MDD} = g$. Every path from vertex s_{MDD} to vertex g_{MDD} represents an optimal path in underlying graph $\mathcal{G}$. The MDD depicted in Figure 1(b) represents all optimal paths for the problem instance presented in Figure 1(a). Note that, vertices s, C, and g are maintained once as they are used by the two optimal paths.

Multi-Agent Path Finding

The *Multi-Agent Path Finding* problem (MAPF) (Stern et al. 2019) extends PF to the case of multiple agents requiring multiple paths. The problem is defined by a tuple $\langle \mathcal{G}, A, S, G \rangle$, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected graph representing the set of vertices $\mathcal{V}$ at which an agent can be positioned and the set of transitions $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ between them; $A = (a_1, \ldots, a_k)$ is a list of k agents; and $S = (s_1, \ldots, s_k)$ and $G = (g_1, \ldots, g_k)$ are lists of start and goal vertices, respectively. Each agent a_i requires a path from its start vertex s_i to its goal vertex g_i. A *plan* $\Pi = (\pi_1, \ldots, \pi_k)$ is a list of paths for the agents. The cost $C(\Pi)$ of plan Π (also called sum-of-costs) equals the sum of the costs of its paths ($= \sum_{\pi_i \in \Pi} C(\pi_i)$). A *vertex conflict* $\langle a_i, a_j, v, t \rangle$ exists between two paths π_i and π_j of agents a_i and a_j if the two agents are simultaneously at the same vertex ($\exists t : \pi_i(t) = \pi_j(t) = v$). A *swapping conflict* (also known as an *edge conflict*) $\langle a_i, a_j, e, t \rangle$ exists between two paths π_i and π_j of agents a_i and a_j if the agents simultaneously swap vertices ($\exists t : (\pi_i(t), \pi_i(t+1)) = (\pi_j(t+1), \pi_j(t)) = e$). A *solution* to MAPF is a *conflict-free* plan, where any two paths do not conflict (no vertex or swapping conflict). The *optimal* solution is the least-cost plan among all solutions.

Algorithms that optimally solve MAPF can be divided into two main categories: reduction-based approaches and search-based approaches. Reduction-based algorithms compile MAPF into another known problem that has mature and effective solvers. Previous studies on reduction-based approaches include reducing MAPF to *Integer-Linear Program* (ILP) (Yu and LaValle 2013a), *Answer Set Programming* (ASP) (Erdem et al. 2013), *SAT* (Surynek et al. 2016; Barták and Svancara 2019), and more. Search-based algorithms systematically explore different paths (or subpaths) until a solution can be determined. Search-based algorithms that optimally solve MAPF include *M** (Wagner and Choset 2015), *Increasing Cost Tree Search* (ICTS) (Sharon et al. 2013), and *Conflict-Based Search* (CBS) (Sharon et al. 2015). The latter, CBS (described below), is a commonly-used algorithm with many improvements introduced over the years (Boyarski et al. 2015, 2022; Felner et al. 2018; Li et al. 2019c,b, 2021; Zhang et al. 2020; Shen et al. 2023).

3 Representing All Optimal Solutions

In this paper, we aim to find all optimal MAPF solutions. In this section, we discuss three ways to represent all optimal MAPF solutions: MAS, $SMDD$, and $MMDD$.

Maintaining All Solutions (MAS)

The simplest way for representing all optimal solutions is to maintain a set $MAS = \{\Pi_1, \Pi_2, \ldots\}$ of all optimal solutions, i.e., every plan $\Pi' \in MAS$ is a different optimal solution and there is no optimal solution Π' such that $\Pi' \notin MAS$. For example, consider the problem instance in Figure 2(a), which contains two agents a_1 and a_2 with start vertices s_1 and s_2, and goal vertices g_1 and g_2, respectively. In this example, one of the agents must wait to avoid a vertex conflict at vertex C. Each of the two agents has two ver-

tices at which it can wait before reaching vertex C; namely, vertices s_1 and A for agent a_1, and vertices s_2 and B for agent a_2. Therefore, this problem instance has the following four optimal solutions $MAS = \{\Pi_1, \Pi_2, \Pi_3, \Pi_4\}$ (also illustrated in Figure 2(b)):

$$(\Pi_1)\ \pi_1 = (s_1, s_1, A, C, g_1), \pi_2 = (s_2, B, C, g_2)$$
$$(\Pi_2)\ \pi_1 = (s_1, A, A, C, g_1), \pi_2 = (s_2, B, C, g_2)$$
$$(\Pi_3)\ \pi_1 = (s_1, A, C, g_1), \pi_2 = (s_2, B, B, C, g_2)$$
$$(\Pi_4)\ \pi_1 = (s_1, A, C, g_1), \pi_2 = (s_2, s_2, B, C, g_2)$$

Shared state-space MDD ($SMDD$)

While an MDD represents multiple paths from a single start vertex to a single goal vertex, the idea of an MDD can be easily extended to represent plans of multiple agents, from multiple start vertices to multiple goal vertices. To this end, let us consider a shared state-space graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ for multiple agents, where each vertex $v_S \in \mathcal{V}_S$ represents a list of vertices for k agents in the underlying graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, i.e., $v_S = (v_1, \ldots, v_k)$, where $\forall v_i \in v_S : v_i \in \mathcal{V}$; vertex v_i represents the vertex of agent a_i in graph $\mathcal{G}$. The start vertex s_S and goal vertex g_S of graph $\mathcal{G}_S$ are the start and goal vertices of the agents: $s_S = S$ and $g_S = G$. Vertex $v'_S = (v'_1, \ldots, v'_k)$ is traversable from (a neighbor of) vertex $v_S = (v_1, \ldots, v_k)$ if the vertices of each agent a_i are traversable (besides the permutation where all agents wait): $(v_S, v'_S) \in \mathcal{E}_S \Leftrightarrow (\forall a_i \in A : (v_i, v'_i) \in \mathcal{E}) \wedge (v_S \neq v'_S)$. Also, conflicting vertices/edges are invalid and pruned. We call an MDD for this shared state-space, which represents paths for all agents, a *Shared state-space MDD* ($SMDD$). $SMDD = (\mathcal{V}_{SMDD}, \mathcal{E}_{SMDD})$ is a subgraph of $\mathcal{G}_S$, i.e., $\mathcal{V}_{SMDD} \subseteq \mathcal{V}_S$ and $\mathcal{E}_{SMDD} \subseteq \mathcal{E}_S$. It has a single source vertex $s_{SMDD} = s_S$ and a single sink vertex $g_{SMDD} = g_S$. Figure 2(c) presents an $SMDD$ to the problem instance in Figure 2(a), where each vertex represents the vertices of the two agents and every path represents an optimal solution.

