Binary Branching Multi-Objective Conflict-Based Search for Multi-Agent Path Finding

Zhongqiang Ren[1*], Jiaoyang Li[1*], Han Zhang[2], Sven Koenig[2], Sivakumar Rathinam[3], Howie Choset[1]

[1]Carnegie Mellon University, Pittsburgh, PA 15213, USA
[2]University of Southern California, Los Angeles, CA 90007, USA
[3]Texas A&M University, College Station, TX 77843, USA
zhongqir@andrew.cmu.edu, jiaoyangli@cmu.edu, zhan645@usc.edu,
skoenig@usc.edu, srathinam@tamu.edu, choset@andrew.cmu.edu

Abstract

This paper considers a multi-agent multi-objective path-finding problem that requires not only finding collision-free paths for multiple agents from their respective start locations to their respective goal locations but also optimizing multiple objectives simultaneously. In general, there is no single solution that optimizes all the objectives simultaneously, and the problem is thus to find the so-called Pareto-optimal frontier. To solve this problem, an algorithm called Multi-Objective Conflict-Based Search (MO-CBS) was recently developed and is guaranteed to find the exact Pareto-optimal frontier. However, MO-CBS does not scale well with the number of agents due to the large branching factor of the search, which leads to a lot of duplicated effort in agent-agent collision resolution. This paper therefore develops a new algorithm called Binary Branching MO-CBS (BB-MO-CBS) that reduces the branching factor as well as the duplicated collision resolution during the search, which expedites the search as a result. Our experimental results show that BB-MO-CBS reduces the number of conflicts by up to two orders of magnitude and often doubles or triples the success rates of MO-CBS on various maps given a runtime limit.

1 Introduction

Multi-Agent Path Finding (MAPF), as its name suggests, requires finding collision-free paths for multiple agents from their respective start locations to their respective goal locations, while optimizing a scalar measure of the paths. MAPF arises in applications such as warehouse logistics (Wurman, D'Andrea, and Mountz 2008) and airport surface management (Morris et al. 2016). Variants of MAPF have been widely studied over the last few years (Stern et al. 2019). This paper considers a generalization of MAPF to multiple objectives, called *Multi-Agent Multi-Objective Path Finding* (MA-MO-PF) (Weise et al. 2020; Ren, Rathinam, and Choset 2021, 2022). In MA-MO-PF, agents have to trade off one objective for another, such as arrival times, travel risks, and path lengths. MA-MO-PF is a generalization of MAPF and is, in general, NP-Hard to solve to optimality (Yu and LaValle 2013). In general, there does not exist a single MA-MO-PF solution that simultaneously optimizes all the objectives. The goal of MA-MO-PF is thus to find a Pareto-optimal set of solutions, whose corresponding cost vectors (where each component corresponds to an objective to be minimized) form the so-called Pareto-optimal frontier.

A solution is Pareto-optimal if one cannot improve on one objective without deteriorating at least one of the other objectives. Ultimately, the challenge of the MA-MO-PF problem is to compute the Pareto-optimal frontier, which requires both resolving agent-agent collision and optimizing multiple objectives. To address this challenge, evolutionary algorithms (Weise et al. 2020) were proposed to quickly approximate the Pareto-optimal frontier, while search-based algorithms (Ren, Rathinam, and Choset 2021, 2022) were developed to compute the exact Pareto-optimal frontier. We are interested in search algorithms in this paper.

Among these search algorithms, Multi-Objective Conflict-Based Search (MO-CBS) (Ren, Rathinam, and Choset 2022) is a leading approach that leverages both Conflict-Based Search (CBS) (Sharon et al. 2015) for (single-objective) MAPF and the dominance principle from multi-objective optimization (Ehrgott 2005). Similar to CBS, MO-CBS is a two-level search algorithm. It begins by finding a Pareto-optimal set of (individual) paths for each single agent ignoring any agent-agent collision and then takes all combinations of these paths over all the agents to form a set of joint paths, where a joint path is a set of paths with one path for each agent. MO-CBS then creates a constraint tree (Sharon et al. 2015) for each joint path for the high-level search. In a joint path, if two agents occupy the same location at the same time, a conflict happens. To resolve a conflict between two agents, a constraint is added to either of the agents, and the low-level search finds a new Pareto-optimal set of paths for that agent while satisfying all constraints. Each of these paths results in a new branch for future search, and the number of branches (i.e., the branching factor) is often large, as each agent may have many Pareto-optimal paths.

This paper relies on our key observation that the high-level search of MO-CBS often resolves the same conflict many times, which reduces the computational efficiency

*These authors contributed equally.

of MO-CBS. To reduce the wasted computational effort, we develop a new algorithm called Binary Branching MO-CBS (BB-MO-CBS), which (i) employs a new high-level search to update all joint paths intelligently, when a conflict is resolved and enjoys the small constant branching factor of two; and (ii) is able to discard dominated joint paths during the search and thus avoids the duplicated conflict resolution effort for the discarded joint paths. Consequently, BB-MO-CBS runs much faster than MO-CBS. We prove that BB-MO-CBS is guaranteed to find the exact Pareto-optimal frontier. Our experimental results show that BB-MO-CBS reduces the number of conflicts by up to two orders of magnitude. As a result, with a runtime limit of 300 seconds per test instance, BB-MO-CBS often doubles or triples the success rates of MO-CBS on various maps.

2 Problem Definition

Let index set $I = \{1, 2, \dots, N\}$ denote a set of N agents. We use $i, j \in I$ to denote agents. All agents move in a shared workspace, which is represented as a finite undirected graph $G = (V, E)$, where the vertex set V denotes all possible locations of the agents, and the edge set $E \subseteq V \times V$ denotes all possible actions that can move an agent between two vertices in V. An edge between $u, v \in V$ is denoted as $(u, v) \in E$, and the cost of $e \in E$ is an M-dimensional vector: $\text{cost}(e) \in (0, \infty)^M$ with M being a positive integer and each component in $\text{cost}(e)$ being a finite positive real value.

We use a superscript $i, j \in I$ with a variable to indicate the agent that the variable relates to (e.g., $v^i \in V$ means the vertex occupied by agent i). Let $\pi^i(v_1^i, v_\ell^i)$ represent a path that connects vertices v_1^i and v_ℓ^i via a sequence of vertices $(v_1^i, v_2^i, \dots, v_\ell^i)$ in G. Let $\vec{g}(\pi^i(v_1^i, v_\ell^i))$ (as opposed to a scalar g) denote the M-dimensional *cost vector* (also referred to as *path cost*) associated with the path, which is the sum of the cost vectors of all edges present in the path, i.e., $\vec{g}(\pi^i(v_1^i, v_\ell^i)) = \Sigma_{j=1,2,\dots,\ell-1}\text{cost}((v_j^i, v_{j+1}^i))$.

All agents share a global clock and start to move along their paths at time $t = 0$. Each action, either to wait or move along an edge, for any agent requires one unit of time. Agents $i, j \in I$ are in *conflict* iff (i.e., if and only if) one of the following two cases happens. The first case is a *vertex conflict* where two agents occupy the same vertex at the same time. The second case is an *edge conflict* where two agents traverse the same edge from opposite directions at the same time.

