Efficient Approximate Search for Multi-Objective Multi-Agent Path Finding

Fangji Wang[1*], Han Zhang[2*], Sven Koenig[2], Jiaoyang Li[3]

[1] Tsinghua University
[2] University of Southern California
[3] Carnegie Mellon University
wang-fj20@mails.tsinghua.edu.cn, zhan645@usc.edu, skoenig@usc.edu, jiaoyangli@cmu.edu

Abstract

The Multi-Objective Multi-Agent Path Finding (MO-MAPF) problem is the problem of computing collision-free paths for a team of agents while minimizing multiple cost metrics. Most existing MO-MAPF algorithms aim to compute the Pareto frontier. However, the Pareto frontier can be time-consuming to compute. Our first main contribution is BB-MO-CBS-pex, an approximate MO-MAPF algorithm that computes an approximate frontier for a user-specific approximation factor. BB-MO-CBS-pex builds upon BB-MO-CBS, a state-of-the-art MO-MAPF algorithm, and leverages A*pex, a state-of-the-art single-agent multi-objective search algorithm, to speed up different parts of BB-MO-CBS. We also provide two speed-up techniques for BB-MO-CBS-pex. Our second main contribution is BB-MO-CBS-k, which builds upon BB-MO-CBS-pex and computes up to k solutions for a user-provided k-value. BB-MO-CBS-k is useful when it is unclear how to determine an appropriate approximation factor. Our experimental results show that both BB-MO-CBS-pex and BB-MO-CBS-k solved significantly more instances than BB-MO-CBS for different approximation factors and k-values, respectively. Additionally, we compare BB-MO-CBS-pex with an approximate baseline algorithm derived from BB-MO-CBS and show that BB-MO-CBS-pex achieved speed-ups up to two orders of magnitude.

Introduction

The Multi-Agent Path Finding (MAPF) problem is the problem of finding a set of collision-free paths for a team of agents. It is related to many real-world applications (Wurman, D'Andrea, and Mountz 2008; Morris et al. 2016). A *solution* is a set of collision-free paths for all agents. Computing a minimum-cost solution for the MAPF problem is known to be NP-hard (Yu and LaValle 2013; Ma et al. 2016). In this paper, we study a variant of the MAPF problem called the Multi-Objective MAPF (MO-MAPF) problem (Ren, Rathinam, and Choset 2022), which considers multiple cost metrics. Many real-world applications of MAPF can be viewed as multi-objective problems. For example, in multi-robot systems, some interesting cost metrics include travel distance, energy consumption, and risk.

*Fangji Wang and Han Zhang contributed equally to this work.

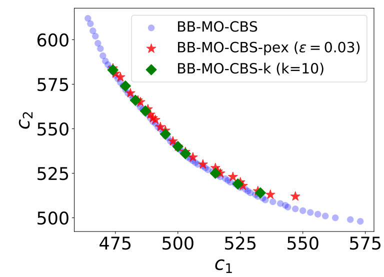


Figure 1: Costs of the solutions computed by different algorithms for an MO-MAPF instance with two objectives and 8 agents, where BB-MO-CBS-pex and BB-MO-CBS-k, our proposed algorithms, achieved speed-ups of $25\times$ and $44\times$ over BB-MO-CBS, respectively.

Most existing MO-MAPF algorithms, such as MO-M* (Ren, Rathinam, and Choset 2021), MO-CBS (Ren, Rathinam, and Choset 2022), and BB-MO-CBS (Ren et al. 2023), aim to compute the *Pareto frontier*, that is, a set of solutions where each solution is not dominated by any other solutions. A solution P *dominates* another solution P' if the cost of P is no larger than the cost of P' for every cost metric and the cost for at least one cost metric is smaller. Unfortunately, even in the multi-objective single-agent case, the size of the Pareto frontier can be exponential in the size of the graph being searched (Ehrgott 2005; Breugem, Dollevoet, and van den Heuvel 2017). Therefore, computing Pareto frontiers for MO-MAPF can be time-consuming. Existing works on multi-objective single-agent search have proposed to compute approximate frontiers (Perny and Spanjaard 2008; Goldin and Salzman 2021; Zhang et al. 2022) instead, which significantly speeds up the search. However, this has yet to be investigated for MO-MAPF.

Our first main contribution is *BB-MO-CBS-pex*, an approximate MO-MAPF algorithm that computes an approximate frontier for the user-specific approximation factor. BB-MO-CBS-pex builds upon BB-MO-CBS, a state-of-the-art MO-MAPF algorithm that consists of a low-level search to plan paths for each agent and a high-level search to resolve collisions. BB-MO-CBS-pex leverages A*pex (Zhang et al. 2022), a state-of-the-art multi-objective single-agent approximate search algorithm, as the low-level search algorithm and also applies the algorithmic idea behind A*pex to speed up the high-level search. In addition, we provide two techniques to further speed up BB-MO-CBS-pex.

In practice, a too large approximation factor can cause BB-MO-CBS-pex to return only one solution, offering no trade-off to users, while a too small one provides no chance for BB-MO-CBS-pex to speed up. Therefore, specifying a good approximation factor is important yet non-trivial, and one might prefer to specify the desired number of solutions instead. To this end, our second main contribution is *BB-MO-CBS-k*, which builds upon BB-MO-CBS-pex and computes a set of up to k solutions for a user-provided k-value.

In our experimental study, we compare BB-MO-CBS-pex and BB-MO-CBS-k with BB-MO-CBS. Our results show that BB-MO-CBS-pex and BB-MO-CBS-k solved significantly more problem instances than BB-MO-CBS within the given runtime limit of 120 seconds for different approximation factors and k-values, respectively. Additionally, we compare BB-MO-CBS-pex with BB-MO-CBS-ε, an approximate baseline algorithm derived from BB-MO-CBS. Our results show that BB-MO-CBS-pex solved significantly more instances and achieved up to two orders of magnitude speed-up compared to BB-MO-CBS-ε.

Terminology and Problem Definition

We use **boldface** font to denote vectors or vector functions and v_i to denote the i-th component of vector or vector function $\mathbf{v}$. We define the addition of two M-dimensional vectors $\mathbf{u}$ and $\mathbf{v}$ as $\mathbf{u}+\mathbf{v}=[u_1+v_1, u_2+v_2, \ldots, u_M+v_M]$. We define the *vector minimum* of $\mathbf{u}$ and $\mathbf{v}$ as *vector_min*$(u, v) = [\min(u_1, v_1), \min(u_2, v_2), \ldots, \min(u_M, v_M)]$. We say that $\mathbf{u}$ *weakly dominates* $\mathbf{v}$, denoted as $\mathbf{u} \preceq \mathbf{v}$, if $u_i \leq v_i$, $i = 1, 2, \ldots, M$. We say that $\mathbf{u}$ *dominates* $\mathbf{v}$, denoted as $\mathbf{u} \prec \mathbf{v}$, if $\mathbf{u} \preceq \mathbf{v}$ and $\exists i \in \{1, 2, \ldots, M\}, u_i < v_i$. We say that $\mathbf{u}$ $\boldsymbol{\varepsilon}$-*dominates* $\mathbf{v}$ for an *approximation factor* (or, more precisely, vector of approximation factors) $\boldsymbol{\varepsilon} = [\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_M]$, denoted as $\mathbf{u} \preceq_{\boldsymbol{\varepsilon}} \mathbf{v}$, if $u_i \leq (1+\varepsilon_i)v_i$, $i = 1, 2, \ldots, M$.

In the MO-MAPF problem, we are given a shared workspace, represented by a finite directed graph $G = \langle V, E \rangle$, and a set of N agents $\{a^1, a^2, \ldots, a^N\}$. V denotes the set of vertices, and each vertex $v \in V$ corresponds to a possible location for agents. $E \subseteq V \times V$ denotes the set of edges, and each edge $e = \langle u, v \rangle \in E$ corresponds to a move action from u to v. Note that an edge from a vertex to itself can also be included in E, which means that agents can wait at the vertex. The *cost* of an edge e is a positive M-dimensional vector denoted as $\mathbf{c}(e) \in \mathbb{R}^M_{>0}$, where M is the *number of objectives*. The agents are indexed by $I = \{1, 2, \ldots, N\}$. In the rest of the paper, we use $|I|$ instead of N to denote the number of agents. We use superscript $^{i \in I}$ to indicate that a variable is related to agent a^i. Each agent a^i has a *start vertex* $v^i_{\text{start}} \in V$ and a *goal vertex* $v^i_{\text{goal}} \in V$.

A *path* $\pi^i = (v^i_1, v^i_2, \ldots, v^i_\ell)$ for agent a^i is a sequence of vertices with $v^i_1 = v^i_{\text{start}}$, $v^i_\ell = v^i_{\text{goal}}$, and $\langle v^i_j, v^i_{j+1} \rangle \in E, j = 1, 2 \ldots \ell - 1$. The *cost* of path π^i is defined as $\mathbf{c}(\pi^i) = \sum_{j=1}^{\ell-1} \mathbf{c}(\langle v^i_j, v^i_{j+1} \rangle)$. A path also corresponds to a sequence of move and wait actions. Agents stay at their goal vertices forever after they execute their last actions.

