A Conflict-Based Search Framework for Multi-Objective Multi-Agent Path Finding

Zhongqiang Ren[1], Sivakumar Rathinam[2] and Howie Choset[1]

***Abstract*—Conventional multi-agent path planners typically compute an ensemble of paths while optimizing a single objective, such as path length. However, many applications may require multiple objectives, say fuel consumption and completion time, to be simultaneously optimized during planning and these criteria may not be readily compared and sometimes lie in competition with each other. The goal of the problem is thus to find a Pareto-optimal set of solutions instead of a single optimal solution. Naively applying existing multi-objective search algorithms, such as multi-objective A* (MOA*), to multi-agent path finding may prove to be inefficient as the dimensionality of the search space grows exponentially with the number of agents. This article presents an approach named Multi-Objective Conflict-Based Search (MO-CBS) that attempts to address this so-called curse of dimensionality by leveraging prior Conflict-Based Search (CBS), a well-known algorithm for single-objective multi-agent path finding, and principles of dominance from multi-objective optimization literature. We also develop several variants of MO-CBS to improve its performance. We prove that MO-CBS and its variants can compute the entire Pareto-optimal set. Numerical results show that MO-CBS outperforms MOM*, a recently developed state-of-the-art multi-objective multi-agent planner.**

***Note to Practitioners*—The motivation of this article originates from the need to optimize multiple path criteria when planning conflict-free paths for multiple mobile robots in applications such as warehouse logistics, surveillance, construction site routing, and hazardous material transportation. Existing methods for multi-agent planning typically consider optimizing a single path criteria. This article develops a novel multi-objective multi-agent planner as well as its variants that are guaranteed to find all Pareto-optimal solutions for the problem. We also provide an illustrative example of the algorithm to plan paths for multiple agents that transport materials in a construction site while optimizing both path length and risk. In this example, computing and visualizing a set of Pareto-optimal solutions makes it intuitive for the practitioner to understand the underlying trade-off between conflicting objectives and to choose the most preferred solution for execution based on their domain knowledge.**

***Index Terms*—Multi-Agent Path Finding, Path Planning, Multi-Objective Optimization.**

I. Introduction

MULTI-Agent Path Finding (MAPF) computes a set of collision-free paths for multiple agents connecting their respective start and goal locations while optimizing a scalar measure of paths. Variants of MAPF have been widely studied in the robotics community over the last few years [33]. In this article, we investigate a natural generalization of the MAPF to include multiple objectives for multiple agents and hence the name *Multi-Objective Multi-Agent Path Finding* (MOMAPF). In MOMAPF, agents have to trade-off multiple objectives such as completion time, travel risk and other domain-specific measures. MOMAPF is a generalization of MAPF, and is therefore NP-Hard [41].

In the presence of multiple conflicting objectives, in general, no (single) solution can simultaneously optimize all the objectives. Therefore, the goal of MOMAPF is to find the set of all Pareto-optimal solutions rather than a single optimal solution as in MAPF. A solution is Pareto-optimal if there exists no other solution that will yield an improvement in one objective without causing a deterioration in at least one of the other objectives. Finding this set of solutions while ensuring collision-free paths for agents in each solution is quite challenging: even though there are many single-agent multi-objective search algorithms [17], [34], [36] that can compute all Pareto-optimal solutions, a naive application of

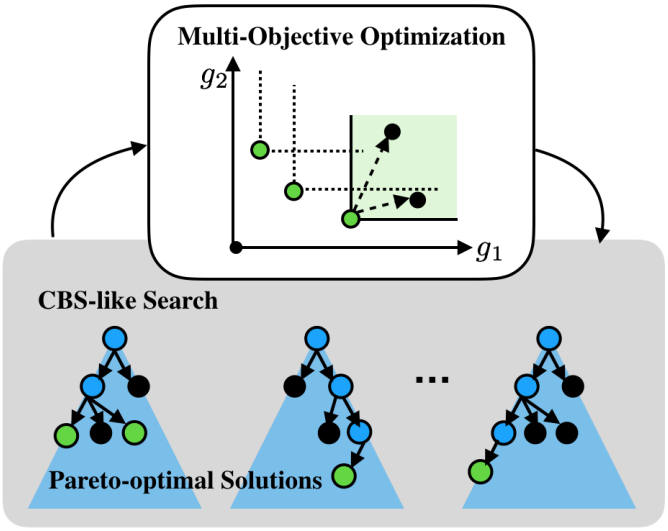


Fig. 1: A conceptual visualization of Multi-Objective Conflict-Based Search. It leverages Conflict-Based Search (CBS) to resolve conflicts between agents (the lower half of the figure) and compares candidate solutions using the dominance principle (the upper half of the figure) from the Multi-Objective Optimization literature in order to find all conflict-free Pareto-optimal solutions.