Let $v_o^i, v_d^i \in V$ denote the initial vertex and the destination of agent i. To simplify the notation, we also refer to a path $\pi^i(v_o^i, v_d^i)$ for agent i between its initial vertex and destination simply as π^i. Let $\pi = (\pi^1, \pi^2, \dots, \pi^N)$ represent a *joint path* for all agents. The cost vector of a joint path is defined by the vector sum of the individual path costs over all agents, i.e., $\vec{g}(\pi) = \Sigma_i \vec{g}(\pi^i)$. In this work, an agent $i \in I$ eventually stays at v_d^i and incurs no cost from then on.

To compare any two paths or joint paths, we compare their cost vectors. Given two vectors $\vec{a}$ and $\vec{b}$, vector $\vec{a}$ is said to *dominate* vector $\vec{b}$ iff every component in $\vec{a}$ is no larger than the corresponding component in $\vec{b}$ and there exists at least one component in $\vec{a}$ that is smaller than the corresponding component in $\vec{b}$. Let $\vec{a}(m), m \in \{1, 2, \cdots, M\}$ denote the m-th component in $\vec{a}$.

Definition 1 (Dominance (Ehrgott 2005)) *Given two vectors $\vec{a}$ and $\vec{b}$ of length M, $\vec{a}$ dominates $\vec{b}$, notationally $\vec{a} \prec \vec{b}$, iff $\forall m \in \{1, 2, \dots, M\}$, $\vec{a}(m) \leq \vec{b}(m)$, and $\exists m \in \{1, 2, \dots, M\}$, $\vec{a}(m) < \vec{b}(m)$.*

Any two joint paths (or two individual paths) are undominated with respect to each other iff their cost vectors do not dominate each other. A joint path π is undominated with respect to a set of joint paths Π if π is undominated by any $\pi' \in \Pi$. Among all conflict-free joint paths (i.e., solutions), the set of all undominated ones is called the *Pareto-optimal set*. Given the graph G, v_o^i and $v_d^i, i \in I$, the MA-MO-PF problem seeks to find all *cost-unique* Pareto-optimal solutions, i.e., any maximal subset of the Pareto-optimal set, where any two solutions in this subset do not have the same cost vector. The cost vectors of the solutions in this subset form the *Pareto-optimal frontier*.

Definition 2 *Given two vectors $\vec{a}$ and $\vec{b}$ of length M, $\vec{a}$ weakly dominates $\vec{b}$, notationally $\vec{a} \preceq \vec{b}$, iff $\forall m \in \{1, 2, \dots, M\}$, $\vec{a}(m) \leq \vec{b}(m)$.*

$\vec{a} \preceq \vec{b}$ is equivalent to that $\vec{a}$ dominates or is equal to $\vec{b}$. Finally, given a set Π of joint paths, let *ND*(Π) denote a maximal cost-unique undominated subset of Π.

3 Preliminaries

This section reviews CBS (Sharon et al. 2015) and MO-CBS (Ren, Rathinam, and Choset 2022). The concepts and notations in this section will be used later.

3.1 Review of CBS

Conflict-Based Search (CBS) is a two-level search algorithm that finds a minimum-cost solution for any solvable (single-objective) MAPF. On the high level, every node P is defined as a tuple (π, g, Ω), where:

- Ω is a set of constraints. Each constraint has form (i, v, t) (or (i, e, t)), which means that agent i is forbidden to occupy vertex v (or to traverse edge e) at time t (between t and $t + 1$).
- $\pi = (\pi^1, \pi^2, \dots, \pi^N)$ is a joint path that connects v_o^i and v_d^i for all agents. Each path π^i in π is a minimum-cost path that satisfies the constraints in Ω related to agent i.
- g is the scalar cost value of π (i.e., $g = g(\pi)$).

CBS constructs a constraint tree with the root node $P_o = (\pi_o, g(\pi_o), \emptyset)$, where π_o is constructed by running the low-level (single-agent) planner, such as A*, for every agent with an empty constraint set while ignoring all other agents. P_o is then added to OPEN, a queue that prioritizes nodes based on their g-values from the minimum to the maximum.

In each search iteration, a node $P = (\pi, g, \Omega)$ with the minimum g-value is popped from OPEN for expansion. To expand P, every pair of individual paths in π is checked for a vertex conflict (i, j, v, t) (and an edge conflict (i, j, e, t)), which means agents i and j are in conflict at vertex v (and edge e, respectively) at time t. If no conflict is detected, π

Algorithm 1 Pseudocode for MO-CBS and MO-CBS-t

1: Compute $\{\Pi^i, \forall i \in I\}$, create root nodes and add them to OPEN.
2: $\mathcal{C} \leftarrow \emptyset$
3: **while** OPEN is not empty **do**
4: $P_k = (\pi_k, \vec{g}_k, \Omega_k) \leftarrow$ OPEN.pop()
5: **if** *SolutionFilter*($\mathcal{C}$, P_k) **then continue**
6: *cft*$\leftarrow$*DetectConflict*(π_k)
7: **if** *cft*$= NULL$ **then**
8: *SolutionUpdate*($\mathcal{C}$, P_k)
9: add $\vec{g}_k$ to $\mathcal{C}$
10: **continue**
11: $\{\omega^i, \omega^j\} \leftarrow$ *GenerateConstraints*(*cft*)
12: **for all** $i' \in \{i, j\}$ **do**
13: $\Omega_l = \Omega_k \cup \{\omega^{i'}\}$
14: $\Pi^{i'}_* \leftarrow$ *LowLevelSearch*(i', Ω_l)
15: **for all** $\pi^{i'}_* \in \Pi^{i'}_*$ **do**
16: $\pi_l \leftarrow \pi_k$, then replace $\pi^{i'}_l$ (in π_l) with $\pi^{i'}_*$
17: $\vec{g}_l \leftarrow \vec{g}(\pi_l)$
18: $P_l = (\pi_l, \vec{g}_l, \Omega_l)$
19: **if** not *SolutionFilter*($\mathcal{C}$, P_l) **then**
20: add P_l to OPEN
21: **return** $\mathcal{C}$

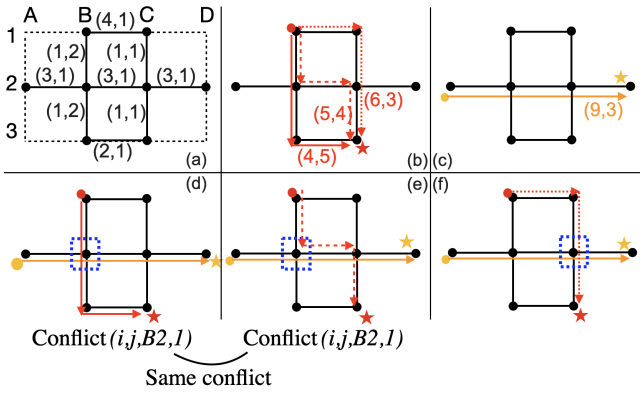

Figure 1: A toy example where the same conflict is resolved multiple times in different constraint trees during the search of MO-CBS. (a) shows the graph where the numbers in parentheses are edge cost vectors. (b) and (c) show the individual Pareto-optimal paths for the red (i) and the orange (j) agent. (d), (e), and (f) are the three root nodes created when initializing MO-CBS. Each blue dashed box shows a conflict. (d) and (e) show that the same conflict is resolved in two different constraint trees during the search. Our BB-MO-CBS is able to resolve both conflicts in (d) and (e) at the same time.