For a subset of agent indices $I' \subseteq I$, a *joint path* $P = \{\pi^i : i \in I'\}$ is a set of paths, one for each agent whose index is in I'. Throughout this paper, we assume that $I' = I$, unless mentioned otherwise. The *cost* of joint path P is defined as $\mathbf{c}(P) = \sum_{i \in I'} \mathbf{c}(\pi^i)$. We consider two types of conflicts: A *vertex conflict* happens when two agents stay at the same vertex simultaneously, and an *edge conflict* happens when two agents switch their vertices simultaneously. A *solution* is a conflict-free joint path for index set I.

In this paper, we use symbol P to denote a joint path, which is a set of paths for different agents, and symbol Π to denote a set of paths for the same agent. Additionally, we use symbol $\mathbb{P}$ to denote a set of joint paths.

We say that a path π weakly dominates another path π' (resp. π $\boldsymbol{\varepsilon}$-dominates π') if $\mathbf{c}(\pi) \preceq \mathbf{c}(\pi')$ (resp. $\mathbf{c}(\pi) \preceq_{\boldsymbol{\varepsilon}} \mathbf{c}(\pi')$). A set of paths Π is *undominated* if its paths do not weakly dominate each other. A *Pareto frontier* of Π is defined as an undominated subset of Π such that each path in Π is weakly dominated by at least one path in the Pareto frontier. An $\boldsymbol{\varepsilon}$-*approximate frontier* of Π is defined as an undominated subset of Π such that each path in Π is $\boldsymbol{\varepsilon}$-dominated by at least one path in the $\boldsymbol{\varepsilon}$-approximate frontier.

For joint paths, we define weakly dominance, $\boldsymbol{\varepsilon}$-dominance, undominated sets, Pareto frontiers, and $\boldsymbol{\varepsilon}$-approximate frontiers in the same way that we do for paths. Unless mentioned otherwise, we use a Pareto frontier (resp. an $\boldsymbol{\varepsilon}$-approximate frontier) to refer to a Pareto frontier (resp. an $\boldsymbol{\varepsilon}$-approximate frontier) of all solutions for the MO-MAPF problem instance we consider.

Algorithm Background

This section reviews CBS (Sharon et al. 2015), BB-MO-CBS (Ren et al. 2023), and A*pex (Zhang et al. 2022).

CBS

CBS (Sharon et al. 2015) is a complete and optimal single-objective MAPF algorithm. It consists of two levels. On the high level, CBS performs a best-first search on a *Constraint Tree* (CT). Each CT node contains a set of constraints and a joint path, whose path for each agent satisfies all these constraints and has the minimum path cost when ignoring conflicts. A *constraint* ω^i has the form $\langle i, v, t \rangle$ or $\langle i, e, t \rangle$, where $i \in I, v \in V, e \in E$, and $t \in \mathbb{N}_{>0}$. For the first case, any path $\pi^i = (v^i_1, v^i_2, \ldots, v^i_l)$ for a^i is prohibited from $v^i_t = v$; for the second case, any path $\pi^i = (v^i_1, v^i_2, \ldots, v^i_l)$ for a^i is prohibited from $\langle v^i_t, v^i_{t+1} \rangle = e$. The *g-value* of a CT node is defined as the cost of its joint path. CBS maintains an $Open$ list for all generated but not yet expanded nodes and initializes $Open$ with the root CT node, which has an empty set of constraints and a path for each agent that has the minimum path cost when ignoring conflicts. In each iteration, CBS extracts a CT node with the minimum g-value from $Open$ and returns its joint path as the solution if the joint path is conflict-free. Otherwise, CBS picks a conflict of the joint path to resolve, splits the CT node into two child CT nodes, and adds a constraint to each child CT node to prohibit either one or the other of the two conflicting agents from using the conflicting vertex or edge at the conflicting timestep. CBS then calls its low level to replan the path of the newly constrained agent in each child CT node. The low-level planner finds a path with the minimum path cost while satisfying all constraints of the child CT node but ignoring conflicts.

BB-MO-CBS

BB-MO-CBS (Ren et al. 2023) generalizes CBS from single-objective MAPF to MO-MAPF and computes a Pareto frontier for the given MO-MAPF problem instance.

Algorithm 1 shows the pseudocode for BB-MO-CBS. BB-MO-CBS maintains an $Open$ list for all generated but not yet expanded nodes and a solution set $\mathcal{S}$ for the solutions it has found. Similar to CBS, BB-MO-CBS also consists of two levels. On the high level, BB-MO-CBS also maintains a CT. A major difference between CBS and BB-MO-CBS is that, while a CT node of CBS corresponds to one joint path, a CT node of BB-MO-CBS corresponds to a set of joint paths that are different combinations of paths for each agent. This design allows BB-MO-CBS to resolve the same conflict in different joint paths simultaneously. More specifically, in BB-MO-CBS, we redefine a CT node as a tuple $n = \langle \Omega, \{\Pi^i \mid i \in I\}, \mathbb{P}\rangle$, which contains (1) a set of constraints Ω, where a constraint has the same form as the constraints in CBS, (2) a Pareto frontier of paths Π^i for each agent a^i that satisfy the constraints in Ω, and (3) a set of joint paths $\mathbb{P} \subseteq PF(\Pi^1 \times \Pi^2 \times \ldots \times \Pi^{|I|})$, where $PF(\Pi^1 \times \Pi^2 \times \ldots \times \Pi^{|I|})$ denotes a Pareto frontier of all joint paths that consist of a path from Π^i for each agent a^i. As we will show later, BB-MO-CBS repeatedly updates $\mathbb{P}$ to the subset of $PF(\Pi^1 \times \Pi^2 \times \ldots \times \Pi^{|I|})$ that are not weakly dominated by any solution in $\mathcal{S}$. The *current joint path* of CT node n, denoted as $\mathbb{P}.lexFirst$, is defined as the joint path with the lexicographically smallest cost in $\mathbb{P}$. The g-value of CT node n is defined as $\mathbf{c}(\mathbb{P}.lexFirst)$.

During the initialization, BB-MO-CBS first computes a Pareto frontier of paths Π^i_o, ignoring conflicts, for each agent a^i, and a Pareto frontier of joint paths $\mathbb{P}_o = PF(\Pi^1_o \times \Pi^2_o \times \ldots \times \Pi^{|I|}_o)$ (Lines 1-4). It then initializes $Open$ with the root CT node $n_o = \langle \emptyset, \{\Pi^i_o \mid i \in I\}, \mathbb{P}_o\rangle$ (Line 5).

In each iteration, BB-MO-CBS extracts a CT node $n = \langle \Omega, \{\Pi^i \mid i \in I\}, \mathbb{P}\rangle$ with the lexicographically smallest g-value (Line 7). The current joint path of n, namely $\mathbb{P}.lexFirst$, must have the lexicographically smallest cost among (and hence is not dominated by) the joint paths of all CT nodes in $Open$. BB-MO-CBS first computes $\mathbb{P}'$ by removing the joint paths weakly dominated by any solution in $\mathcal{S}$ from $\mathbb{P}$ (Line 8). If $\mathbb{P}'$ is empty, BB-MO-CBS discards n and ends the iteration (Line 9). If the current joint path changes, (that is, $\mathbb{P}'.lexFirst \neq \mathbb{P}.lexFirst$), BB-MO-CBS reinserts n with the updated joint path set $\mathbb{P}'$ to $Open$ and ends this iteration (Lines 10-12). If the current joint path does not change and is conflict-free, BB-MO-CBS adds it to $\mathcal{S}$. Different from CBS, BB-MO-CBS does not terminate in this case. It removes the new solution $\mathbb{P}'.lexFirst$ from $\mathbb{P}'$ and reinserts a CT node with the updated joint path set $\mathbb{P}'$ to $Open$ if $\mathbb{P}'$ is still not empty (Lines 14-19). BB-MO-CBS does this because the remaining joint paths in $\mathbb{P}'$ still have the potential to lead to new solutions. If the current joint path is not conflict-free, similar to CBS, BB-MO-CBS picks a conflict of the joint path to resolve, splits the CT node into two child CT nodes, and adds a constraint to each child CT node (Lines 20-23). BB-MO-CBS then calls its low level to replan a Pareto frontier of paths for the newly constrained agent in each child CT node that satisfy all constraints of the child CT node and insert the child nodes to $Open$ (Lines 24-26). The low level of BB-MO-CBS can be implemented with any multi-objective single-agent search algorithm that computes a Pareto frontier, such as BOA* (Hernández et al. 2023) and EMOA* (Ren et al. 2022).