Manuscript received November, 30, 2021; Revised April, 28, 2022; Accepted June, 11, 2022. This paper was recommended for publication by Editor Jing Li upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by National Science Foundation under Grant No. 2120219 and 2120529. *(Corresponding author: Zhongqiang Ren.)*

Zhongqiang Ren and Howie Choset are with Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA. (email: zhongqir@andrew.cmu.edu; choset@andrew.cmu.edu).

Sivakumar Rathinam is with Texas A&M University, College Station, TX 77843-3123. (email: srathinam@tamu.edu).

such algorithms to the MOMAPF problem may prove to be inefficient as the size of search space grows exponentially with respect to the number of agents [9], [41]. Among the algorithms that optimally solve the single-objective MAPF problems, Conflict-Based Search (CBS) [30] has received significant attention due to its computational efficiency on average. This method has also been extended to solve several other variants of MAPF as noted in [2], [15]. However, how to leverage CBS to solve MOMAPF remains an under-explored question. This article aims to address this gap. By building on multi-objective dominance techniques [3], [17], we develop a new algorithm named Multi-Objective Conflict-Based Search (MO-CBS) (Fig.1) that is able to compute the entire Pareto-optimal set of collision-free paths with respect to multiple objectives.

MO-CBS takes a similar strategy as CBS to resolve conflicts along paths of agents while extending CBS to handle multiple objectives. MO-CBS begins by computing individual Pareto-optimal paths for each agent ignoring agent-agent conflicts and letting agents follow those paths. When a conflict between agents is found along their paths, MO-CBS splits the conflict by adding constraints to the individual search space of each agent (involved in the conflict) and invokes a single-agent multi-objective planner to compute new individual Pareto-optimal paths subject to those added constraints. In addition, MO-CBS uses dominance rules to select candidate solutions for conflict-checking and compares them until all the candidates are either pruned or identified as Pareto-optimal.

MO-CBS is a search framework in a sense that different (single-agent) planners can be used for the low-level search. This work investigates using both BOA* [36], a state-of-the-art single-agent bi-objective planner, and NAMOA*-dr [20], a single-agent planner for multiple objectives, within the MO-CBS framework. Additionally, we also develop a variant of MO-CBS that takes a different expansion strategy on its high-level search to improve memory usage. Compared with an existing approach MOM* [25] that is guaranteed to find all Pareto-optimal solutions for MOMAPF, the numerical results show that the proposed MO-CBS and its variants outperform MOM* in terms of success rates under bounded time in various maps. Our C++ implementation is available online.[1]

Preliminary versions of this research have previously appeared in [23]. This article contains a new proof of completeness and optimality of the proposed approach which applies to all the variants of MO-CBS. We also conduct a new, comprehensive set of experiments to compare MO-CBS with MOM*, and to analyze the performance of MO-CBS variants. For the rest of this article, we review related work in Sec. II and formulate the problem in Sec. III. We first revisit CBS in Sec. IV and then present the basic version of MO-CBS in Sec. V. Variants of MO-CBS are then presented in Sec. VI. We analyze the properties of MO-CBS in Sec. VII and show numerical results in Sec. VIII. Finally, conclusion and future work are presented in Sec. IX.

[1]https://github.com/wonderren/public_cppmomapf

II. Related Work

A. Multi-Objective Path Planning

Multi-objective (*single-agent*) path planning (MOPP) problems aim to find a set of Pareto-optimal paths for the agent between its start location and destination with respect to *multiple* objectives. MOPP arises in applications such as construction site routing [31], hazardous material transportation [5], and others [19], [40]. One common approach to solve a MOPP is to weight the multiple objectives and transform it to a single-objective problem [3], [4]. The transformed problem can then be solved using a corresponding single-objective algorithm. This approach has two main drawbacks: First, the choice of the weights for the objectives must be known a-priori and requires in-depth domain knowledge which may not always be possible; Second, it may also require one to repeatedly solve the transformed single-objective problem for different sets of weights in order to capture the Pareto-optimal set which is quite challenging [18].

Additionally, MOPP and its variants have been solved directly via graph search techniques [17], [24], [27], [34], [36] and evolutionary algorithms [38] where a Pareto-optimal set of solutions is computed exactly or approximately. These graph-based approaches provide guarantees about finding all Pareto-optimal solutions but can run slow for hard cases, where the number of Pareto-optimal solutions is large. MO-CBS developed in this work belongs to this category of search techniques that directly computes a Pareto-optimal set with quality guarantees.