is conflict-free and returned as an optimal solution. Otherwise, to resolve the detected conflict (i, j, v, t), two constraints $\omega^i = (i, v, t)$ and $\omega^j = (j, v, t)$ are generated, and two new corresponding constraint sets $\Omega \bigcup \{(i, v, t)\}$ and $\Omega \bigcup \{(j, v, t)\}$ are created. Edge conflicts are handled in a similar manner, which is thus omitted. Then, for each new constraint (i, v, t) and the corresponding constraint set $\Omega' = \Omega \bigcup \{(i, v, t)\}$, the low-level planner is invoked to plan a minimum-cost path π'^i for agent i while satisfying all the constraints related to agent i in Ω'. The low-level planner typically runs an A*-like search in a time-augmented graph G' with vertices $V \times \{0, 1, 2, \cdots\}$ with constraints representing inaccessible vertices or edges in this graph G'. A joint path $\pi' = (\pi^1, \pi^2, \cdots, \pi'^i, \cdots, \pi^N)$ is then formed and a new node $(\pi', g(\pi'), \Omega')$ is created and added to OPEN.

3.2 Review of MO-CBS

MO-CBS (Ren, Rathinam, and Choset 2022) (Alg. 1) extends CBS to optimize multiple objectives. During initialization, MO-CBS invokes a single-agent multi-objective A* planner to find all cost-unique Pareto-optimal paths Π^i_o for each agent $i \in I$. MO-CBS then takes all combinations of these individual paths over all agents to form a set of joint paths $\Pi_o = \Pi^1_o \times \Pi^2_o \times \cdots \times \Pi^N_o$, and each joint path $\pi \in \Pi_o$ is then used to instantiate a root node $(\pi, \vec{g}(\pi), \emptyset)$. The definition of a node in MO-CBS is similar to the one in CBS with the only difference that the scalar-cost $g(\pi)$ in CBS becomes a vector-cost $\vec{g}(\pi)$ in MO-CBS. There are $|\Pi^1_o| \times |\Pi^2_o| \times \cdots \times |\Pi^N_o|$ root nodes in total, all of which are added to OPEN, which is a queue that prioritizes the nodes based on their $\vec{g}$-vectors in lexicographic order from the minimum to the maximum. During the search, MO-CBS maintains a set $\mathcal{C}$ of all undominated cost vectors of solutions found so far.[1]

After the initialization, in every search iteration, a node $P_k = (\pi_k, \vec{g}_k, \Omega_k)$ with the lexicographic minimum $\vec{g}$-vector is popped from OPEN for expansion.[2] MO-CBS checks if $\vec{g}_k$ is weakly dominated by any cost vector in $\mathcal{C}$ (Line 5 in Alg. 1, *SolutionFilter*). If yes, then MO-CBS discards P_k since it cannot lead to a cost-unique Pareto-optimal solution, and we say P_k is *filtered*. Otherwise, MO-CBS checks P_k for a conflict (Line 6). If there is no conflict, then π_k is a solution, and MO-CBS removes any cost vectors in $\mathcal{C}$ that are dominated by $\vec{g}_k$ (Line 8) before adding $\vec{g}_k$ to $\mathcal{C}$ (Line 9). If there is a conflict, then MO-CBS resolves the conflict in a similar way to CBS (Lines 11-20). The differences between MO-CBS and CBS are: (i) the low-level search of MO-CBS invokes a single-agent multi-objective planner, which returns a set of cost-unique Pareto-optimal paths that respect all constraints; (ii) MO-CBS generates a node for each path that is returned by the low-level search (Lines 16-18); and (iii) MO-CBS compares the cost vector $\vec{g}_l$ of each generated node P_l against the cost vectors in $\mathcal{C}$ for filtering (Line 19) before adding P_l to OPEN.

MO-CBS is guaranteed to find the Pareto-optimal frontier for any solvable MA-MO-PF. Additionally, since the number $|\Pi^1_o| \times |\Pi^1_o| \times \cdots \times |\Pi^N_o|$ of root nodes can grow quickly with the number of agents, it becomes computationally pro-

[1] We only mention storing (and returning) the solution cost vectors since it is trivial to store (and return) the corresponding solutions as well, which form the desired cost-unique Pareto-optimal set (of solutions).

[2] We use subscripts k and l to denote a specific node (e.g., P_k) or a specific element (e.g. Π_k) in the corresponding node P_k.

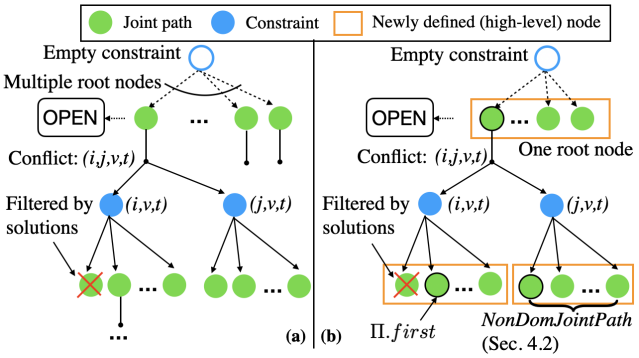

Figure 2: A conceptual visualization of MO-CBS and BB-MO-CBS with the differences discussed in Sec. 4.3.

hibitive to generate all root nodes before the search. To bypass this difficulty, a variant of MO-CBS, called MO-CBS-t, modifies Line 4 in Alg. 1 by popping nodes from OPEN in a "tree-by-tree" fashion. By doing so, MO-CBS-t can generate root nodes on demand and each constraint tree (rooted in a generated root node) is exhaustively searched before a new root node is generated.

A limitation of MO-CBS (and MO-CBS-t) is that a conflict might be repeatedly resolved in multiple constraint trees and multiple branches of the same constraint tree, which can make MO-CBS inefficient. We show a toy example in Fig. 1 and present a new algorithm (with a different high-level search) in the following that addresses this issue.

4 Method

This section begins with an overview of BB-MO-CBS and explains the key procedures in the subsequent subsections.

4.1 Algorithm Description

The first key idea behind BB-MO-CBS is to update *all* joint paths when a conflict is resolved, rather than updating only a single joint path in a single constraint tree as in MO-CBS, which is conceptually visualized in Fig. 2. The second key idea is to consider only the undominated subset of joint paths during the search rather than all joint paths (Sec. 4.3).