Algorithm 1 BB-MO-CBS

1: $\mathcal{S} \leftarrow \emptyset$; $Open \leftarrow \emptyset$
2: **for all** $i \in I$ **do**
3: $\quad \Pi^i_o \leftarrow LowLevelSearch(i, \emptyset)$
4: $\mathbb{P}_o \leftarrow PF(\Pi^1_o \times \Pi^2_o \times \cdots \times \Pi^N_o)$
5: add $\langle \emptyset, \{\Pi^i_o \mid i \in I\}, \mathbb{P}_o\rangle$ to $Open$
6: **while** $Open \neq \emptyset$ **do**
7: $\quad n = \langle \Omega, \{\Pi^i \mid i \in I\}, \mathbb{P}\rangle \leftarrow Open.extract\_min()$
8: $\quad \mathbb{P}' \leftarrow \{P \in \mathbb{P} \mid \nexists P' \in \mathcal{S}, \mathbf{c}(P') \preceq \mathbf{c}(P)\}$
9: $\quad$ **if** $\mathbb{P}' = \emptyset$ **then continue**
10: $\quad$ **if** $\mathbb{P}'.lexFirst \neq \mathbb{P}.lexFirst$ **then**
11: $\quad\quad$ add $\langle \Omega, \{\Pi^i \mid i \in I\}, \mathbb{P}'\rangle$ to $Open$
12: $\quad\quad$ **continue**
13: $\quad$ *cft* $\leftarrow DetectConflict(\mathbb{P}'.lexFirst)$
14: $\quad$ **if** *cft* does not exist **then**
15: $\quad\quad$ add $\mathbb{P}'.lexFirst$ to $\mathcal{S}$
16: $\quad\quad$ remove $\mathbb{P}'.lexFirst$ from $\mathbb{P}'$
17: $\quad\quad$ **if** $\mathbb{P}' \neq \emptyset$ **then**
18: $\quad\quad\quad$ add $\langle \Omega, \{\Pi^i \mid i \in I\}, \mathbb{P}'\rangle$ to $Open$
19: $\quad\quad$ **continue**
20: $\quad \{\omega^i, \omega^j\} \leftarrow GenerateConstraints(cft)$
21: $\quad$ **for all** $i' \in \{i, j\}$ **do**
22: $\quad\quad \{\Pi^i_{new} \mid i \in I\} \leftarrow \{\Pi^i \mid i \in I\}$
23: $\quad\quad \Omega_{new} \leftarrow \Omega \cup \{\omega^{i'}\}$
24: $\quad\quad \Pi^{i'}_{new} \leftarrow LowLevelSearch(i', \Omega_{new})$
25: $\quad\quad \mathbb{P}_{new} \leftarrow PF(\Pi^1_{new} \times \Pi^2_{new} \times \cdots \times \Pi^N_{new})$
26: $\quad\quad$ add $\langle \Omega_{new}, \{\Pi^i_{new} \mid i \in I\}, \mathbb{P}_{new}\rangle$ to $Open$
27: **return** $\mathcal{S}$

BB-MO-CBS terminates and returns $\mathcal{S}$ when $Open$ is empty. Ren et al. (2023) showed that $\mathcal{S}$ is a Pareto frontier for the given MO-MAPF problem instance.

A straightforward approach to introduce approximation to BB-MO-CBS is to prune joint paths that are ε-dominated by any solution in $\mathcal{S}$. We propose BB-MO-CBS-ε, an approximate variant of BB-MO-CBS that we will use as a baseline. It changes only Line 8 of BB-MO-CBS: When computing the updated joint path set $\mathbb{P}'$, it removes all the joint paths in $\mathbb{P}$ that are ε-dominated by any solution in $\mathcal{S}$. It is easy to show that, given an MO-MAPF problem instance and an ε-value, BB-MO-CBS-ε computes an ε-approximate frontier of the solutions.

A*pex

A*pex (Zhang et al. 2022) is a multi-objective (single-agent) search algorithm that computes an ε-approximate frontier of paths from a given start vertex v_{start} to a given goal vertex v_{goal} for a user-provided ε-value. In A*pex, a node n corresponds to a vertex v and a set of paths Π from v_{start} to v. Instead of explicitly storing Π, A*pex stores only one path $\pi \in \Pi$, called the *representative path* of n, and a cost vector $\mathbf{A}(n)$, called the *apex* of n. $\mathbf{A}(n)$ is the vector minimum value of the costs of all paths in Π. We say that node n is *ε-bounded* if $\mathbf{c}(\pi) + \mathbf{h}(v) \preceq_{\boldsymbol{\varepsilon}} \mathbf{A}(n) + \mathbf{h}(v)$, where $\mathbf{h}$ is a consistent heuristic function where each component of $\mathbf{h}(v)$ provides a lower bound on the cost of any path from v to the goal vertex v_{goal} for the corresponding objective.

By *merging* nodes at the same vertex on condition that the resulting node is ε-bounded, A*pex reduces the search effort and can quickly compute an ε-approximate frontier. When merging two nodes n and n', the new apex is the vector minimum of $\mathbf{A}(n)$ and $\mathbf{A}(n')$, and the new representative path is either one of the two representative paths of n and n'. Zhang et al. (2022) proposed several approaches for choosing the new representative path. When expanding a node n at v_{goal}, A*pex adds the representative path of n, denoted as π, to the solution set it maintains. Slightly abusing the notation, we use $\mathbf{A}(\pi)$ to denote $\mathbf{A}(n)$, that is, the apex of the node that contains π as the representative path, and call it the *apex* of π. When A*pex terminates, it returns a set of paths, denoted as $\Pi_{\boldsymbol{\varepsilon}}$. In the rest of this paper, we assume that A*pex also outputs the apex of each path in $\Pi_{\boldsymbol{\varepsilon}}$. Let Π_* denote a Pareto frontier from v_{start} to v_{goal}. The apexes of paths in $\Pi_{\boldsymbol{\varepsilon}}$ collectively "lower-bound" Π_*, that is, $\forall \pi_* \in \Pi_* \, \exists \pi \in \Pi_{\boldsymbol{\varepsilon}} \, \mathbf{A}(\pi) \preceq \mathbf{c}(\pi_*)$.

BB-MO-CBS-pex

In this section, we introduce BB-MO-CBS-pex, a variant of BB-MO-CBS that computes an ε-approximate frontier for a given MO-MAPF problem instance and a user-provided ε-value. BB-MO-CBS-pex builds upon BB-MO-CBS-ε with the two major improvements:

1. BB-MO-CBS-pex leverages A*pex to speed up the low-level search.
2. BB-MO-CBS-pex generalizes the merging idea of A*pex to reduce the sizes of joint paths for CT nodes (and hence speed up the high-level search).

Similar to a representative path in A*pex, a joint path P in BB-MO-CBS-pex is a representative for a set of joint paths that are ε-dominated by P. BB-MO-CBS-pex maintains an *apex* $\mathbf{A}(P)$ for P to keep track of this set of joint paths. Similar to an apex in A*pex, $\mathbf{A}(P)$ is the vector minimum of the costs of these joint paths. We say that P is *ε-bounded* if $\mathbf{c}(P) \preceq_{\boldsymbol{\varepsilon}} \mathbf{A}(P)$. When *merging* two joint paths P and P', the resulting joint path P_{new} is equal to either P or P', and the apex of P_{new} is the vector minimum of $\mathbf{A}(P)$ and $\mathbf{A}(P')$. We show the merge function on Lines 1-5 of Algorithm 3. BB-MO-CBS-pex merges two joint paths only when the resulting joint path is ε-bounded.

Similar to BB-MO-CBS, a CT node in BB-MO-CBS-pex is a tuple $n = \langle \Omega, \{\Pi^i | i \in I\}, \mathbb{P} \rangle$ with two differences: (1) Π^i for each agent a^i is an ε-approximate frontier of paths that satisfy constraints in Ω, and (2) $\mathbb{P}$ is a set of joint paths computed by merging joint paths in $PF(\Pi^1 \times \Pi^2 \times \ldots \times \Pi^{|I|})$. The *current joint path* of CT node n, denoted as $\mathbb{P}.lexFirst$, is defined as the joint path with the lexicographically smallest apex in $\mathbb{P}$. The $\mathbf{g}$-value of CT node n is defined as $\mathbf{A}(\mathbb{P}.lexFirst)$.