Multiple MDDs ($MMDD$)

Unlike the single agent case, in the case where multiple agents exist, an optimal MAPF solution may contain paths where an agent visits a vertex multiple times. Therefore, in the multi-agent case, when creating an MDD for a single agent a_i (denoted MDD_i), each vertex $v_{MDD_i} = (v, t)$ is a pair of vertex $v \in \mathcal{G}$ from the underlying graph and a timestep t. For our multiple agents, let $MDDs = (MDD_1, \ldots, MDD_k)$, i.e., an $MDD_i \in MDDs$ for each agent a_i. *Multiple MDDs* ($MMDD$) represent all optimal solutions by a set of $MDDs$ ($MMDD = \{MDDs_1, MDDs_2, \ldots\}$). For any $MDDs_i \in MMDD$, any permutation of paths for the agents results is an optimal MAPF solution. Importantly, in a valid $MMDD$, there is no permutation of paths where agents conflict. Figure 2(d) illustrates an $MMDD$ to the problem instance in Figure 2(a). Note that, the timestep of each vertex is not presented in the figure and is equal to the depth of each vertex in the MDD; that is, the number of edges from the start vertex. Therefore, multiple vertices in the $MMDD$ may contain the same underlying vertices but with different timesteps. $MMDD$ is a middle ground between MAS and $SMDD$; like MAS, $MMDD$ does not maintain the underlying graph's vertices of all agents in each of its vertices and, like $SMDD$, multiple paths are represented using the same vertices.

It is easy to linearize any of the above representations and produce any optimal solution. Therefore, we allow our algorithms presented next to return any of these representations.

Algorithm 1: A$^*_{AS}$

Input: Graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$, Vertex s_S, Vertex g_S
Output: $SMDD$ SMDD

```
1  A*_AS (Graph G_S = (V_S, E_S), Vertex s_S, Vertex g_S)
2      Init OPEN, CLOSED, SMDD, goal
3      Init UB = ∞
4      Init root; root.v_S = s_S; root.p = {}; g(root) = 0
5      Insert root into OPEN
6      while OPEN is not empty do
7          Extract n from OPEN // lowest f(n)
8          if f(n) > UB then
9              return AddToSMDD (SMDD, goal, Null)
10         if n.v_S = g_S then
11             goal = n
12             UB = g(n)
13         Insert n into CLOSED
14         foreach v'_S s.t. (n.v_S, v'_S) ∈ E_S do
15             Init n'; n'.v_S = v'_S; n'.p = {n}
16             g(n') = g(n) + c(n.v_S, n'.v_S)
17             if ∃n'' ∈ OPEN ∪ CLOSED s.t. n''.v_S = v'_S
                 then
18                 if g(n'') < g(n') then
19                     continue
20                 if g(n'') > g(n') then
21                     Remove n'' from OPEN ∪ CLOSED
22                 if g(n'') = g(n') then
23                     n''.p = n''.p ∪ n'.p
24                     continue
25             f(n') = g(n') + h(v'_S)
26             Insert n' into OPEN
27     return SMDD
28 AddToSMDD (SMDD SMDD, Node n^curr, Node n^next)
29     if n^next ≠ Null then
30         E_SMDD = E_SMDD ∪ {(n^curr.v_S, n^next.v_S)}
31     if n^curr.v_S ∉ V_SMDD then
32         V_SMDD = V_SMDD ∪ {n^curr.v_S}
33         foreach n^prev ∈ n^curr.p do
34             AddToSMDD (SMDD, n^prev, n^curr)
35     return SMDD
```

4 A*-based Approach (A$^*_{AS}$)

Adapting A* to find *All Solutions* (A$^*_{AS}$) for MAPF is straightforward. For the completeness of the paper, we describe it in detail. A$^*_{AS}$ is executed on the shared state-space graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$, defined in Section 3, and it represents all optimal solutions from vertex s_S to vertex g_S us-

ing an $SMDD$. A$^*_{AS}$, presented in Algorithm 1, constructs a tree of nodes, where each node n contains a vertex $n.v_S$ in our search space. Each node n is associated with a g-value ($g(n)$), representing the cost of reaching vertex $n.v_S$ from vertex s_S along the search tree; an f-value ($f(n) = g(n) + h(n.v_S)$), representing an estimate for the cost of the entire path from vertex s_S to vertex g_S via vertex $n.v_S$ along the search tree; and a set of backpointers $n.p$. The backpointers are used to represent multiple paths of the same cost to a similar vertex. A similar idea is performed by K* (Aljazzar and Leue 2011) (mentioned in Section 2). Tracking the search tree is done by two lists OPEN and CLOSED; SMDD is used for constructing the $SMDD$; and $goal$ is for maintaining the goal node (line 2). An upper bound UB, initialized with ∞ (line 3), is used to maintain the cost of the optimal solution, when it is found, and to halt after the last optimal solution is found (standard A* halts when the first solution is found). The search starts by inserting into OPEN a $root$ node containing the start vertex s_S ($root.v_S = s_S$; lines 4-5). Iteratively, the node n with the lowest f-value in OPEN is extracted (line 7). If $f(n) > UB$, it means that all optimal solutions are found and SMDD is constructed backwards from $goal$ and returned (lines 8-9, 29-35). Otherwise, if $n.v_S = g_S$, n is maintained in $goal$ and UB is updated (lines 10-12). Then, n is inserted into CLOSED (line 13) and expanded (lines 14-26): (1) a node n' is created for each neighboring vertex v'_S of vertex $n.v_S$ (lines 14-16); and (2) a duplicate-detection mechanism is activated to prevent duplication and, if we already found a node n'' such that $n''.v_S = n'.v_S$ (line 17) then, if $g(n'') < g(n')$, n' is pruned (lines 18-19), if $g(n'') > g(n')$, n'' is pruned (lines 20-21) and, if $g(n'') = g(n')$, n' is pruned but its backpointers are maintained in n'''s backpointers (lines 22-24). In case n' is not pruned, it is inserted into OPEN (lines 25-26). A$^*_{AS}$ is complete and guaranteed to return all optimal solutions.