We begin by redefining what a node is in BB-MO-CBS. A node is a tuple $P = (\Pi, \pi_{rep}, \vec{g}_{rep}, \Omega, \{\Pi^i, \forall i \in I\})$, where:

- Π is a set of cost-unique joint paths such that (i) any pair of joint paths in Π are undominated by each other, and (ii) all joint paths in Π are lexicographically sorted based on their $\vec{g}$-vectors from the minimum to the maximum;[3]
- π_{rep} is the *representative joint path* of node P, which is the first joint path in Π (denoted as $\Pi.first$). In other words, π_{rep} is the joint path in Π with the lexicographic minimum cost vector;
- $\vec{g}_{rep}$ is the *representative (cost) vector* of node P, which is the same as $\vec{g}(\pi_{rep})$;
- Ω is a constraint set as used by CBS and MO-CBS; and
- $\{\Pi^i, \forall i \in I\}$ is a set of cost-unique Pareto-optimal paths for each agent $i \in I$ that satisfy the constraints in Ω. The following relationship holds for each node: $\Pi = ND(\Pi^1 \times \Pi^2 \times \cdots \times \Pi^N)$. We denote $\{\Pi^i, \forall i \in I\}$ simply as $\{\Pi^i\}$ without confusion.

BB-MO-CBS is shown in Alg. 2. During initialization (Line 2), BB-MO-CBS first plans for each agent i a set of Pareto-optimal paths Π_o^i ignoring other agents and then computes the cost-unique undominated subset $ND(\Pi_o)$ of $\Pi_o = \Pi_o^1 \times \Pi_o^2 \times \cdots \times \Pi_o^N$ by invoking a procedure *NonDomJointPath*, which is explained in the next section. There is a difference between $ND(\Pi_o)$ and *NonDomJointPath*$(\{\Pi_o^i\})$: $ND(\Pi_o)$ is a set of joint paths, while *NonDomJointPath* is a procedure that computes $ND(\Pi_o)$. BB-MO-CBS then generates the initial node $P_o = (ND(\Pi_o), \pi_{o,rep}, \vec{g}_{o,rep}, \emptyset, \{\Pi_o^i\})$, which forms the root node of the constraint tree. The root node is then added to OPEN, which is a queue that prioritizes nodes based on the lexicographic order of their representative vectors from the minimum to the maximum. During the search, BB-MO-CBS maintains a set $\mathcal{C}$ of all undominated cost vectors of solutions found so far (Line 3).

After the initialization, BB-MO-CBS iteratively selects a node $P_k = (\Pi_k, \pi_{k,rep}, \vec{g}_{k,rep}, \Omega_k, \{\Pi_k^i\})$ with the lexicographic minimum representative vector from OPEN for expansion. BB-MO-CBS invokes the procedure *DomPrune* (Line 6), which iterates the joint paths in Π_k and removes joint paths whose cost vectors are weakly dominated by any vector in $\mathcal{C}$. We explain *DomPrune* in the next section. Afterwards, if at least one joint path was removed from Π_k by *DomPrune* and Π_k is non-empty, then BB-MO-CBS re-calculates the representative path and the representative vector of node P_k (Line 9), and adds P_k back to OPEN (Line 10). If no joint path is removed from Π_k by *DomPrune*, then BB-MO-CBS checks $\pi_{k,rep}$ for a conflict (Line 12). If no conflict is detected (Lines 13), then $\pi_{k,rep}$ is a cost-unique Pareto-optimal solution. In this case, its cost vector $\vec{g}_{k,rep}$ is added to $\mathcal{C}$, and $\pi_{k,rep}$ is deleted from Π_k. Afterwards, if Π_k is not empty, then node P_k (with a new $\pi_{k,rep} \leftarrow \Pi_k.first$ and $\vec{g}_{k,rep} \leftarrow \vec{g}(\pi_{k,rep})$) is added to OPEN (Lines 17-18) for future expansion.

If a conflict is detected in $\pi_{k,rep}$, then the conflict is resolved in the same way as in CBS, namely by generating two constraints (Line 20). For each newly generated constraint ω^i, BB-MO-CBS makes a copy of the set of paths $\{\Pi_k^i\}$ (Line 22), and generates a new constraint set Ω_l by adding ω^i to Ω_k (Line 23). Then, BB-MO-CBS invokes the low-level search for agent i (Line 24), which finds all cost-unique Pareto-optimal paths Π_*^i that satisfy all constraints related to agent i in Ω_l. BB-MO-CBS replaces the path set Π_l^i with Π_*^i (Line 25), and recompute the joint path set $\Pi_l = ND(\Pi_l^1 \times \Pi_l^2 \times \cdots \times \Pi_l^N)$ by invoking the procedure *NonDomJointPath* (Line 26). $\Pi_l.first$ becomes the representative path $\pi_{l,rep}$ of the new node P_l (Line 29), and its cost vector $\vec{g}_{l,rep}$ becomes the representative vector of P_l.

[3] There is no need to generate and store Π (which is often a large set). Instead, one can store the set B of undominated $\vec{g}$-vector of each joint path in $\pi \in \Pi$. Each $\vec{b} \in B$ is the sum of $\vec{g}(\pi^i), \pi^i \in \Pi^i$ for all $i \in I$, by storing this correspondence between $\vec{b}$ and $\pi^i, i \in I$, one can readily reconstruct the joint path π for any $\vec{b} \in B$.

Algorithm 2 Pseudocode for BB-MO-CBS

1: $P_o \leftarrow (ND(\Pi_o), \pi_{o,rep}, \vec{g}_{o,rep}, \emptyset, \{\Pi_o^i\})$
2: add P_o to OPEN
3: $\mathcal{C} \leftarrow \emptyset$
4: **while** OPEN is not empty **do**
5: $P_k = (\Pi_k, \pi_{k,rep}, \vec{g}_{k,rep}, \Omega_k, \{\Pi_k^i\}) \leftarrow$ OPEN.pop
6: $\Pi_k \leftarrow$*DomPrune*($\mathcal{C}$, Π_k)
7: **if** any joint path in Π_k is filtered **then**
8: **if** $\Pi_k \neq \emptyset$ **then**
9: $\pi_{k,rep} \leftarrow \Pi_k.first$, $\vec{g}_{k,rep} \leftarrow \vec{g}(\pi_{k,rep})$
10: add P_k to OPEN
11: **continue**
12: *cft*$\leftarrow$*DetectConflict*($\pi_{k,rep}$)
13: **if** *cft*$= NULL$ **then**
14: add $\vec{g}_{k,rep}$ to $\mathcal{C}$
15: delete $\pi_{k,rep}$ from Π_k
16: **if** $\Pi_k \neq \emptyset$ **then**
17: $\pi_{k,rep} \leftarrow \Pi_k.first$, $\vec{g}_{k,rep} \leftarrow \vec{g}(\pi_{k,rep})$
18: add P_k to OPEN
19: **continue**
20: $\{\omega^i, \omega^j\} \leftarrow$ *GenerateConstraints*(*cft*)
21: **for all** $i' \in \{i, j\}$ **do**
22: $\{\Pi_l^i\} \leftarrow \{\Pi_k^i\}$
23: $\Omega_l \leftarrow \Omega_k \cup \{\omega^{i'}\}$
24: $\Pi_*^{i'} \leftarrow$ *LowLevelSearch*(i', Ω_l)
25: replace $\Pi_l^{i'}$ (in $\{\Pi_l^i\}$) with $\Pi_*^{i'}$
26: $\Pi_l \leftarrow$ *NonDomJointPath*($\{\Pi_l^i\}$)
27: **if** $\Pi_l = \emptyset$ **then continue**
28: $\vec{\pi}_{l,rep} \leftarrow \Pi_l.first$, $\vec{g}_{l,rep} \leftarrow \vec{g}(\pi_{l,rep})$
29: add $P_l = (\Pi_l, \pi_{l,rep}, \vec{g}_{l,rep}, \Omega_l, \{\Pi_l^i\})$ to OPEN
30: **return** $\mathcal{C}$

P_l is added to OPEN for future expansion.