The changes of BB-MO-CBS-pex over BB-MO-CBS are listed as the following:

1. **Lines 3 and 24.** When initializing the root CT node n_0 and replanning paths for agents, BB-MO-CBS-pex uses A*pex to compute an ε-approximate frontier, instead of a Pareto frontier, of paths for each agent in each CT node.
2. **Lines 4 and 25.** When computing the set of joint paths for a CT node, BB-MO-CBS-pex calls $MergeJointPaths$, which we will explain later, to compute an ε-approximate frontier of joint paths. This reduces the search effort of BB-MO-CBS-pex because fewer joint paths are considered for each CT node.
3. **Line 8.** When computing $\mathbb{P}'$ for an extracted CT node, BB-MO-CBS-pex calls $PruneApproxDom$ (Lines 17-23 of Algorithm 3) to remove a joint path P if $\mathbf{A}(P)$ is ε-dominated by the cost of some solution P_{sol} in $\mathcal{S}$. When removing the joint path, BB-MO-CBS-pex also updates $\mathbf{A}(P_{\text{sol}})$ to the vector minimum of $\mathbf{A}(P)$ and $\mathbf{A}(P_{\text{sol}})$ (Line 21 of Algorithm 3). This update guarantees that, if BB-MO-CBS-pex merges P_{sol} with other solutions later on Line 15, the cost of the resulting solution still ε-dominates $\mathbf{A}(P)$.
4. **Line 15.** When adding a solution $\mathbb{P}.lexFirst$ to $\mathcal{S}$, BB-MO-CBS-pex attempts to merge $\mathbb{P}.lexFirst$ with another solution in $\mathcal{S}$ on condition that the resulting solution is still ε-bounded (Lines 25-26 of Algorithm 3).

Lines 6-16 of Algorithm 3 show the pseudocode for function $MergeJointPaths$, which iteratively computes $\mathbb{P}_i$, $i = 1, 2, \ldots, |I|$, a set of joint paths for the set of agent indices $I' = \{1, 2, \ldots, i\}$. $\mathbb{P}_1$ is initialized with Π^1 (Line 7). To compute $\mathbb{P}_i, i = 2, 3, \ldots, |I|$, BB-MO-CBS-pex iterates over all combinations in $\mathbb{P}_{i-1} \times \Pi_i$, where each combination corresponds to a joint path P for agent indices $\{1, 2, \ldots, i\}$. BB-MO-CBS-pex first checks if $\mathbb{P}_i$ contains a joint path P' that can be merged with P (Line 13) and, if so, replaces P' (in $\mathbb{P}_i$) with the merged joint path (Line 14). Otherwise, P is added to $\mathbb{P}_i$ (Line 15). Eventually, function $MergeJointPaths$ returns $\mathbb{P}_{|I|}$.

We propose two additional improvement techniques.

Conflict-based merging: While Zhang et al. (2022) proposed to choose representative paths based on path costs for A*pex, we propose to choose representative paths (in the low-level search) or joint paths (in $Merge$) based on conflicts. In the low-level search, we use a *Conflict Avoidance Table* (CAT) (Sharon et al. 2015) to store the number of other agents in the current joint path ($\mathbb{P}.lexFirst$) that visit a given vertex or a given edge at a given timestep. Therefore, for each path, we can compute its number of conflicts with other paths using the CAT. In A*pex, when merging two paths, the low-level search chooses the less conflicting path as the representative path on condition that the resulting node is ε-bounded and otherwise chooses the other path. In $Merge$, we also compute the number of conflicts for each

Algorithm 2 BB-MO-CBS-pex

1: $\mathcal{S} \leftarrow \emptyset$; $Open \leftarrow \emptyset$
2: **for all** $i \in I$ **do**
3: $\quad \Pi_o^i \leftarrow ApproxLowLevelSearch(i, \emptyset, \boldsymbol{\varepsilon})$
4: $\mathbb{P}_o \leftarrow MergeJointPaths(\{\Pi_o^i | i \in I\}, \boldsymbol{\varepsilon})$
5: add $\langle \emptyset, \{\Pi_o^i | i \in I\}, \mathbb{P}_o \rangle$ to $Open$
6: **while** $Open \neq \emptyset$ **do**
7: $\quad n = \langle \Omega, \{\Pi^i | i \in I\}, \mathbb{P} \rangle \leftarrow Open.extract\_min()$
8: $\quad \mathbb{P}' \leftarrow PruneApproxDom(\mathbb{P}, \mathcal{S}, \boldsymbol{\varepsilon})$
9: $\quad$ **if** $\mathbb{P}' = \emptyset$ **then continue**
10: $\quad$ **if** $\mathbb{P}'.lexFirst \neq \mathbb{P}.lexFirst$ **then**
11: $\quad\quad$ add $\langle \Omega, \{\Pi^i | i \in I\}, \mathbb{P}' \rangle$ to $Open$
12: $\quad\quad$ **continue**
13: $\quad cft \leftarrow DetectConflict(\mathbb{P}'.lexFirst)$
14: $\quad$ **if** *cft* does not exist **then**
15: $\quad\quad AddSolution(\mathbb{P}'.lexFirst, \mathcal{S}, \boldsymbol{\varepsilon})$
16: $\quad\quad$ remove $\mathbb{P}'.lexFirst$ from $\mathbb{P}'$
17: $\quad\quad$ **if** $\mathbb{P}' \neq \emptyset$ **then**
18: $\quad\quad\quad$ add $\langle \Omega, \{\Pi^i | i \in I\}, \mathbb{P}' \rangle$ to $Open$
19: $\quad\quad$ **continue**
20: $\quad \{\omega^i, \omega^j\} \leftarrow GenerateConstraints(cft)$
21: $\quad$ **for all** $i' \in \{i, j\}$ **do**
22: $\quad\quad \{\Pi_{\text{new}}^i | i \in I\} \leftarrow \{\Pi^i | i \in I\}$
23: $\quad\quad \Omega_{\text{new}} \leftarrow \Omega \cup \{\omega^{i'}\}$
24: $\quad\quad \Pi_{\text{new}}^{i'} \leftarrow ApproxLowLevelSearch(i', \Omega_{new}, \boldsymbol{\varepsilon})$
25: $\quad\quad \mathbb{P}_{\text{new}} \leftarrow MergeJointPaths(\{\Pi_{\text{new}}^i | i \in I\}, \boldsymbol{\varepsilon})$
26: $\quad\quad$ add $\langle \Omega_{\text{new}}, \{\Pi_{\text{new}}^i | i \in I\}, \mathbb{P}_{\text{new}} \rangle$ to $Open$
27: **return** $\mathcal{S}$

Algorithm 3 Functions for BB-MO-CBS-pex

1: **procedure** $Merge(P, P', \boldsymbol{\varepsilon})$
2: $\quad P_{\text{new}} \leftarrow$ a copy of P or P'
3: $\quad \mathbf{A}(P_{\text{new}}) \leftarrow vector\_min(\mathbf{A}(P), \mathbf{A}(P'))$
4: $\quad$ **if** $\mathbf{c}(P_{\text{new}}) \preceq_{\boldsymbol{\varepsilon}} \mathbf{A}(P_{\text{new}})$ **then return** P_{new}
5: $\quad$ **else return** $\emptyset$
6: **procedure** $MergeJointPaths(\{\Pi^i \mid i \in I\}, \boldsymbol{\varepsilon})$
7: $\quad \mathbb{P}_1 \leftarrow \Pi^1$
8: $\quad$ **for all** $i = 2, 3, \ldots, |I|$ **do**
9: $\quad\quad \mathbb{P}_i \leftarrow \emptyset$
10: $\quad\quad$ **for all** $\langle P_{i-1} = \{\pi^1, \pi^2, \ldots, \pi^{i-1}\}, \pi^i \rangle \in \mathbb{P}_{i-1} \times \Pi^i$ **do**
11: $\quad\quad\quad P \leftarrow \{\pi^1, \pi^2, \ldots, \pi^i\}$
12: $\quad\quad\quad \mathbf{A}(P) \leftarrow \mathbf{A}(P_{i-1}) + \mathbf{A}(\pi^i)$
13: $\quad\quad\quad$ **if** $\exists P' \in \mathbb{P}_i \; Merge(P, P', \boldsymbol{\varepsilon}) \neq \emptyset$ **then**
14: $\quad\quad\quad\quad$ replace P' in $\mathbb{P}_i$ with $Merge(P, P', \boldsymbol{\varepsilon})$
15: $\quad\quad\quad$ **else** add P to $\mathbb{P}_i$
16: $\quad$ **return** $\mathbb{P}_{|I|}$
17: **procedure** $PruneApproxDom(\mathbb{P}, \mathcal{S}, \boldsymbol{\varepsilon})$
18: $\quad \mathbb{P}' \leftarrow$ a copy of $\mathbb{P}$
19: $\quad$ **for all** $P \in \mathbb{P}'$ **do**
20: $\quad\quad$ **if** $\exists P_{sol} \in \mathcal{S} \; \mathbf{c}(P_{sol}) \preceq_{\boldsymbol{\varepsilon}} \mathbf{A}(P)$ **then**
21: $\quad\quad\quad \mathbf{A}(P_{sol}) \leftarrow vector\_min(\mathbf{A}(P), \mathbf{A}(P_{sol}))$
22: $\quad\quad\quad$ remove P from $\mathbb{P}'$
23: $\quad$ **return** $\mathbb{P}'$
24: **procedure** $AddSolution(P, \mathcal{S}, \boldsymbol{\varepsilon})$
25: $\quad$ **if** $\exists P_{\text{sol}} \in \mathcal{S} \; Merge(P, P_{\text{sol}}, \boldsymbol{\varepsilon}) \neq \emptyset$ **then**
26: $\quad\quad$ replace P_{sol} in $\mathcal{S}$ with $Merge(P, P_{\text{sol}}, \boldsymbol{\varepsilon})$
27: $\quad$ **else** add P to $\mathcal{S}$

joint path and prefer the less-conflicting joint path as the representative joint path when merging (Line 2 of Algorithm 3).