5 Reduction-based Approach (RED$_{AS}$)

A reduction-based approach compiles MAPF into another problem that has a general-purpose solver. Following the SAT-based MAPF encoding presented by Surynek (2016), we define a Boolean variable for every triplet (a, t, v) of agent (a), timestep (t), and vertex (v). This variable is true iff agent a occupies vertex v at timestep t. To ensure that a valid plan is computed, the following constraints are set on these variables: (1) each agent occupies exactly one vertex at each timestep; (2) no two agents occupy the same vertex at any timestep; (3) an agent may only traverse between two adjacent vertices at every two consecutive timesteps; and (4) no two agents traverse the same edge in opposite directions between two consecutive timesteps. To find optimal solutions, minimizing the cost of the plan is defined as objective.

In our experiments, we used Picat (Zhou, Kjellerstrand, and Fruhman 2015) to implement this reduction (Barták and Svancara 2019). Picat is a publicly-available logic-based programming language. Encoding MAPF in Picat has the advantage that the model can be compiled to SAT, CP, or MILP, and then solved by an appropriate solver. We run a CP compilation of Picat, as it performed best on a small-scale experiment we performed. Picat supports incrementally returning each solution separately when it is found, without re-executing the program from scratch. Thus, it represents all optimal solutions by MAS. This approach, denoted RED$_{AS}$, is complete and halts when all optimal solutions are found.

6 CBS-based Approach

Conflict-Based Search (CBS) (Sharon et al. 2015) has two levels. Its high level constructs a constraint tree (CT) and its low level plans paths under the high level's constraints. A constraint $\langle a_i, x, t\rangle$ prohibits agent a_i from occupying vertex x at timestep t or from traversing edge x between timesteps t and $t+1$. We propose two CBS-based algorithms CBS$_{AS}$ and CBS-M$_{AS}$, which find all optimal solutions. Both algorithms' high level constructs a similar CT (presented once, in Algorithm 2) and halts when a set of CT leaves is found. Also, both algorithms use this set of CT leaves to create an $MMDD$. The main difference between the two algorithms is the set of CT leaves their high level finds and the way they use this set of CT leaves to represent all optimal solutions; in CBS$_{AS}$, the resulted $MMDD$ is not a valid $MMDD$ and may contain conflicts, and the algorithm, thus, creates an $SMDD$ from the $MMDD$ to represent all optimal solutions; in CBS-M$_{AS}$, the resulted $MMDD$ is valid and used to represent all optimal solutions. We start by describing CBS$_{AS}$ (Algorithm 2), which calls the high-level search in line 2.

CBS$_{AS}$

In the high-level search, each CT node N contains a set of constraints $N.constraints$; a plan $N.\Pi$ that satisfies $N.constraints$; the cost $N.cost$ of plan $N.\Pi$ ($C(N.\Pi)$); and $MDDs$ for all agents $N.MDDs$ that satisfies $N.constraints$.[1] The high level starts by initializing OPEN, MMDD, UB, and $root$, and inserting $root$ into OPEN (lines 6-9). Then, an expansion cycle is executed (lines 10-21). The CT node N with the lowest cost is extracted from OPEN (lines 11). If the cost $N.cost$ of the current CT node N exceeds the upper bound UB, the search halts and returns MMDD (lines 12-13). Otherwise, it checks if N is a solution (line 14). In CBS$_{AS}$, as opposed to CBS-M$_{AS}$ described below, CT node N is a solution if its plan $N.\Pi$ is conflict-free (lines 33-35). If it is, N's $MDDs$ ($N.MDDs$) is added to MMDD, UB is updated, and we continue to the next CT node (lines 15-17). If CT node N is not a solution, a conflict $\langle a_i, a_j, x, t\rangle$ is chosen to be resolved (line 18; x is either a vertex or an edge). In CBS$_{AS}$, this conflict is found in plan $N.\Pi$ (line 31). We generate two new CT child nodes N_i and N_j that inherit the constraints of CT node N. To resolve the conflict, CT nodes N_i and N_j also add a new constraint on each of the conflicting agents a_i and a_j to not occupy vertex x at timestep t or traverse edge x between timesteps t and $t+1$ ($\langle a_i, x, t\rangle$ and $\langle a_j, x, t\rangle$); namely, at least one of the two agents a_i or a_j must not be at x at timestep t. The new CT nodes N_i and N_j are then inserted into OPEN (lines 19-21).

[1] Note that, CBS-M$_{AS}$ (described below) does not need to maintain the plan $N.\Pi$ at every CT node N; we define CBS$_{AS}$ and CBS-M$_{AS}$ with a similar definition of CT nodes to ease the presentation of the algorithms.