BB-MO-CBS terminates when OPEN is empty and then returns the set $\mathcal{C}$, which the Pareto-optimal frontier of the given MA-MO-PF problem instance (see Sec. 5).

4.2 Key Procedures of BB-MO-CBS

Filtering Solutions Procedure *DomPrune* removes (i.e., filters) all joint paths of a node that are weakly dominated by existing solutions, similar to *SolutionFilter* of MO-CBS. However, due to the new high-level search of BB-MO-CBS, *DomPrune* works differently from *SolutionFilter*. As shown in Alg. 3, *DomPrune* iterates over the given set of joint paths Π, where the joint paths are lexicographically sorted in increasing order of their cost vectors. *DomPrune* ensures that the first joint path in Π is not weakly dominated by a solution already found during the search. It does this by repeatedly removing the first joint path from Π until the first joint path in Π is no longer weakly dominated by a solution already found during the search. *DomPrune* needs to ensure this property only for the first joint path in Π since Alg. 2 operates only on that joint path (i.e., use it as the representative joint path of the correpsonding node for conflict detection). *DomPrune* could ensure this property for all joint paths in Π but then would often be less efficient since Π can be large.

Algorithm 3 Pseudocode for *DomPrune*

Input: $\mathcal{C}$ is the set of solution cost vectors, and Π is a set of joint paths of some node.
Output: Π filtered by $\mathcal{C}$.
1: **for all** $\pi \in \Pi$ **do**
2: **if** $\exists \vec{c} \in \mathcal{C}$, $\vec{c} \preceq \vec{g}(\pi)$ **then**
3: remove π from Π
4: **else break**
5: **return** Π

Algorithm 4 Pseudocode for *NonDomJointPath*

Input: $\{\Pi^i\}$.
Output: *ND*($\Pi^1 \times \Pi^2 \times \cdots \times \Pi^N$).
1: Let A^i denote the cost vectors of paths in Π^i.
2: $B \leftarrow A^1$
3: **for all** $j = 2, 3, \ldots, N$ **do**
4: $B \leftarrow \{b + a \ , \ b \in B, a \in A^j\}$
5: $B \leftarrow$ *NonDomVec*(B)
6: **return** the joint paths corresponding to B

Computing Undominated Subsets The procedure *NonDomJointPath* computes a cost-unique undominated maximal subset of a given set of joint paths. A naive implementation would first compute $\Pi = \Pi^1 \times \Pi^2 \times \cdots \times \Pi^N$ and then iterate over each pair of joint paths in Π to find a maximal cost-unique undominated subset. However, this naive implementation is computationally prohibitive since the size of Π is often large. For example, if $|\Pi^i| = 10, \forall i \in I$, and there are ten agents ($N = 10$), then $|\Pi| = 10^{10}$.

Instead of calculating *ND*($\Pi^1 \times \Pi^2 \times \cdots \times \Pi^N$), we could calculate *ND*($\cdots$(*ND*(*ND*($\Pi^1 \times \Pi^2$) $\times \cdots$) $\times \Pi^N$). Let A^i denote the cost vectors of paths in Π^i. It is faster to perform this calculation with the set A^i as follows. Alg. 4 calculates $B =$ *NonDomVec*($\cdots$ *NonDomVec*(*NonDomVec*($A^1 + A^2$) + $\cdots$) + A^N), where $A^i + A^j$ means taking the sum of every pair of cost vectors $\vec{a}^i \in A^i$ and $\vec{a}^j \in A^j$, and *NonDomVec* computes the undominated subset of the input cost vectors (Lines 2-5 in Alg. 4). The joint paths corresponding to B can then be reconstructed at the end of Alg. 4. For each cost vector $\vec{b} \in B$, we know that $\vec{b} = \sum_{i \in I} \vec{g}(\pi^i), \pi^i \in \Pi^i$. Given $\vec{g}(\pi^i)$, this reconstruction can look up the corresponding π^i in Π^i since $\{\Pi^i\}$ is stored in each node. In practice, we only need to reconstruct the joint path that corresponds to the lexicographically minimum cost vector of B (which is used as the representative joint path of a node in Alg. 2).

An important procedure in Alg. 4 is *NonDomVec*, which returns the undominated subset of a given set of vectors B. A naive implementation iterates over every pair of vectors in B, resulting in a runtime complexity of $O(|B|^2)$. To expedite the computation, we leverage Kung's method (Kung, Luccio, and Preparata 1975), which is a fast algorithm to compute an undominated subset of vectors.

4.3 Relationship of BB-MO-CBS with MO-CBS and CBS

Relationship of BB-MO-CBS and MO-CBS The main differences between BB-MO-CBS and MO-CBS are:

- *Searching a single constraint tree with the branching factor of two*: In MO-CBS, each node contains one joint path, while BB-MO-CBS introduces a new notion of nodes where each node contains a set of undominated joint paths. Correspondingly, MO-CBS searches multiple constraint trees with varying branching factors of at least two (but which are often larger than two), while BB-MO-CBS searches only one constraint tree with the constant branching factor of two.
- *Ignoring dominated joint paths*: During the search, while MO-CBS has to consider all joint paths for conflict resolution,[4] BB-MO-CBS considers only a maximal subset of undominated joint paths in each node for conflict resolution, since the dominated joint paths are discarded in *NonDomJointPath* (Line 26 in Alg. 2). Only BB-MO-CBS is able to leverage *NonDomJointPath* during the search due to its new high-level search.
- *Incurring negligible computational overhead per node*: In each iteration, MO-CBS resolves a conflict by updating a single joint path, while BB-MO-CBS resolves a conflict by updating multiple joint paths. During this update, BB-MO-CBS incurs a computational overhead over MO-CBS due to procedure *NonDomJointPath* (which is not used in MO-CBS). However, with the help of Kung's method, this overhead is small, as our experiments show.

Relationship of BB-MO-CBS and CBS We consider that BB-MO-CBS is "more similar" to CBS than to BB-MO-CBS in the following sense.

- Both CBS and BB-MO-CBS build a single constraint tree with the branching factor of two.
- While each node in BB-MO-CBS contains multiple joint paths (instead of one), BB-MO-CBS uses the first joint path as the representative joint path of the node. If we consider the representative joint path in BB-MO-CBS as the counterpart of the joint path in CBS, then the high-level searches of CBS and BB-MO-CBS are analogous (e.g., with respect to how the costs of nodes are computed or how conflicts are resolved), except that BB-MO-CBS exhausts OPEN before termination while CBS terminates when it finds the first solution.