Eager solution update: BB-MO-CBS and BB-MO-CBS-pex can be considered as updating solutions "lazily", that is, they try to add solutions to $\mathcal{S}$ only when extracting a CT node n from $Open$ with its current joint path being conflict-free. We propose an eager solution-update scheme, which can be applied to both BB-MO-CBS(-$\boldsymbol{\varepsilon}$) and BB-MO-CBS-pex: In BB-MO-CBS-$\boldsymbol{\varepsilon}$ with eager solution update, after Line 8 of Algorithm 1, we remove all conflict-free joint paths from $\mathbb{P}'$, add them to $\mathcal{S}$, and remove all dominated solutions from $\mathcal{S}$. In BB-MO-CBS-pex with eager solution update, after Line 8 of Algorithm 2, we remove all conflict-free joint paths from $\mathbb{P}'$ and call $AddSolution$ to add them to $\mathcal{S}$. After each new solution P is added to $\mathcal{S}$ (or merged with another solution in $\mathcal{S}$) by $AddSolution$, we also remove other solutions that are weakly dominated by P from $\mathcal{S}$.

BB-MO-CBS-k

In practice, choosing an appropriate $\boldsymbol{\varepsilon}$-value for a given MO-MAPF problem instance can be challenging. If $\boldsymbol{\varepsilon}$ is set too large, BB-MO-CBS-$\boldsymbol{\varepsilon}$ or BB-MO-CBS-pex might return only one solution, which provides no trade-off to users. If $\boldsymbol{\varepsilon}$ is set too small, BB-MO-CBS-$\boldsymbol{\varepsilon}$ or BB-MO-CBS-pex might not benefit from approximation at all. Instead of specifying an approximation factor, one might prefer to specify a desirable number of solutions k. Therefore, we propose BB-MO-CBS-k, a variant of BB-MO-CBS-pex that computes a set of up to k solutions for any user-specified k-value. In BB-MO-CBS-k, all components of $\boldsymbol{\varepsilon}$ are equal, i.e., $\boldsymbol{\varepsilon} = [\varepsilon, \varepsilon, \ldots, \varepsilon]$, and we will denote $\boldsymbol{\varepsilon}$ simply as ε in the rest of this section.

BB-MO-CBS-k builds upon BB-MO-CBS-pex with the following changes:

1. The approximation factor ε is initialized to zero and dynamically updated by a modified $AddSolution$ function, which is explained later.
2. Every time after the low-level search for an agent a^i, BB-MO-CBS-k calls function $MergeUntil$ to merge the set of paths Π^i until the size of Π^i is no larger than k. $MergeUntil$ is explained later.
3. BB-MO-CBS-k uses a modified $MergeJointPaths$ function, which always outputs a set of at most k joint paths, and a modified $AddSolution$ function, which always keeps the size of $\mathcal{S}$ no larger than k. The two modified functions are also explained later.

For a joint path P, we define its *boundedness factor* as

$$BF(P) := \max\left(0, \max_{i=1,2,\ldots,M} \frac{c_i(P)}{A_i(P)} - 1\right),$$

which is the smallest ε-value such that joint path P is ε-bounded (i.e., $c_i(P) \leq (1+\varepsilon)A_i(P)$ for $i = 1, 2, \ldots, M$).

Algorithm 4 shows the pseudocode for the $MergeUntil$ function and the modified $Merge$, $MergeJointPaths$, and $AddSolution$ functions. The modified $Merge$ function merges paths without checking the $\boldsymbol{\varepsilon}$-boundedness. The $MergeUntil$ function iteratively chooses two joint paths

Algorithm 4 Functions for BB-MO-CBS-k

1: **procedure** $Merge(P, P')$
2: $P_{\text{new}} \leftarrow$ a copy of P or P'
3: $\mathbf{A}(P_{\text{new}}) \leftarrow vector\_min(\mathbf{A}(P), \mathbf{A}(P'))$
4: **return** P_{new}
5: **procedure** $MergeUntil(\mathbb{P}, k)$
6: **while** $|\mathbb{P}| > k$ **do**
7: choose two joint paths P and P' from $\mathbb{P}$ such that $BF(Merge(P, P'))$ is minimized
8: remove P and P' from $\mathbb{P}$
9: add $Merge(P, P')$ to $\mathbb{P}$
10: **procedure** $MergeJointPaths(\{\Pi^i \mid \forall i \in I\}, k)$
11: $\mathbb{P}_1 \leftarrow \Pi^1$
12: $MergeUntil(\mathbb{P}_1, k)$
13: **for all** $i = 2, 3, \ldots, |I|$ **do**
14: $\mathbb{P}_i \leftarrow \emptyset$
15: **for all** $\langle P_{i-1} = \{\pi^1, \pi^2, \ldots, \pi^{i-1}\}, \pi^i\rangle \in \mathbb{P}_{i-1} \times \Pi^i$ **do**
16: $P \leftarrow \{\pi^1, \pi^2, \ldots, \pi^i\}$
17: $\mathbf{A}(P) \leftarrow \mathbf{A}(P_{i-1}) + \mathbf{A}(\pi^i)$
18: add P to $\mathbb{P}_i$
19: $MergeUntil(\mathbb{P}_i, k)$
20: **return** $\mathbb{P}_{|I|}$
21: **procedure** $AddSolution(P, \mathcal{S}, \varepsilon, k)$
22: add P to $\mathcal{S}$
23: $MergeUntil(\mathcal{S}, k)$
24: $\varepsilon \leftarrow \max(\{BF(P) \mid P \in \mathcal{S}\} \cup \{\varepsilon\})$

P and P' from a given set of joint paths $\mathbb{P}$ such that $BF(Merge(P, P'))$ is minimized (by iterating all possible combinations of joint paths in $\mathbb{P}$) and replaces P and P' with $Merge(P, P')$ in $\mathbb{P}$ until the size of $\mathbb{P}$ is no larger than k. The modified $MergeJointPaths$ function calls $MergeUntil$ to keep the sizes of $\mathbb{P}_i$, $i = 1, 2, \ldots, |I|$, no larger than k (Lines 12 and 19). The modified $AddSolution$ function also uses $MergeUntil$ to keep the size of $\mathcal{S}$ no larger than k. Additionally, it updates the approximation factor ε to the largest bounded factor of $\mathcal{S}$. We also generalize the $MergeUntil$ function to merge paths output by the low-level search. We omit the pseudocode for this function because the generalization is trivial. The boundedness factor for path is defined in the same way as the one for joint path. When BB-MO-CBS-k terminates, it returns $\mathcal{S}$, which contains no more than k solutions. Additionally, $\mathcal{S}$ is guaranteed to be an ε-approximate frontier for the eventual value of ε.

Theoretical Results

Definition (CVN set) *Given a set of joint paths $\mathbb{P}$ and a CT node n with constraints Ω, let $CVN(n, \mathbb{P})$ be the set of all joint paths that (i) satisfy all constraints in Ω, (ii) are conflict-free, and (iii) are of costs not weakly dominated by the apex of any joint path in $\mathbb{P}$.*

We say a node n *permits* a joint path P with respect to $\mathbb{P}$ if $P \in CVN(n, \mathbb{P})$.

Lemma 1. *For agent a^i and constraints Ω, let $\Pi^i := ApproxLowLevelSearch(i, \Omega, \varepsilon)$. We have (1) for each path π' of agent a^i that satisfies Ω, there exists a path $\pi \in \Pi^i$ with $\mathbf{A}(\pi) \preceq \mathbf{c}(\pi')$, and (2) all paths in Π^i are ε-bounded.*

Proof. The lemma is shown by Theorem 1 in the paper of A*pex (Zhang et al. 2022). □

Lemma 2. *Let $n_{\text{new}} = \langle \Omega_{\text{new}}, \{\Pi^i_{\text{new}} | i \in I\}, \mathbb{P}_{\text{new}}\rangle$ denote the CT node that BB-MO-CBS-pex inserts to Open on Line 26 of Algorithm 2. We have (1) for any joint path P' that satisfies Ω_{new}, there exists a joint path $P \in \mathbb{P}_{\text{new}}$ with $\mathbf{A}(P) \preceq \mathbf{c}(P')$ and (2) all joint paths in $\mathbb{P}_{\text{new}}$ are ε-bounded.*

Proof. Since $\mathbb{P}_{\text{new}}$ is computed by $MergeJointPaths$ (defined on Lines 6-16 of Algorithm 3) on Line 25 of Algorithm 2, we use induction to show that, before $MergeJointPaths$ terminates (namely before Line 16 of Algorithm 3), $\mathbb{P}_i$, $i = 1, 2, \ldots, |I|$ mentioned there satisfies that (Condition 1) for any joint path P' for agent indices $\{1, 2, \ldots, i\}$ that satisfies Ω_{new}, there exists a joint path $P \in \mathbb{P}_i$ with $\mathbf{A}(P) \preceq \mathbf{c}(P')$ and, (Condition 2) all joint paths in $\mathbb{P}_i$ are ε-bounded. Note that $\Pi^i, i \in I$, in $MergeJointPaths$ refers to the same set of paths as Π^i_{new}.