Algorithm 2: CBS$_{AS}$

Input: MAPF problem instance $instance$
Output: $SMDD$ SMDD

1 **CBS$_{AS}$ (MAPF problem instance** $instance$**)**
2 MMDD = HighLevel ($instance$)
3 SMDD = CreateSMDD (MMDD)
4 **return** SMDD
5 **HighLevel (MAPF problem instance** $instance$**)**
6 Init OPEN, MMDD
7 Init $UB = \infty$
8 Init $root$ with an initial plan and no constraints
9 Insert $root$ into OPEN
10 **while** OPEN *is not empty* **do**
11 Extract N from OPEN // *lowest cost*
12 **if** $N.cost > UB$ **then**
13 **return** MMDD
14 **if** *IsSolution* (N) **then**
15 MMDD = MMDD $\cup \{N.MDDs\}$
16 $UB = N.cost$
17 continue
18 $\langle a_i, a_j, x, t\rangle$ = GetConflict (N)
19 N_i = GenerateChild (N, $\langle a_i, x, t\rangle$)
20 N_j = GenerateChild (N, $\langle a_j, x, t\rangle$)
21 Insert N_i and N_j into OPEN
22 **return** MMDD
23 **GenerateChild (CT node** N**, Constraint** C**)**
24 $N'.constraints = N.constraints \cup \{C\}$
25 $N'.\Pi = N.\Pi$; $N'.MDDs = N.MDDs$
26 Update $N'.\Pi$ to satisfy $N'.constraints$
27 Update $N'.MDDs$ to satisfy $N'.constraints$
28 $N'.cost = C(N'.\Pi)$
29 **return** N'
30 **GetConflict (CT node** N**)**
31 **return** Conflict $\langle a_i, a_j, x, t\rangle$ in $N.\Pi$
32 **IsSolution (CT node** N**)**
33 **if** $N.\Pi$ *is conflict free* **then**
34 **return** true
35 **return** false

When the high-level search halts, it returns a set of $MDDs$ ($MMDD$). However, in CBS$_{AS}$, each of these $MDDs$ may still contain conflicts between agents, i.e., the returned $MMDD$ (MMDD) is not a valid representation of all optimal solutions as it also represents non-solution plans. For example, consider the problem instance in Figure 3(a), which contains three optimal solutions. CBS$_{AS}$ may find an optimal solution Π at the root CT node ($root$), as can be seen in Figure 3(b). Then, in the $MDDs$ returned, both agents can be at vertex C at timestep 1. To solve this issue, CBS$_{AS}$ merges the set of all $MDDs$ into an $SMDD$ and returns it (lines 3-4). In our implementation, we performed this merge by executing A$^*_{AS}$ (presented above) on each $MDDs \in$ MMDD and creating a single $SMDD$ representing all optimal solutions.

Algorithm 3: CBS-M$_{AS}$

Input: MAPF problem instance $instance$
Output: $MMDD$ MMDD

1 **CBS-M$_{AS}$ (MAPF problem instance** $instance$**)**
2 MMDD = HighLevel ($instance$)
3 **return** MMDD
4 **GetConflict (CT node** N**)**
5 **return** Conflict $\langle a_i, a_j, x, t\rangle$ in $N.MDDs$
6 **IsSolution (CT node** N**)**
7 **if** $N.MDDs$ *is conflict free* **then**
8 **return** true
9 **return** false

CBS-M$_{AS}$

Contrary to CBS$_{AS}$, CBS-M$_{AS}$ (Algorithm 3) resolves a new type of conflict, an $MDDs$ conflict (line 5).

Definition 1 ($MDDs$ conflict)**.** *An $MDDs$ conflict $\langle a_i, a_j, x, t\rangle$ exists between two agents a_i and a_j if both MDD_i and MDD_j contain vertex x at timestep t or edge x between timesteps t and $t+1$.*[2]

An $MDDs$ conflict in which x is a vertex is analogous to a vertex conflict and an $MDDs$ conflict in which x is an edge is analogous to a swapping conflict. In CBS-M$_{AS}$, we resolve $MDDs$ conflicts to represent all optimal solutions by a valid $MMDD$. Identifying $MDDs$ conflicts can be done using the standard *Conflict Avoidance Table* data structure (Sharon et al. 2015). An $MDDs$ conflict $\langle a_i, a_j, x, t\rangle$ is resolved similarly to the way a standard conflict is resolved: creating two new CT child nodes and adding the constraint $\langle a_i, x, t\rangle$ to one CT child node and the constraint $\langle a_j, x, t\rangle$ to the other CT child node. In CBS-M$_{AS}$, a CT node N is a solution if it does not contain any $MDDs$ conflict (lines 7-9). Therefore, CBS-M$_{AS}$'s $MMDD$ (MMDD) only represents valid solutions, and can be returned as is to represent all optimal solutions (line 3). Figure 3(c) presents the high level of CBS-M$_{AS}$ when executed on the problem instance in Figure 3(a). As can be seen, the two $MDDs$ of the two CT leaves contain all optimal solutions and only them; for either of the two $MDDs$, any permutation of paths for the agents results in a valid optimal solution and form a valid $MMDD$.

Theorem 1. *CBS$_{AS}$ and CBS-M$_{AS}$ are guaranteed to return all optimal solutions if solutions exist.*

Proof. Let N be a non-solution CT node, containing at least one conflict $\langle a_i, a_j, x, t\rangle$, and let $\Pi^*(N)$ be the set of all optimal solutions satisfying $N.constraints$. When N's conflict is resolved, two CT child nodes N_i and N_j are created. Any solution must satisfy at least one of the new constraints set on these two CT nodes, i.e., there is no solution where agents a_i and a_j are simultaneously at vertex/edge x. Therefore, by resolving a conflict, any solution that satisfies $N.constraints$, also satisfies $N_i.constraints$,

[2]Our conflict definition shares similarities with the one suggested by Morag et al. (2024). However, both conflicts are used to solve different problems, resulting in different solutions.