5 Analysis

We prove that BB-MO-CBS computes the Pareto-optimal frontier $\mathcal{C}_*$. A joint path is *consistent* with a set of constraints Ω if the joint path satisfies every constraint in Ω.

[4] Intuitively, if dominated joint paths were pruned in MO-CBS, it means the corresponding individual paths in these nodes are discarded. When MO-CBS later generates a child node and replan the paths for an agent, MO-CBS may lose solutions that consist of the pruned paths of the other agents and the new path of this agent. This makes the MO-CBS incomplete.

Definition 3 (CVN set) *Given an arbitrary set of cost vectors $\mathcal{C}$ and a node P with constraint set Ω, let CVN$(P, \mathcal{C})$ be the set of all joint paths π that (i) are consistent with Ω, (ii) are conflict-free (i.e., valid), and (iii) have cost vectors $\vec{g}(\pi)$ that are not weakly dominated by any cost vector in $\mathcal{C}$.*

Correspondingly, a node P *permits* a joint path π with respect to $\mathcal{C}$ iff $\pi \in$ *CVN*$(P, \mathcal{C})$. For the rest of this section, let $P_k = (\Pi_k, \pi_{k,rep}, \vec{g}_{k,rep}, \Omega_k, \{\Pi_k^i\})$ denote a node that is popped from OPEN for processing in the k-th iteration in Alg. 2 and $\mathcal{C}_k$ denote the set of solution cost vectors computed till the beginning of the k-th iteration of the search.

Lemma 1 *Every time when Alg. 2 reaches Line 12, for every joint path $\pi' \in$ CVN$(P_k, \mathcal{C}_k)$, there exists a joint path $\pi_k \in \Pi_k$ such that $\vec{g}(\pi_k) \preceq \vec{g}(\pi')$.*

Proof 1 *In P_k, Π_k^i for every agent $i \in I$ is computed by the low-level search and is thus guaranteed to be a maximal subset of all cost-unique Pareto-optimal paths that are consistent with Ω_k. Let $\Pi' = ND(\Pi_k^1 \times \Pi_k^2 \times \cdots \times \Pi_k^N)$. Then, for any joint path π that is consistent with Ω_k, $\exists \pi' \in \Pi', \vec{g}(\pi') \preceq \vec{g}(\pi)$ (Claim 1). Π_k in P_k is initialized to Π' when P_k is generated on Line 26. A joint path in Π_k is then deleted only if (i) it is added to $\mathcal{C}_k$ on Lines 14 and 15 or (ii) its cost is weakly dominated by a cost vector in $\mathcal{C}_k$ (Line 6). Therefore, given a joint path $\pi' \in \Pi'$, either $\pi' \in \Pi_k$ or $\exists \vec{c} \in \mathcal{C}_k, \vec{c} \preceq \vec{g}(\pi')$. Combined with Claim 1, we know that, given a joint path π that is consistent with Ω_k, either $\exists \pi_k \in \Pi_k, \vec{g}(\pi_k) \preceq \vec{g}(\pi)$ or $\exists \vec{c} \in \mathcal{C}_k, \vec{c} \preceq \vec{g}(\pi)$. Since, by definition, any joint path $\pi' \in$ CVN$(P_k, \mathcal{C}_k)$ is consistent with Ω_k and $\nexists \vec{c} \in \mathcal{C}_k, \vec{c} \preceq \vec{g}(\pi')$, we know that $\exists \pi_k \in \Pi_k, \vec{g}(\pi_k) \preceq \vec{g}(\pi')$.*

Lemma 2 *At the beginning of a search iteration, for every Pareto-optimal solution π_* with $\vec{g}(\pi_*) \in \mathcal{C}_* \backslash \mathcal{C}_k$, there exists a node P in OPEN that permits π_* with respect to $\mathcal{C}_k$.*

Proof 2 *We prove this lemma by induction. After the initialization, OPEN contains only the root node P_o, which has an empty constraint set Ω_o. P_o thus permits any conflict-free joint path with respect to $\mathcal{C}_o$ since $\mathcal{C}_o$ and Ω_o are empty. At the beginning of the k-th iteration, assume that there is a node P' in OPEN that permits π_* with respect to $\mathcal{C}_k$. If P' is not popped from OPEN (i.e., the popped node P_k does not permit π_* with respect to $\mathcal{C}_k$), then P' is still in OPEN after the iteration and still permits π_* with respect to the new $\mathcal{C}_k$ (i.e., $\mathcal{C}_{k+1}$) after the iteration. Therefore the lemma holds. Otherwise, there are three cases. First, some joint paths in Π_k are filtered (Lines 6-11). Because of Lemma 1 and because $\pi_* \in$ CVN$(P_k, \mathcal{C}_k)$, Π_k cannot be empty after the filtering. P_k is thus re-inserted into OPEN, and the lemma holds, Second, $\pi_{k,rep}$ is conflict-free (Lines 13-19). If $\pi_{k,rep} = \pi_*$, then $\vec{g}(\pi_*)$ is added to $\mathcal{C}_k$ (i.e., $\vec{g}(\pi_*) \notin \mathcal{C}_* \backslash \mathcal{C}_k$) and the lemma holds. If $\pi_{k,rep} \neq \pi_*$, then P_k is re-inserted into OPEN (since Π_k contains at least π_* and cannot be empty) and the lemma holds. Third, Alg. 2 branches on P_k to resolve a conflict (Lines 12 and 20-29). Two constraints ω^i and ω^j and two new corresponding nodes P_{l^i} and P_{l^j} are created. π_* cannot violate both ω^i and ω^j (because, otherwise, π_* would not be conflict-free). Thus, π_* does not violate at least one of the two constraints (say ω^i), and the cor-*

responding node P_{l^i} permits π_*. Both nodes are then added to OPEN, and the lemma holds.

Lemma 3 *Every time when Alg. 2 reaches (and before executing) Line 14, $\vec{g}_{k,rep}$ cannot be weakly dominated by any $\vec{g} \in \mathcal{C}_k$ (i.e., $\nexists \vec{g} \in \mathcal{C}_k$, $\vec{g} \preceq \vec{g}_{k,rep}$).*

Proof 3 *This lemma holds because of Lines 6-11 in Alg. 2. In other words, if $\vec{g}_{k,rep}$ is weakly dominated by some $\vec{g} \in \mathcal{C}_k$, $\vec{g}_{k,rep}$ is discarded before reaching Line 14.*