Conditions 1 and 2 hold for $\mathbb{P}_1$ because $\mathbb{P}_1$ is set to Π^1_{new} (Line 7), which was computed by $ApproxLowLevelSearch$, and Lemma 1 holds. Assume that Conditions 1 and 2 hold for $i = 1, 2, \ldots, j-1$. For any joint path $P' = \{\pi'^1, \pi'^2, \ldots, \pi'^j\}$ that satisfies Ω_{new}, there exists a joint path $P_{j-1} \in \mathbb{P}_{j-1}$ with $\mathbf{A}(P_{j-1}) \preceq \mathbf{c}(\{\pi'^1, \pi'^2, \ldots, \pi'^{j-1}\})$. From Lemma 1, there also exists a path $\pi^j \in \Pi^j_{\text{new}}$ with $\mathbf{A}(\pi^j) \preceq \mathbf{c}(\pi'^j)$. When $MergeJointPaths$ reaches Line 13 with the combination of P_{j-1} and π^j, the resulting joint path P mentioned there is either added to $\mathbb{P}_j$ or merged with another joint path in $\mathbb{P}_j$. Either case results in a joint path whose apex weakly dominates $\mathbf{c}(P')$ in $\mathbb{P}_j$. $MergeJointPaths$ might merge this joint path several (more) times with other joint paths, but the apex of this joint path will still weakly dominate $\mathbf{c}(P')$. Therefore, Condition 1 holds for $i = j$. Since both P_{j-1} and π^j are ε-bounded and $\mathbf{A}(P) = \mathbf{A}(P_{j-1}) + \mathbf{A}(\pi^j)$, P is also ε-bounded. Since $\mathbb{P}_j$ is updated either on Line 15, where ε-bounded joint path P is added to $\mathbb{P}_j$, or on Line 14, where P is merged with a joint path in $\mathbb{P}_j$ with the resulting joint path being ε-bounded (as required by the $Merge$ function), Condition 2 holds for $i = j$, too. Therefore, Conditions 1 and 2 hold for $i = 1, 2, \ldots, |I|$. Since $MergeJointPaths$ returns $\mathbb{P}_{|I|}$ as $\mathbb{P}_{\text{new}}$, Lemma 2 holds. □

Lemma 3. *When BB-MO-CBS-pex reaches Line 13 of Algorithm 2, consider CT node n and set of joint paths $\mathbb{P}'$ mentioned there, for any joint path $P \in CVN(n, \mathcal{S})$, there exists a joint path $P' \in \mathbb{P}'$ with $\mathbf{A}(P') \preceq \mathbf{c}(P)$.*

Proof. Before BB-MO-CBS-pex reaches Line 13, CT node n might have been previously extracted from and reinserted to *Open* with different sets of joint paths. Let $\mathbb{P}_{\text{gen}}$ denote the set of joint paths computed by $MergeJointPaths$ when n was generated on Line 26. From Lemma 2, for any joint path P that satisfies Ω, there exists a joint path $P' \in \mathbb{P}_{\text{gen}}$ with $\mathbf{A}(P') \preceq \mathbf{c}(P)$. We prove Lemma 3 by contradiction: Assume that P' is not in $\mathbb{P}'$, which happens only if P' has been removed on Line 8 or 16. If P' was removed on Line 16, it was added to $\mathcal{S}$ on Line 15. If P' was removed on Line 8, or more specifically, on Line 22 of Algorithm 3, the apex of some solution was updated to weakly dominate $\mathbf{A}(P')$ (Line 21 of Algorithm 3). In both cases, there exists a solution in $\mathcal{S}$ whose apex weakly dominates $\mathbf{A}(P')$. BB-MO-CBS-pex might later merge this solution several times with other solutions on Line 26 of Algorithm 3 or update its apex on Line 21 of Algorithm 3, but the apex of this solution will still weakly dominate $\mathbf{A}(P')$. We hence find a contradiction because, by the definition of CVN sets, the cost of P is not weakly dominated by the apex of any solution in $\mathcal{S}$. Thus, P' is in $\mathbb{P}'$. □

Lemma 4. *When BB-MO-CBS-pex reaches (but before executing) Line 7 of Algorithm 2, for any solution P, if there does not exist a solution $P_{sol} \in \mathcal{S}$ with $\mathbf{A}(P_{sol}) \preceq \mathbf{c}(P)$, there exists a CT node $n \in$ Open, which permits P with respect to $\mathcal{S}$.*

Proof. We prove this lemma by induction. After the initialization, $Open$ contains only the root node n_o, which has an empty constraint set and thus permits any solution with respect to $\mathcal{S}$ because $\mathcal{S}$ is empty. Therefore, the lemma holds for the first iteration. Assuming that Lemma 4 holds when BB-MO-CBS-pex reaches Line 7 at an iteration, if there exists a solution $P_{sol} \in \mathcal{S}$ with $\mathbf{A}(P_{sol}) \preceq \mathbf{c}(P)$, there will always exist a solution whose apex weakly dominates $\mathbf{c}(P)$ afterward (because Lines 21 and 26 in Algorithm 3 can only decrease each component of $\mathbf{A}(P_{sol})$), and hence the lemma holds for the next iteration. Otherwise, there must exist a node $n = \langle \Omega, \{\Pi^i | i \in I\}, \mathbb{P}\rangle$ in $Open$ that permits P with respect to $\mathcal{S}$. If n is not extracted from $Open$, it still permits P till the next iteration. Therefore, the lemma holds for the next iteration. There are three cases if n is extracted from $Open$: First, some joint paths in $\mathbb{P}$ are removed on Lines 8. Because of Lemma 3 and $P \in CVN(n, \mathcal{S})$, $\mathbb{P}'$ is not empty, and hence n is reinserted into $Open$. The lemma holds for the next iteration. Second, $\mathbb{P}'.lexFirst$ is conflict-free. If $\mathbf{A}(\mathbb{P}'.lexFirst) \preceq \mathbf{c}(P)$, then $\mathbb{P}'.lexFirst$ is added to $\mathcal{S}$. The lemma holds for the next iteration. Otherwise, from Lemma 3, $\mathbb{P}'$ is not empty, n is reinserted into $Open$ (Line 18), and the lemma also holds for the next iteration. Third, BB-MO-CBS-pex generates two child nodes to resolve a conflict (Lines 20-26). P cannot violate both ω^i and ω^j mentioned on Line 20. Therefore, at least one of the two child nodes added to $Open$ on Line 26 permits P. Thus, the lemma holds for the next iteration. □

Theorem 1. *Given an MO-MAPF instance that has at least one solution, when BB-MO-CBS-pex terminates, $\mathcal{S}$ is an $\boldsymbol{\varepsilon}$-approximate frontier.*

Proof. From Lemma 2, all joint paths in the joint path set of a generated node are $\boldsymbol{\varepsilon}$-bounded. Additionally, whenever BB-MO-CBS-pex merges joint paths (Line 26 of Algorithm 3) or updates the apex of an joint path (Line 21 of Algorithm 3), the resulting joint path or the updated joint path is still always $\boldsymbol{\varepsilon}$-bounded. Therefore, for any $P_{sol} \in \mathcal{S}$, we have $\mathbf{c}(P_{sol}) \preceq_{\boldsymbol{\varepsilon}} \mathbf{A}(P_{sol})$. From Lemma 4, we know that, for any solution P, there exists a solution $P_{sol} \in \mathcal{S}$ with $\mathbf{A}(P_{sol}) \preceq \mathbf{c}(P)$, and hence $\mathbf{c}(P_{sol}) \preceq_{\boldsymbol{\varepsilon}} \mathbf{c}(P)$, when $Open$ is empty, i.e., when BB-MO-CBS-pex terminates. Therefore, $\mathcal{S}$ is an $\boldsymbol{\varepsilon}$-approximate frontier. □

Lemma 5. *BB-MO-CBS-pex never reaches Line 20 of Algorithm 2 with a CT node n if there exists a solution P with $\mathbf{c}(P) \prec \mathbf{g}(n)$.*