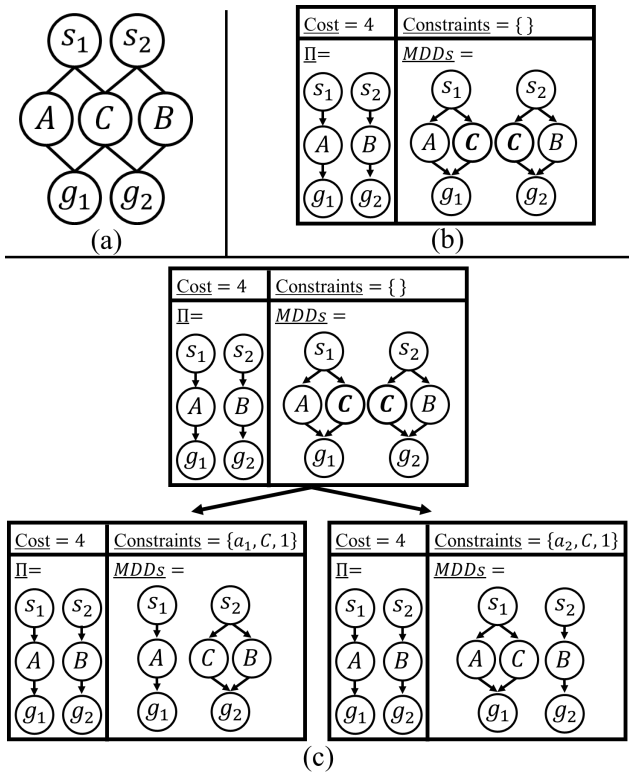

Figure 3: (a) Problem instance. (b) CBS$_{AS}$. (c) CBS-M$_{AS}$.

$N_j.constraints$, or both; i.e., $\Pi^*(N) = \Pi^*(N_i) \cup \Pi^*(N_j)$. The root CT node does not have any constraints and, thus, all optimal solutions satisfy its constraints. Consequently, all optimal solutions can be found in the CT leaves at any time during the high-level search. Moreover, our constraints repeatedly discard conflicting plans that satisfy the constraints and have a minimal cost. As we return all CT leaves that have an optimal cost, all optimal solutions are returned. □

7 Experimental Study

We experimented with A$^*_{AS}$, RED$_{AS}$, CBS$_{AS}$, and CBS-M$_{AS}$ on nine benchmark maps from the *MovingAI* repository (Stern et al. 2019): 8×8 and 32×32 empty grids (*empty-8-8* and *empty-32-32*), 32×32 grids with 20% obstacles (*random-32-32-20*), 32×32 room grids (*room-32-32-4*), 32×32 maze grids (*maze-32-32-2*), and four grid maps from the Dragon Age Origins video game (*den207d*, *hrt002d*, *lgt101d*, and *orz102d*). We based our implementation of CBS$_{AS}$ and CBS-M$_{AS}$ on a publicly available implementation (Li et al. 2021) of CBS-H (Felner et al. 2018) with the WDG heuristic (Li et al. 2019b) and the Disjoint Splitting improvement (Li et al. 2019c), which requires changing the cost of each CT node to its f-value. Our experiments were conducted on an Intel® Core i7-1265U processor with 32GB of RAM (https://github.com/bshahar/MAPF-AS).

For each map, we tested all four algorithms on 25 problem instances containing $k = \{2, 3, \dots\}$ randomly allocated agents, until all algorithms were not able to solve any of the problem instances. We set the time limit to one minute for each problem instance and measured the success rate and average time (in seconds) of each algorithm. The time was set to 60 seconds for an unsolved instance. Figure 4 and Figure 5 present the **success rate and average time**, respectively.

As expected, increasing the number of agents decreases the success rate and increases the time across all four algorithms. On the smallest map (empty-8-8), A$^*_{AS}$ was only able to solve problem instances with a small number of agents (up to 7 agents). On any larger map, A$^*_{AS}$ only solved a few problem instances with a small number of agents. This is due to the size of the shared state-space on which A$^*_{AS}$ is executed, which grows exponentially with the number of agents. While representing all optimal solutions by MAS, the reduction-based algorithm RED$_{AS}$ presented performance similar, to some extent, to A$^*_{AS}$. The CBS-based algorithms CBS$_{AS}$ and CBS-M$_{AS}$ performed better than both A$^*_{AS}$ and RED$_{AS}$. In most maps and number of agents, CBS-M$_{AS}$ outperformed CBS$_{AS}$, solved problem instances with more agents, and resulted in faster runtime; for instance, in empty-8-8, CBS-M$_{AS}$ solved problem instances with 29 agents while CBS$_{AS}$ was only able to solve problem instances with up to 22 agents. In some maps, e.g., empty-32-32, for a small number of agents, CBS$_{AS}$ outperformed CBS-M$_{AS}$; this is because, CBS$_{AS}$ can quickly merge its $MMDD$ and create $SMDD$ when only a few agents exist.

Table 1 presents the **average number of high-level expansions** performed by CBS$_{AS}$ and CBS-M$_{AS}$ for problem instances solved by both algorithms for 2, 4, 6, and 8 agents. CBS$_{AS}$, as expected, expands much fewer CT nodes than CBS-M$_{AS}$. Both CBS$_{AS}$ and CBS-M$_{AS}$ find a set of CT leaves in their high-level search. The high-level search of CBS$_{AS}$ halts sooner than the one of CBS-M$_{AS}$. However, as the CT leaves of CBS$_{AS}$ may contain conflicting $MDDs$, CBS$_{AS}$ constructs an $SMDD$ (conflict-free). When many agents exist, CBS$_{AS}$ invests a long time in constructing $SMDD$ after the high-level search halts. In fact, across all the maps and number of agents, CBS$_{AS}$ spent more than 90% of the time on constructing the $SMDD$. In contrast, CBS-M$_{AS}$ performs a deeper search on its high level (constructs a larger CT) to only find CT leaves that do not contain conflicting $MDDs$ and form a valid $MMDD$. As a result, CBS-M$_{AS}$ can immediately halt after its high level finishes.