Lemma 4 *During the search, $\mathcal{C}_k \subseteq \mathcal{C}_*$.*

Proof 4 *Initially, $\mathcal{C}_o$ is empty and $\mathcal{C}_o \subseteq \mathcal{C}_*$. Assume that, in the k-th iteration, $\mathcal{C}_k \subseteq \mathcal{C}_*$ and a representative vector $\vec{g}_{k,rep}$ of node P_k is added to $\mathcal{C}_k$. Since Π_k is a subset of $ND(\Pi_k^1 \times \Pi_k^2 \times \cdots \times \Pi_k^N)$, we know that $\nexists \pi_k \in \Pi_k, \vec{g}(\pi_k) \prec \vec{g}_{k,rep}$. Additionally, since $\vec{g}_{k,rep}$ is the lexicographic minimum among the representative vector of all nodes in OPEN and the representative vector of a node is the lexicographic minimum of the cost vectors of all joint paths in that node, $\vec{g}_{k,rep}$ cannot be dominated by the cost vector of any joint path in any other nodes in OPEN (Claim 2). Assume $\vec{g}_{k,rep}$ is not Pareto-optimal, we know that $\exists \vec{g}_* \in \mathcal{C}_*$, $\vec{g}_* \prec \vec{g}_{k,rep}$. Because of Lemma 3, we know that $\vec{g}_* \notin \mathcal{C}_k$, and therefore $\vec{g}_* \in \mathcal{C}_* \backslash \mathcal{C}_k$. Because of Lemmata 1 and 2, there exists a joint path π_l in some node P_l in OPEN with $\vec{g}(\pi_l) \preceq \vec{g}_* \prec \vec{g}_{k,rep}$, which contradicts Claim 2. Thus, $\vec{g}_{k,rep}$ is Pareto-optimal. After $\vec{g}_{k,rep}$ is added to $\mathcal{C}_k$, let $\mathcal{C}_{k+1}$ denote the new set. Then, $\mathcal{C}_{k+1} \subseteq \mathcal{C}_*$.*

Theorem 1 *For a feasible instance (i.e., an instance with at least one solution), $\mathcal{C}_k$ is the Pareto-optimal frontier when BB-MO-CBS terminates and every $\vec{g} \in \mathcal{C}_k$ is unique.*

Proof 5 *From Lemma 4, we know that $\mathcal{C}_k \subseteq \mathcal{C}_*$ when BB-MO-CBS terminates. From Lemma 2, we know that $\mathcal{C}_* \backslash \mathcal{C}_k = \emptyset$ when OPEN is empty, i.e., when BB-MO-CBS terminates. Hence, $\mathcal{C}_k = \mathcal{C}_*$ when BB-MO-CBS terminates. Because of Lemma 3, every $\vec{g} \in \mathcal{C}_k$ is unique.*

Theorem 2 *For a feasible instance, BB-MO-CBS terminates in finite time.*

Proof 6 *The proof of Lemma 4 in (Ren, Rathinam, and Choset 2022) applies to BB-MO-CBS.*

6 Experimental Results

We run experiments on a Ubuntu 20.04 laptop with an Intel Core i7-11800H 2.40GHz CPU and 16GB RAM without multi-threading or compiler optimization flags. We implement our BB-MO-CBS in C++[5] and compare it against the online C++ implementation of MO-CBS-t in (Ren, Rathinam, and Choset 2022). We implement BB-MO-CBS based on the MO-CBS-t implementation and keep them as similar as possible. For both BB-MO-CBS and MO-CBS-t, we implement the low-level search with BOA* (Hernández et al. 2023) for bi-objective instances and EMOA* (Ren et al. 2022) for instances with more than two objectives. We select (grid) maps from different categories (e.g., room-like, maze-like, etc.) from the MAPF benchmark set (Stern et al. 2019). For each map, 25 instances provide initial location-destination pairs for all agents. For each map, we generate a four-neighbor undirected graph G. We create cost vectors for its edges by sampling each element of a cost vector randomly from $\{1, 2\}$, similar to (Pulido, Mandow, and Pérez-de-la Cruz 2015; Ren, Rathinam, and Choset 2022). We run experiments with two and three objectives and use a runtime limit of 300 seconds for each instance.

[5]https://github.com/wonderren/public_bbmocbs

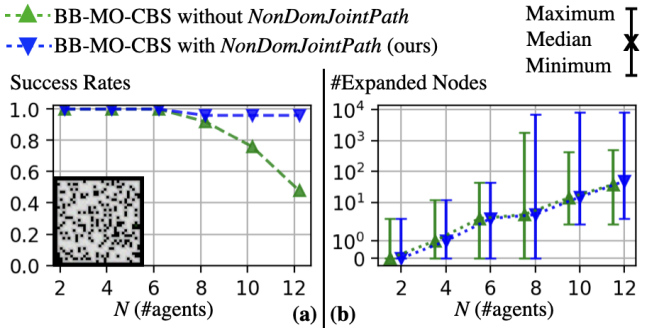

Figure 3: Comparison of BB-MO-CBS with and without *NonDomJointPath*. It shows the efficiency (i.e., higher success rates) of considering only the undominated joint paths during the search in BB-MO-CBS.

6.1 Computing Undominated Subset

We compared the two implementations of *NonDomJointPath*: using Kung's method (Kung, Luccio, and Preparata 1975) and using a naive method (denoted as Naive) as discussed in Sec. 4.2, with two objectives. We run instances with up to $N = 16$ agents on the Random 32x32 and den312d 65x81 maps. We observe that, Naive can take up to about one second per call on average for some instance, while Kung can reduce this number to about 0.05 seconds.

6.2 BB-MO-CBS With and Without Undominated Subset

We compared BB-MO-CBS with and without *NonDomJointPath* on the Random 32x32 map. Specifically, "without *NonDomJointPath*" means that on Line 26 in Alg. 2, all combination of Π_l^i over all agents $i \in I$ are considered without discarding the dominated joint paths. First, this baseline runs out of the 16GB RAM for some instances when $N = 14$ as it has to create joint paths corresponding to all combinations of individual paths. Second, as shown in Fig. 3, BB-MO-CBS achieves higher success rates than this baseline and manages to expand more nodes before the runtime limit is reached.

6.3 BB-MO-CBS versus MO-CBS-t: Success Rates

We compared the success rates of our BB-MO-CBS and the existing MO-CBS-t in four maps of different types and sizes with two objectives. As shown in Fig. 4, BB-MO-CBS achieves higher success rates than MO-CBS-t in all experimental settings. BB-MO-CBS often doubles or triples the

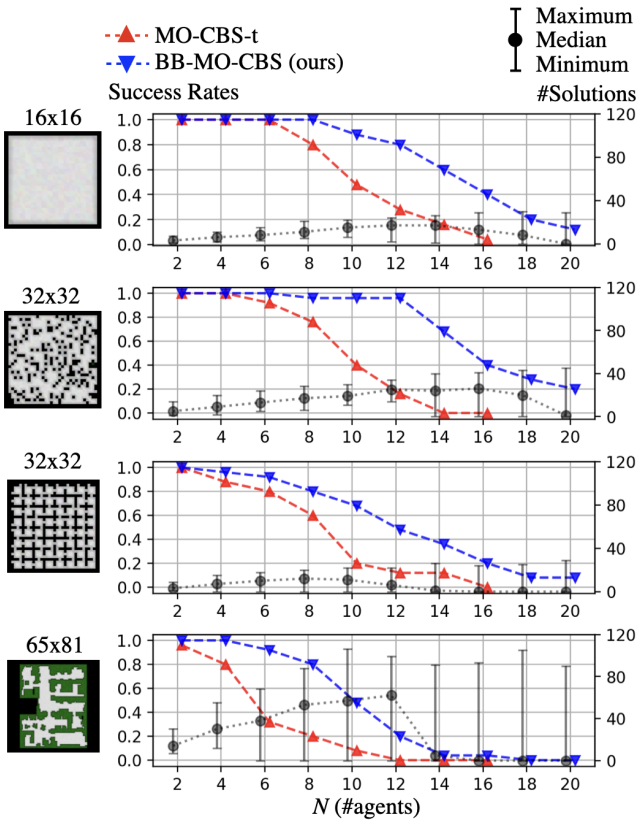

Figure 4: Success rates for the existing MO-CBS-t and our BB-MO-CBS on four maps with two objectives. The left vertical axis shows the success rate. The right vertical axis shows the average number of Pareto-optimal solutions found by BB-MO-CBS within the runtime limit.