B. Multi-Agent Path Finding

Various methods have been developed to compute an optimal solution for MAPF problems including A*-based approaches [6], [32], subdimensional expansion [37], compilation-based solver [35], integer programming-based methods [11] and Conflict-Based Search (CBS) [30]. In addition, different variants of MAPF have also been considered, such as agents moving with different speeds [1], [21], visiting multiple target locations along the path [22], [26], pickup-and-delivery tasks [13], [14], satisfying kinodynamic constraints [2]. However, all these methods optimize a single objective.

For MOMAPF, heuristic approaches and evolutionary algorithms [16], [28], [38], [39] have been leveraged to solve variants of MOMAPF. For example in [38], agents are not allowed to wait in place and collisions between the agent's paths are modeled in one of the objectives and not as a constraint. Recently, in our prior work, by leveraging M* [37], we developed Multi-Objective M* (MOM*) [25] to solve the MOMAPF with solution quality guarantees. Similarly to M*, MOM* begins by planning for each agent independently and couples agents for planning by searching in their joint configuration space only when two agents are in conflict with each other. In addition, MOM* also leverages the dominance principle from the multi-objective optimization literature to compare two partial solutions in order to find all Pareto-optimal solutions. In this work, we compare the proposed MO-CBS with MOM* in various maps, and the result shows that

MO-CBS achieves higher success rates within a runtime limit than MOM* in several maps.

III. Problem Formulation

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

In this work, we use a superscript $i, j \in I$ over a variable to represent the specific agent that the variable belongs to (e.g. $v^i \in V$ means a vertex with respect to agent i). Let $\pi^i(v_1^i, v_\ell^i)$ be 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 the graph G. Let $g^i(\pi^i(v_1^i, v_\ell^i))$ denote the M-dimensional cost vector associated with the path, which is the sum of the cost vectors of all the edges present in the path, *i.e.*, $g^i(\pi^i(v_1^i, v_\ell^i)) = \Sigma_{j=1,2,\dots,\ell-1}\text{cost}(v_j^i, v_{j+1}^i)$.

All agents share a global clock and all the agents start their paths at time $t = 0$. Each action, either wait or move, for any agent requires one unit of time. Any two agents $i, j \in I$ are said to be in conflict if one of the following two cases happens. The first case is a "vertex conflict" where two agents occupy the same location at the same time. The second case is an "edge conflict" where two agents move through the same edge from opposite directions between times t and $t + 1$ for some t.

Let $v_o^i, v_f^i \in V$ respectively denote the initial location and the destination of agent i. Without loss of generality, to simplify the notations, we also refer to a path $\pi^i(v_o^i, v_f^i)$ for agent i between its initial and final locations as simply π^i. Let $\pi = (\pi^1, \pi^2, \dots, \pi^N)$ represent a joint path for all the agents, which is also called a solution. The cost vector of this solution is defined as the vector sum of the individual path costs over all the agents, *i.e.*, $g(\pi) = \Sigma_i g^i(\pi^i)$.

To compare any two solutions, we compare the cost vectors corresponding to them. Given two vectors a and b, a is said to *dominate* b if every component in a is no larger than the corresponding component in b and there exists at least one component in a that is strictly less than the corresponding component in b. Formally, it is defined as:

Definition 1 (Dominance [17]): Given two vectors a and b of length M, a dominates b, notationally $a \succeq b$, if and only if $a(m) \leq b(m)$, $\forall m \in \{1, 2, \dots, M\}$ and $a(m) < b(m)$, $\exists m \in \{1, 2, \dots, M\}$.[2]

Any two solutions are non-dominated with respect to each other if the corresponding cost vectors do not dominate each other. A solution π is non-dominated with respect to a set of solutions Π, if π is not dominated by any $\pi' \in \Pi$. Among all conflict-free (*i.e.* feasible) solutions, the set of all non-dominated solutions is called the *Pareto-optimal* set. In this work, we aim 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.

[2]The definition is also referred to as "Pareto Dominance" in the literature (e.g. [4]). To simplify presentation, we call it "dominance" in this work.

IV. A Brief Review of Conflict-based Search

A. Conflicts and Constraints

Let (i, j, v^i, v^j, t) denote a *conflict* between agent $i, j \in I$, with $v^i, v^j \in V$ representing the vertex of agent i, j at time t. In addition, to represent a vertex conflict, v^i is required to be the same as v^j and they both represent the location where vertex conflict happens. To represent an edge conflict, v^i, v^j denote the adjacent vertices that agent i, j swap at time t and $t+1$. Given a pair of individual paths π^i, π^j of agent $i, j \in I$, to detect a conflict, let $\Psi(\pi^i, \pi^j)$ represent a conflict checking function that returns either an empty set if there is no conflict, or the first conflict detected along π^i, π^j.