It is common knowledge that, for finding a single optimal solution, A* must expand all nodes n with $f(n) < C^*$, where C^* is the cost of the optimal solution; otherwise, a suboptimal may be returned (Dechter and Pearl 1985). Such nodes are often denoted as *must-expand nodes* (MEN). For finding all optimal solutions, A$^*_{AS}$ must expand all nodes n with $f(n) \leq C^*$; otherwise, some optimal solutions may not be found. Therefore, for finding all optimal solutions, the MEN of A$^*_{AS}$ also contains nodes n with $f(n) = C^*$. Similarly, for finding a single optimal solution, CBS must expand all CT nodes N with $N.cost < C^*$ (whether $N.cost$ includes an admissible heuristic value or not). For finding all optimal solutions, CBS$_{AS}$ and CBS-M$_{AS}$ must expand all CT nodes N with $N.cost \leq C^*$; the main difference between the number of CT node expansions performed by CBS$_{AS}$ and CBS-M$_{AS}$ is the definition of a CT goal node which, as can be seen in Table 1, may have a significant

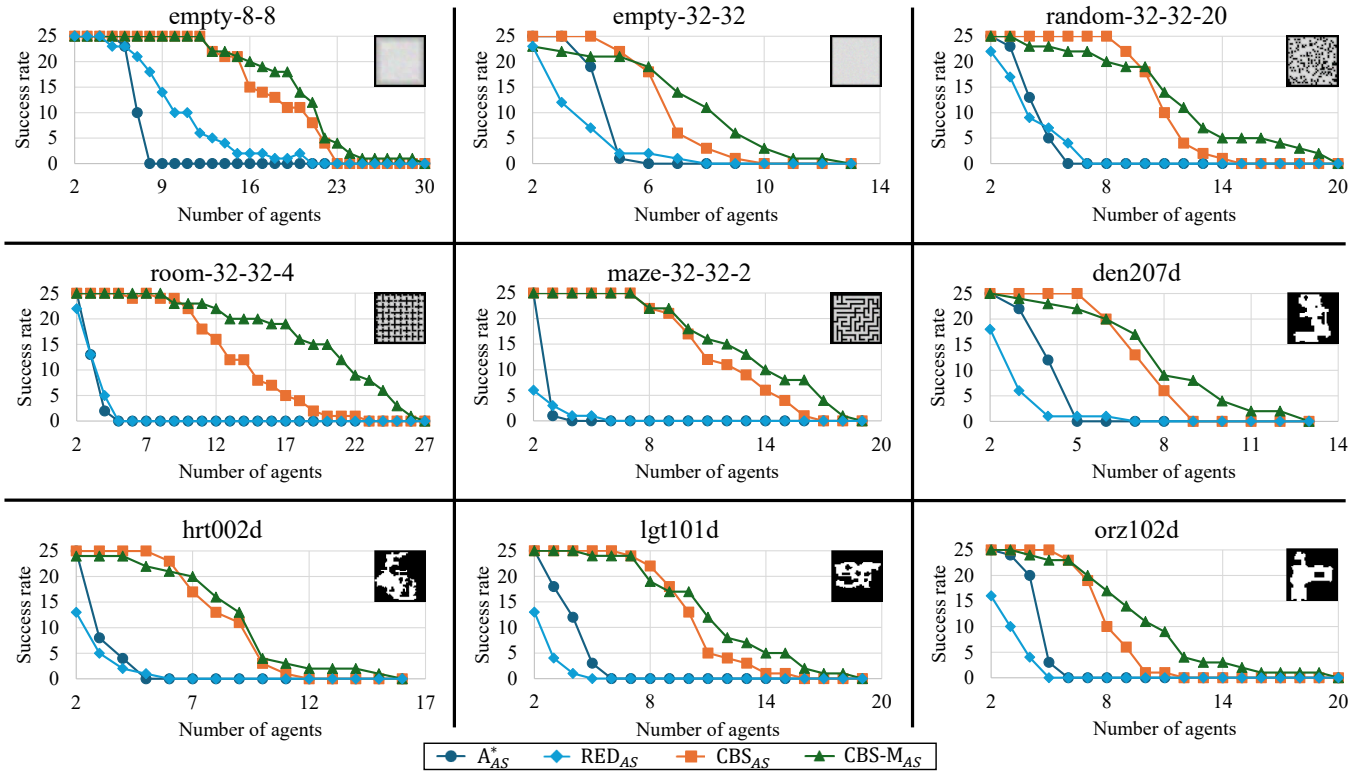

Figure 4: Success rate for A^*_{AS}, RED_{AS}, CBS_{AS}, and CBS-M_{AS} on nine benchmark maps.