Proof. We prove this lemma by contradiction. Assume that BB-MO-CBS-pex reaches Line 20 with such a CT node n and a solution P with $\mathbf{c}(P) \prec \mathbf{g}(n)$. There does not exist a solution $P_{sol} \in \mathcal{S}$ with $\mathbf{A}(P_{sol}) \preceq \mathbf{c}(P)$ because, otherwise, the current joint path of n (whose apex is equal to $\mathbf{g}(n)$) would not be in the joint path set $\mathbb{P}'$ mentioned on Line 8. In this case, BB-MO-CBS-pex would reach Line 12 instead of Line 20. From Lemma 4, there exists a node $n' = \langle \Omega', \{\Pi'^i | i \in I\}, \mathbb{P}'\rangle$ in $Open$ that permits P with respect to $\mathcal{S}$. When n' was generated, there was a joint path P' in the set of joint paths with $\mathbf{A}(P') \preceq \mathbf{c}(P)$ (and hence $\prec \mathbf{g}(\mathbf{n})$) (Lemma 2). If P' has been removed from $\mathbb{P}'$ by $PruneApproxDom$, there should exist a solution in $\mathcal{S}$ whose apex weakly dominates $\mathbf{A}(P')$ (and hence $\mathbf{c}(P)$), which we have already disproved. Therefore, P' is still in $\mathbb{P}'$. Since $\mathbf{g}(n') = \mathbf{A}(\mathbb{P}'.lexFirst)$ is lexicographically smaller than or equal to $\mathbf{A}(P')$, $\mathbf{g}(n')$ is lexicographically smaller than $\mathbf{g}(n)$, which contradicts that n had the lexicographically smallest $\mathbf{g}$-value when it was extracted from $Open$ (Line 7). □

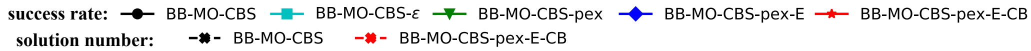
Theorem 2. *Given an MO-MAPF instance that has at least one solution, BB-MO-CBS-pex terminates in finite time.*

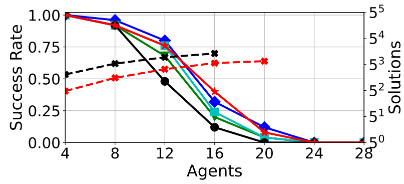
Proof. Because the given graph G is finite (i.e., has finite vertices and edges) and the cost of each edge in G is a positive M-dimensional vector, there are only a finite number of $\boldsymbol{\varepsilon}$-bounded joint paths whose apexes are not dominated by the cost of any solution. Because of Lemma 5, when BB-MO-CBS-pex reaches Line 20 with CT node n, the current joint path of n must be a joint path whose apex is not dominated by the cost of any solution. When generating a child node for a node n, BB-MO-CBS-pex adds a new constraint, which prevents at least one joint path (i.e., the current joint path of n), whose apex is not dominated by the cost of any solution. Therefore, the CT of BB-MO-CBS-pex must contain a finite number of nodes. Because each node in BB-MO-CBS-pex can only be reinserted to $Open$ for finite times, BB-MO-CBS-pex terminates in finite time. □

Experimental Results

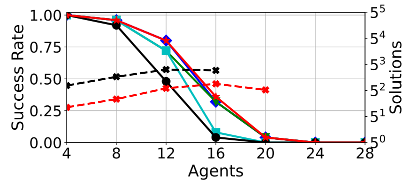
In our experimental results, we evaluated (1) BB-MO-CBS, (2) BB-MO-CBS-ε, (3) BB-MO-CBS-pex, (4) BB-MO-CBS-pex-E (BB-MO-CBS-pex with *E*ager solution update), (5) BB-MO-CBS-pex-E-CB (BB-MO-CBS-pex-E with *C*onflict-*B*ased merging), and (6) BB-MO-CBS-k (which also has eager solution update and conflict-based merging). All algorithms are implemented in C++[1] and share a common code base as much as possible. We conducted all experiments on a Ubuntu 20.04.5 laptop with an Intel Core i7-10510U 1.80GHz CPU and 16GB RAM.

[1]https://github.com/FangjiW/BBMOCBS-approx

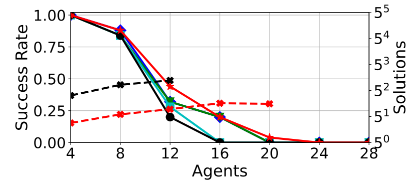




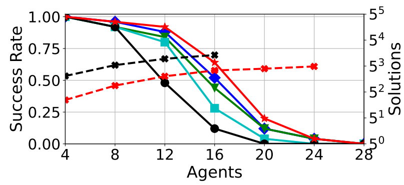
(a) empty-48-48, $\varepsilon = 0.03$



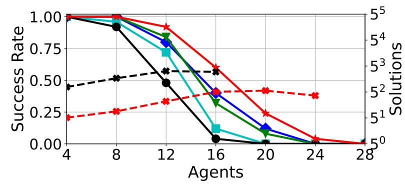
(b) random-32-32-20, $\varepsilon = 0.03$



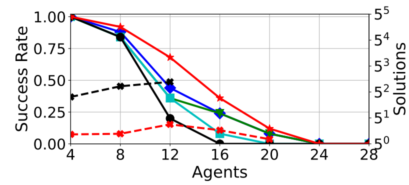
(c) room-32-32-4, $\varepsilon = 0.03$



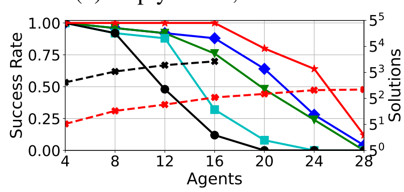
(d) empty-48-48, $\varepsilon = 0.05$



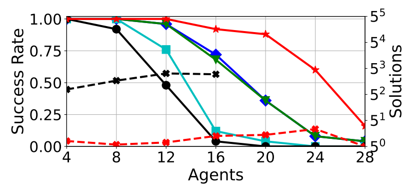
(e) random-32-32-20, $\varepsilon = 0.05$



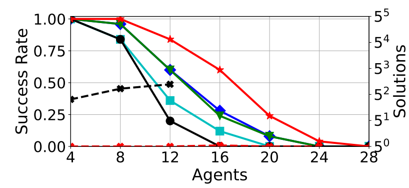
(f) room-32-32-4, $\varepsilon = 0.05$



(g) empty-48-48, $\varepsilon = 0.10$

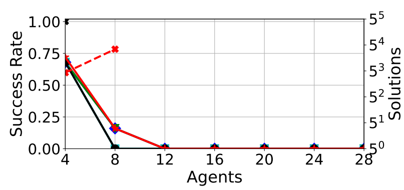


(h) random-32-32-20, $\varepsilon = 0.10$

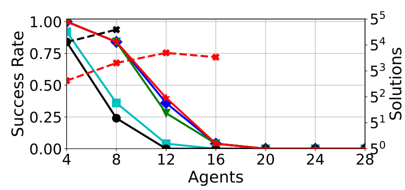


(i) room-32-32-4, $\varepsilon = 0.10$

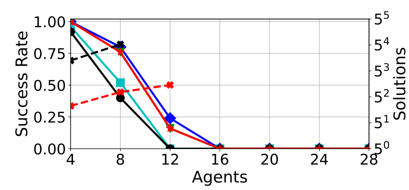
Figure 2: Experimental results for BB-MO-CBS(-ε) and different variants of BB-MO-CBS-pex with two objectives.



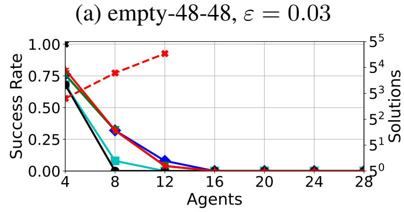
(a) empty-48-48, $\varepsilon = 0.03$



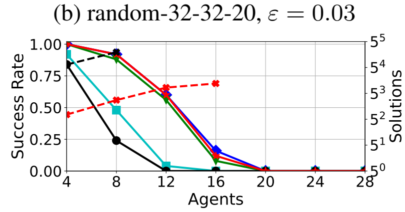
(b) random-32-32-20, $\varepsilon = 0.03$



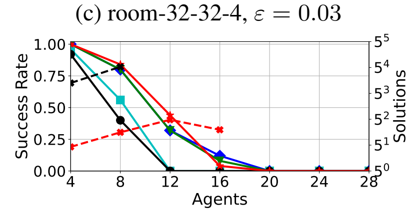
(c) room-32-32-4, $\varepsilon = 0.03$



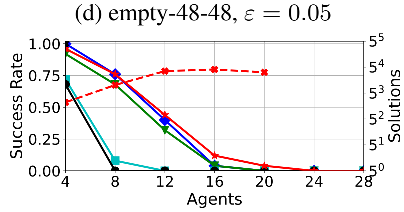
(d) empty-48-48, $\varepsilon = 0.05$



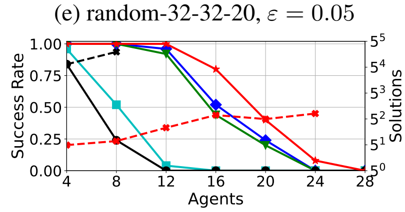
(e) random-32-32-20, $\varepsilon = 0.05$