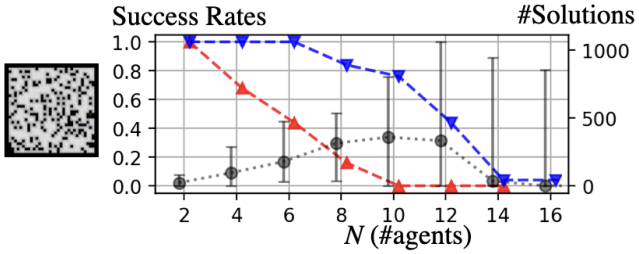

Figure 5: Success rates of the existing MO-CBS-t and our BB-MO-CBS on the Random 32x32 map with three objectives. The labels are the same as in Fig. 4.

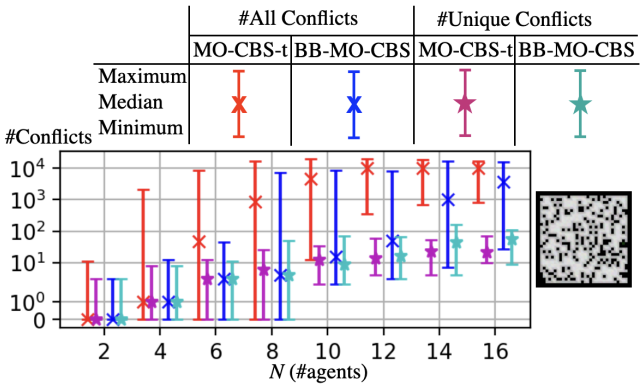

Figure 6: Numbers of conflicts resolved by MO-CBS-t and BB-MO-CBS on the Random 32x32 map. The bar-with-cross (and the bar-with-star) markers show the maximum/median/minimum (over 25 instances) of the total number of conflicts per instance (and the number of unique conflicts per instance) resolved by the algorithms.

success rates of the baseline. Fig. 4 also shows the maximum/median/minimum numbers of solutions of each instance computed by BB-MO-CBS within the runtime limit. Even if BB-MO-CBS does not terminate within the runtime limit, the computed solutions are still Pareto-optimal. We also compare BB-MO-CBS and MO-CBS-t on the Random 32x32 map with three objectives. Fig. 5 shows that BB-MO-CBS again achieves higher success rates than MO-CBS-t.

6.4 BB-MO-CBS versus MO-CBS-t: Resolved Conflicts

We compared the numbers of resolved conflicts of BB-MO-CBS and MO-CBS-t on the Random 32x32 map with two objectives. Since a conflict (e.g. (i, j, v, t)) can be resolved many times during a search, we show the maximum/median/minimum (over 25 instances for each N) of both the numbers of all conflicts per instance and the numbers of unique conflicts per instance resolved by both algorithms in Fig. 6. As Fig. 6 shows, BB-MO-CBS resolves up to about two orders of magnitude fewer conflicts than MO-CBS-t. For example, for $N = 6$, BB-MO-CBS and MO-CBS-t achieve similar success rates, but BB-MO-CBS resolves at most 100 conflicts while MO-CBS-t resolves at most 10,000 conflicts. As Fig. 6 also shows, BB-MO-CBS and MO-CBS-t resolve about the same numbers of unique conflicts for $N = 6$. Thus, the efficiency gain of BB-MO-CBS is due to it not resolving the same conflicts repeatedly.

7 Conclusions and Future Work

This paper developed a new algorithm BB-MO-CBS to find the Pareto-optimal frontier for the MA-MO-PF problems. Different from MO-CBS, BB-MO-CBS (i) uses a new high-level search to update multiple joint paths intelligently when a conflict is resolved and is able to reduce the branching factor to the small constant of two; and (ii) is able to discard dominated joint paths during the search and thus avoids the duplicated conflict resolution effort for the discarded joint paths. Consequently, BB-MO-CBS runs significantly faster than MO-CBS. We verify the advantage of BB-MO-CBS over MO-CBS experimentally.

It is future work to leverage the conflict resolution techniques from CBS, such as (Li et al. 2021; Zhang et al. 2022), or improve the low-level search, to expedite the search process of BB-MO-CBS further.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 2120219, 2120529 and 2121028. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Ehrgott, M. 2005. *Multicriteria optimization*, volume 491. Springer Science & Business Media.

Hernández, C.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; Koenig, S.; and Salzman, O. 2023. Simple and efficient bi-objective search algorithms via fast dominance checks. *Artificial Intelligence*, 314: 103807.

Kung, H.-T.; Luccio, F.; and Preparata, F. P. 1975. On finding the maxima of a set of vectors. *Journal of the ACM*, 22(4): 469–476.

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

Morris, R.; Pasareanu, C. S.; Luckow, K.; Malik, W.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2016. Planning, scheduling and monitoring for airport surface operations. In *AAAI Workshop: Planning for Hybrid Systems*, 608–614.

Pulido, F.-J.; Mandow, L.; and Pérez-de-la Cruz, J.-L. 2015. Dimensionality reduction in multiobjective shortest path search. *Computers & Operations Research*, 64: 60–70.

Ren, Z.; Rathinam, S.; and Choset, H. 2021. Subdimensional expansion for multi-objective multi-agent path finding. *IEEE Robotics and Automation Letters*, 6(4): 7153–7160.

Ren, Z.; Rathinam, S.; and Choset, H. 2022. A conflict-based search framework for multiobjective multiagent path finding. *IEEE Transactions on Automation Science and Engineering*, 1–13.

Ren, Z.; Zhan, R.; Rathinam, S.; Likhachev, M.; and Choset, H. 2022. Enhanced multi-objective A* using balanced binary search trees. In *Proceedings of International Symposium on Combinatorial Search*, volume 15, 162–170.

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

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

Weise, J.; Mai, S.; Zille, H.; and Mostaghim, S. 2020. On the scalable multi-objective multi-agent pathfinding problem. In *Proceedings of IEEE Congress on Evolutionary Computation*, 1–8.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1): 9.

Yu, J.; and LaValle, S. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, 1443–1449.

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