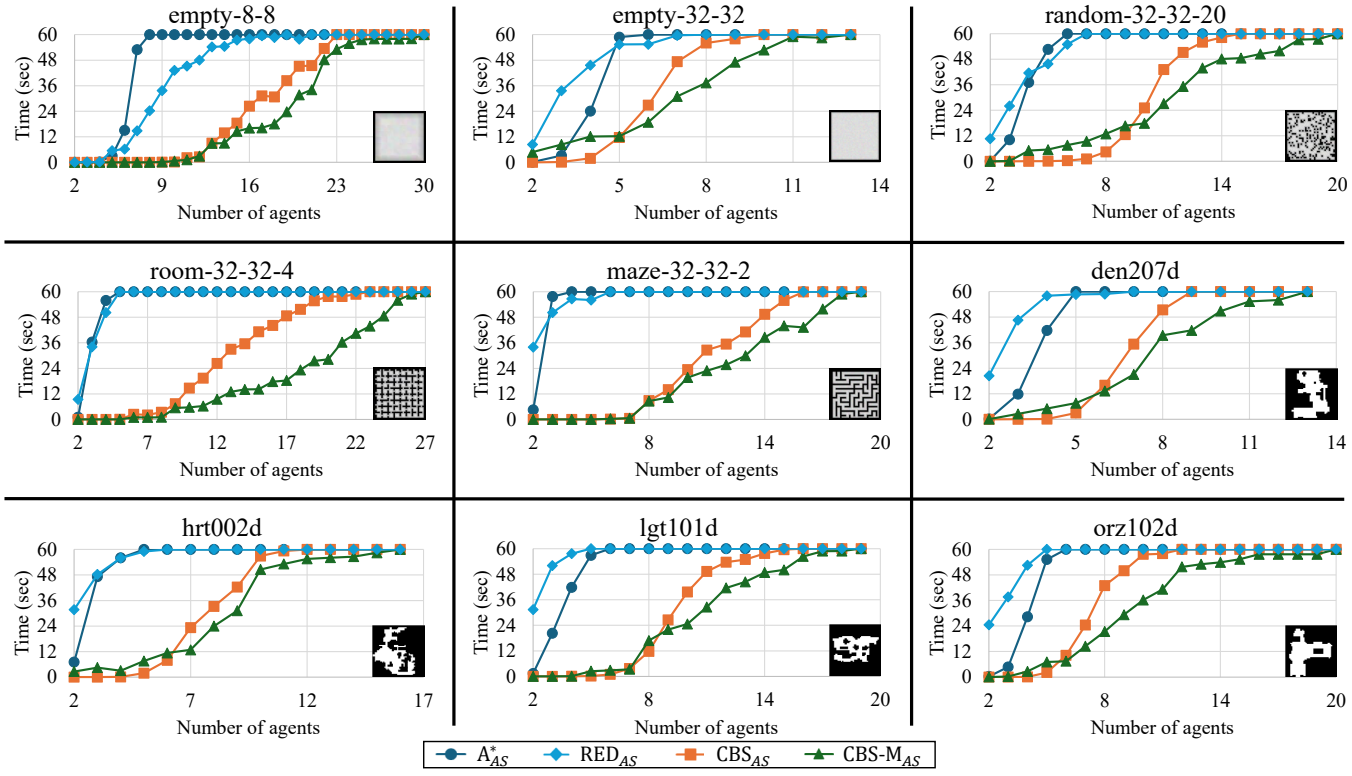

Figure 5: Average time (in seconds) for A^*_{AS}, RED_{AS}, CBS_{AS}, and CBS-M_{AS} on nine benchmark maps.

Algorithm	CBS$_{AS}$				CBS-M$_{AS}$			
#Agents	2	4	6	8	2	4	6	8
empty-8-8	1	3	3	4	2	22	91	226
empty-32-32	1	1	1	2	11	5,877	11,069	12,317
random-32-32-20	1	2	3	4	2	1,038	1,484	3,318
room-32-32-4	1	3	5	9	2	14	29	254
maze-32-32-2	2	3	17	34	3	32	1,610	9,827
den207d	1	1	1	1	339	936	2,665	6,760
hrt002d	1	2	3	3	646	1,667	3,665	8,386
lgt101d	1	2	2	4	19	400	2,944	15,459
orz102d	1	1	2	2	247	941	14,041	16,414

Table 1: Average number of high-level (CT) expansions.

#Agents	2	4	6	8
empty-8-8	1.7E+02	1.5E+03	1.3E+05	1.3E+06
empty-32-32	6.7E+16	3.9E+21	3.4E+28	4.2E+28
random-32-32-20	9.6E+10	1.5E+21	2.4E+26	1.8E+29
room-32-32-4	1.1E+06	4.2E+11	4.4E+13	3.1E+20
maze-32-32-2	2.1E+15	2.7E+26	1.4E+36	3.2E+44
den207d	9.8E+18	9.1E+31	3.6E+37	1.3E+40
hrt002d	9.6E+20	1.8E+33	3.7E+39	9.4E+42
lgt101d	1.2E+11	7.7E+21	3.7E+24	7.6E+28
orz102d	8.4E+14	8.6E+23	6.8E+28	5.6E+33

Table 2: Average number of optimal MAPF solutions.

impact on the expansions made by these algorithms. However, as demonstrated by Figures 4 and 5, expanding fewer CT nodes, as done by CBS$_{AS}$, requires the additional time-consuming construction of an $SMDD$. As CBS-M$_{AS}$ do not need such an operation, it results in superb performance.

For all maps, we also went over the represented solutions returned by the algorithms and counted the **average number of optimal solutions** found by the algorithms for 2, 4, 6, and 8 agents.[3] The results are presented in Table 2. The number of optimal solutions significantly increases with the number of agents. In the maze grids, for instance, for two agents, there were 2.1^{15} optimal solutions and, for eight agents, there were 3.2^{44} optimal solutions, on average.

While, in our results, the average number of optimal solutions increased with the number of agents, it was not always the case. Consider, for example, the problem instance presented in Figure 3(a), which contains two agents and has three optimal solutions. Now, consider a third agent that is added to the problem instance, as illustrated in Figure 6. For the new problem instance, only a single optimal solution exists, where agent a_1 goes through vertex A, agent a_2 goes through vertex C, and agent a_3 goes through vertex B.

8 Conclusions and Future Work

Multi-Agent Path Finding (MAPF) aims to find a set of conflict-free paths. In this paper, we study the requirement of finding all optimal MAPF solutions. We propose three ways for representing all optimal solutions (MAS, $SMDD$, and $MMDD$) and four algorithms for finding all optimal solutions (A$^*_{AS}$, RED$_{AS}$, CBS$_{AS}$, and CBS-M$_{AS}$). Our experiments show that (1) in most cases, CBS-M$_{AS}$ outperforms A$^*_{AS}$, RED$_{AS}$, and CBS$_{AS}$, in terms of success rate and runtime; (2) CBS-M$_{AS}$ requires expanding many more CT nodes than CBS$_{AS}$; and (3) the number of optimal solutions significantly increases with the number of agents.

[3]Counting the number of optimal solutions is simple for any of the three representation methods. For example, in $SMDD$, any path represents an optimal solution. Therefore, we can easily count the number of paths leading to each vertex starting from its root and considering all incoming edges to each vertex.

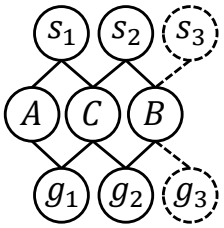

Figure 6: Adding an agent reduces the number of solutions.