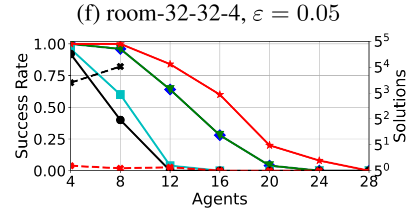
(f) room-32-32-4, $\varepsilon = 0.05$



(g) empty-48-48, $\varepsilon = 0.10$



(h) random-32-32-20, $\varepsilon = 0.10$

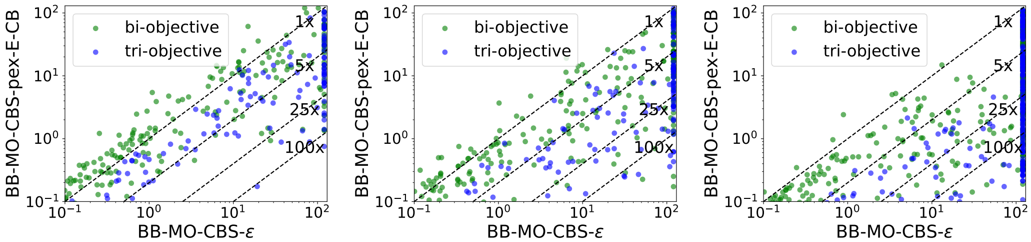


(i) room-32-32-4, $\varepsilon = 0.10$

Figure 3: Experimental results for BB-MO-CBS(-ε) and different variants of BB-MO-CBS-pex with three objectives.

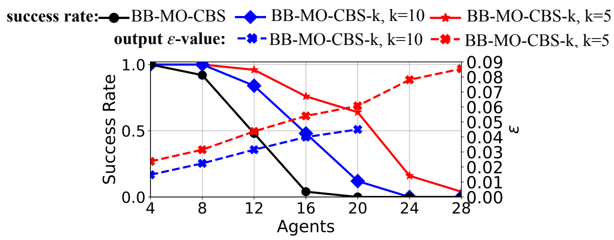
The low level of BB-MO-CBS and BB-MO-CBS-ε is implemented with BOA* for bi-objective domains and NAMOA*dr for domains with more than two objectives. The BB-MO-CBS-pex variants without conflict-based merging use the "greedy" merging strategy, which is proposed by Zhang et al. (2022) and has the best overall performance among different merging strategies.

We use three four-neighbor grids from the MAPF bench-

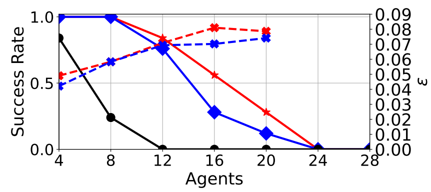


(a) $\varepsilon = 0.03$ (b) $\varepsilon = 0.05$ (c) $\varepsilon = 0.10$

Figure 4: Runtime of BB-MO-CBS-ε and BB-MO-CBS-pex-E-CB for each problem instance.



(a) random-32-32-20, bi-objective



(b) random-32-32-20, tri-objective

Figure 5: Experiment results for BB-MO-CBS and BB-MO-CBS-k.

mark (Stern et al. 2019): empty-48-48, random-32-32-20, and room-32-32-4. We generate the cost for each edge by randomly sampling each cost component from 1 to 5. The MAPF benchmark contains 25 random scenarios for each map, and each scenario provides a list of start-goal pairs. For each scenario, we vary the number of agents N from 4 to 28 and generate problem instances with the first N start-goal pairs. We run experiments with two and three objectives and a runtime limit of 120 seconds for each problem instance.

Different Variants of BB-MO-CBS-pex

We compare BB-MO-CBS, BB-MO-CBS-ε, and different variants of BB-MO-CBS-pex with approximation factors of 0.03, 0.05, and 0.1.

Figures 2 and 3 show the experimental results on instances with two and three objectives, respectively. The solid lines show the success rate (i.e., the percentage of instances solved by an algorithm within the runtime limit) for each algorithm. In all cases, all variants of BB-MO-CBS-pex have higher success rates than BB-MO-CBS-ε, which in turn has higher success rates than BB-MO-CBS. Among the variants of BB-MO-CBS, BB-MO-CBS-pex has the lowest success rate, while BB-MO-CBS-E-CB has the highest success rate in almost all cases, which shows the usefulness of the eager solution update and conflict-based merging techniques. The improvements in success rates of these techniques are more significant for larger ε-values. For example, in random-32-32-30 with two objectives and 20 agents, the addition of conflict-based merging doubles the success rate. The dashed lines in Figures 2 and 3 show the average numbers of solutions of BB-MO-CBS and BB-MO-CBS-pex-E-CB. We can see that introducing approximation to the MO-MAPF problem reduces the sizes of solution sets significantly.

Figure 4 shows the individual runtime of BB-MO-CBS-ε and BB-MO-CBS-pex-E-CB for each problem instance. BB-MO-CBS-pex-E-CB is overall significantly more efficient than BB-MO-CBS-ε, especially when ε becomes larger or the instances have more objectives. The max speed-up of BB-MO-CBS-pex-E-CB over BB-MO-CBS-ε is more than two orders of magnitude.

BB-MO-CBS-k

We compare BB-MO-CBS and BB-MO-CBS-k on random-32-32-20 with two and three objectives. For BB-MO-CBS-k, we use two k-values, namely 5 and 10. The experimental results are shown in Figure 5. The solid lines show the success rate for each algorithm, and we can see that BB-MO-CBS-k has significantly higher success rates than BB-MO-CBS. The dashed lines show the average approximation factor output by BB-MO-CBS-k, and we can see that, with $k = 5$ and $k = 10$, BB-MO-CBS-k still computes solution sets with approximation factors smaller than 0.1.

Conclusions

In this paper, we proposed BB-MO-CBS-pex, which leverages A*pex to compute approximate frontiers for the MO-MAPF problem with a user-specified approximation factor. Based on BB-MO-CBS-pex, we proposed BB-MO-CBS-k, which computes up to k solutions for a user-provided k-value. Our experimental results show that both BB-MO-CBS-pex and BB-MO-CBS-k solved significantly more instances than BB-MO-CBS for different approximation factors and k-values, respectively. We also show that BB-MO-CBS-pex achieved speed-ups up to two orders of magnitude compared to BB-MO-CBS-ε, our baseline approximation variant of BB-MO-CBS.

Acknowledgements

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, and 2112533. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or governments.

References

Breugem, T.; Dollevoet, T.; and van den Heuvel, W. 2017. Analysis of FPTASes for the Multi-Objective Shortest Path Problem. *Computers & Operations Research*, 78: 44–58.

Ehrgott, M. 2005. *Multicriteria Optimization (2nd ed.)*. Springer.

Goldin, B.; and Salzman, O. 2021. Approximate Bi-Criteria Search by Efficient Representation of Subsets of the Pareto-Optimal Frontier. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 149–158.

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.

Ma, H.; Tovey, C.; Sharon, G.; Kumar, T. K. S.; and Koenig, S. 2016. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing problem. In *AAAI Conference on Artificial Intelligence (AAAI)*, 3166–3173.

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-16 Workshop on Planning for Hybrid Systems*.

Perny, P.; and Spanjaard, O. 2008. Near Admissible Algorithms for Multiobjective Search. In *European Conference on Artificial Intelligence (ECAI)*, 490–494.

Ren, Z.; Li, J.; Zhang, H.; Koenig, S.; Rathinam, S.; and Choset, H. 2023. Binary Branching Multi-Objective Conflict-Based Search for Multi-Agent Path Finding. *Proceedings of the International Conference on Automated Planning and Scheduling*, 33(1): 361–369.

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*, 20(2): 1262–1274.

Ren, Z.; Zhan, R.; Rathinam, S.; Likhachev, M.; and Choset, H. 2022. Enhanced multi-objective A* using balanced binary search trees. In *Symposium on Combinatorial Search (SOCS)*, 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.; Atzmon, D.; Walker, T.; Li, J.; Cohen, L.; Ma, H.; Kumar, T. K. S.; Felner, A.; and Koenig, S. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Symposium on Combinatorial Search (SOCS)*, 151–158.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1): 9–20.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI Conference on Artificial Intelligence (AAAI)*, 1443–1449.

Zhang, H.; Salzman, O.; Kumar, T. K. S.; Felner, A.; Ulloa, C. H.; and Koenig, S. 2022. A* pex: Efficient Approximate Multi-Objective Search on Graphs. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 394–403.