There are many possible directions for future work.
(1) Future work can suggest other ways to represent all optimal solutions. Consider a graph where all the start and goal vertices are only connected to a vertex C. In this example, there are many different optimal solutions, which do not share many of their vertices. Therefore, for this example, our proposed representation methods may be impractical.
(2) Future work can study other related problems. For example, suggesting all optimal solutions to a human decision maker may be overwhelming. Therefore, finding a diverse subset of optimal solutions can be better in such a scenario.
(3) Future work may find all optimal solutions faster by applying CBS's improvements (Felner et al. 2018; Zhang et al. 2020; Li et al. 2021; Shen et al. 2023), or adjusting other algorithms, such as *ICTS* (Sharon et al. 2013), *BCP* (Lam et al. 2022), *M** (Wagner and Choset 2015), and *Lazy CBS* (Gange, Harabor, and Stuckey 2019).
(4) Future work can find all solutions for MAPF variants (Wan et al. 2018; Švancara et al. 2019; Morag et al. 2022; Atzmon et al. 2023; Maliah, Atzmon, and Felner 2025).

Acknowledgments

This research was supported by Israel's Ministry of Innovation, Science and Technology (Czech-Israeli cooperative scientific research) under grant #6908.

References

Aljazzar, H.; and Leue, S. 2011. K*: A heuristic search algorithm for finding the k shortest paths. *AIJ*, 175: 2129–2154.

Arthur, J.; Hachey, M.; Sahr, K.; Huso, M.; and Kiester, A. 1997. Finding all optimal solutions to the reserve site selection problem: formulation and computational analysis. *Environmental and Ecological Statistics*, 4: 153–165.

Atzmon, D.; Bernardini, S.; Fagnani, F.; and Fairbairn, D. 2023. Exploiting Geometric Constraints in Multi-Agent Pathfinding. In *ICAPS*, 17–25.

Atzmon, D.; Diei, A.; and Rave, D. 2019. Multi-Train Path Finding. In *SoCS*, 125–129.

Barták, R.; and Svancara, J. 2019. On SAT-Based Approaches for Multi-Agent Path Finding with the Sum-of-Costs Objective. In *SoCS*, 10–17.

Boyarski, E.; Chan, S.; Atzmon, D.; Felner, A.; and Koenig, S. 2022. On Merging Agents in Multi-Agent Pathfinding Algorithms. In *SoCS*, 11–19.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. E. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *IJCAI*, 740–746.

Byers, T. H.; and Waterman, M. S. 1984. Determining All Optimal and Near-Optimal Solutions When Solving Shortest Path Problems by Dynamic Programming. *Operations Research*, 32(6): 1381–1384.

Dechter, R.; and Pearl, J. 1985. Generalized Best-First Search Strategies and the Optimality of A^*. *J. ACM*, 32(3).

Eppstein, D. 1998. Finding the k Shortest Paths. *SICOMP*, 28(2): 652–673.

Erdem, E.; Kisa, D. G.; Oztok, U.; and Schueller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*, 290–296.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2018. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In *ICAPS*.

Gange, G.; Harabor, D.; and Stuckey, P. 2019. Lazy CBS: implicit Conflict-based Search using Lazy Clause Generation. In *ICAPS*, 155–162.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2019. Persistent and Robust Execution of MAPF Schedules in Warehouses. *IEEE RA-L*, 4(2): 1125–1131.

Isermann, H. 1977. The Enumeration of the Set of All Efficient Solutions for a Linear Multiple Objective Program. *J. Oper. Res. Soc.*, 28(3): 711–725.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A Novel Iterative Approach to Top-k Planning. In *ICAPS*.

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

Li, H.; Long, T.; Xu, G.; and Wang, Y. 2019a. Coupling-Degree-Based Heuristic Prioritized Planning Method for UAV Swarm Path Generation. In *CAC*, 3636–3641.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019b. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *IJCAI*, 442–449.

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

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019c. Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search. In *ICAPS*, 279–283.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2020. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *AAAI*, 11272–11281.

Mahadevan, R.; and Schilling, C. 2003. The effects of alternate optimal solutions in constraint-based genome-scale metabolic models. *Metabolic Engineering*, 5(4): 264–276.

Maliah, A.; Atzmon, D.; and Felner, A. 2025. Minimizing Makespan with Conflict-Based Search for Optimal Multi-Agent Path Finding. In *AAMAS*, 1418–1426.

Morag, J.; Felner, A.; Stern, R.; Atzmon, D.; and Boyarski, E. 2022. Online Multi-Agent Path Finding: New Results. In *SoCS*, 229–233.

Morag, J.; Zhang, Y.; Koyfman, D.; Chen, Z.; Felner, A.; Harabor, D.; and Stern, R. 2024. Prioritised Planning with Guarantees. In *SoCS*, 82–90.

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

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *AIJ*, 195: 470–495.

Shen, B.; Che, Z.; Li, J.; Cheema, M. A.; Harabor, D. D.; and Stuckey, P. J. 2023. Beyond Pairwise Reasoning in Multi-Agent Path Finding. In *ICAPS*, 384–392.

Siegmund, F.; Ng, A. H.; and Deb, K. 2012. Finding a preferred diverse set of Pareto-optimal solutions for a limited number of function calls. In *CEC*, 2417–2424.

Srinivasan, A.; Ham, T.; Malik, S.; and Brayton, R. K. 1990. Algorithms for discrete function manipulation. In *ICCAD*.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *SoCS*, 151–159.

Surynek, P. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *AAAI*, 1261–1263.

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 *ECAI*, 810–818.

Švancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online multi-agent pathfinding. In *AAAI*, 7732–7739.

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

Wan, Q.; Gu, C.; Sun, S.; Chen, M.; Huang, H.; and Jia, X. 2018. Lifelong Multi-Agent Path Finding in A Dynamic Environment. In *ICARCV*, 875–882.

Yen, J. Y. 1971. Finding the k shortest loopless paths in a network. *management Science*, 17(11): 712–716.

Yu, J.; and LaValle, S. M. 2013a. Multi-agent path planning and network flow. In *Algorithmic foundations of robotics X*, 157–173. Springer.

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

Zhang, H.; Li, J.; Surynek, P.; Koenig, S.; and Kumar, T. K. S. 2020. Multi-Agent Path Finding with Mutex Propagation. In *ICAPS*, 323–332.

Zhou, N.-F.; Kjellerstrand, H.; and Fruhman, J. 2015. *Constraint solving and planning with Picat*. Springer.