A conflict (i, j, v^i, v^j, t) can be avoided by adding a corresponding constraint to the path of either agent i or agent j. Specifically, let $\omega^i = (i, u_a^i, u_b^i, t), u_a^i, u_b^i \in V$ denote a *constraint* belonging to agent i, which is generated from conflict (i, j, v^i, v^j, t) with $u_a^i = v^i, u_b^i = v^j$ and the following specifications:

- If $u_a^i = u_b^i$, ω^i forbids agent i from entering u_a^i at time t and is named as a *vertex constraint* as it corresponds to a vertex conflict.
- If $u_a^i \neq u_b^i$, ω^i forbids agent i from moving from u_a^i to u_b^i between time t and $t + 1$ and is named as an *edge constraint* as it corresponds to an edge conflict.

Given a set of constraints Ω, let $\Omega^i \subseteq \Omega$ represent the subset of all constraints in Ω that belong to agent i, and clearly $\Omega = \bigcup_{i \in I} \Omega^i$. Additionally, a path π^i is said to be *consistent* with respect to Ω if π^i satisfies every constraint in Ω^i. A joint path π is consistent with respect to Ω if every individual path $\pi^i \in \pi$ is consistent.

B. Two Level Search

CBS is a two level search algorithm. The low-level search in CBS is a single-agent path planner that plans an optimal (*i.e.* minimum cost) and consistent path for an agent i with respect to the set of constraints in Ω^i. If there is no consistent path for agent i given Ω^i, the low-level search reports failure.

For the high-level search, CBS constructs a search tree $\mathcal{T}$ with each tree node P containing:

- $\pi = (\pi^1, \pi^2, \dots, \pi^N)$, a joint path that connects the start and destination vertices of agents respectively,
- g, a scalar cost value associated with π and
- a set of constraints Ω.

The root node P_o of $\mathcal{T}$ has an empty set of constraints $\Omega_o = \emptyset$ and the corresponding joint path π_o is constructed by the low-level search for every agent respectively with $\Omega_o^i = \emptyset$.

To "expand" a high-level search node $P_k = (\pi_k, g_k, \Omega_k)$, where subscript k identifies a specific node, conflict checking $\Psi(\pi_k^i, \pi_k^j)$ is computed for any pair of individual paths in π_k with $i, j \in I, i \neq j$. If there is no conflict detected, a solution is found and the algorithm terminates. Otherwise, for the first detected conflict (i, j, v^i, v^j, t), CBS conducts the following

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

1: *Initialization()*
2: $\mathcal{C} \leftarrow \emptyset$
3: **while** OPEN not empty **do**
4: $P_k = (\pi_k, \vec{g}_k, \Omega_k) \leftarrow$ OPEN.pop()
5: // $P_k = (\pi_k, \vec{g}_k, \Omega_k) \leftarrow$ OPEN.pop-tree-by-tree()
6: **if** *Filter*(P_k) **then continue** ▷ End of iteration
7: **if** no conflict detected in π_k **then**
8: *Update*(P_k)
9: add $\vec{g}_k$ to $\mathcal{C}$
10: **continue** ▷ End of iteration
11: $\Omega \leftarrow$ split detected conflict
12: **for all** $\omega^i \in \Omega$ **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: $\pi_l \leftarrow \pi_k$
17: Replace π^i_l (in π_l) with π^i_*
18: $\vec{g}_l \leftarrow$ compute path cost π_l
19: $P_l = (\pi_l, \vec{g}_l, \Omega_l)$
20: **if** not *Filter*(P_l) **then**
21: add P_l to OPEN
22: **return** $\mathcal{C}$

procedures to resolve it. First, CBS *splits* the detected conflict to generate two constraints $\omega^i = (i, u^i_a = v^i, u^i_b = v^j, t)$ and $\omega^j = (j, u^j_a = v^j, u^j_b = v^i, t)$. Second, CBS generates two corresponding nodes $P_{l^i} = (\pi_{l^i}, g_{l^i}, \Omega_{l^i}), P_{l^j} = (\pi_{l^j}, g_{l^j}, \Omega_{l^j})$, where $\Omega_{l^i} = \Omega_k \cup \{w^i\}$ and $\Omega_{l^j} = \Omega_k \cup \{w^j\}$. Finally, CBS lets $\pi_{l^i} \leftarrow \pi_k$ (and $\pi_{l^j} \leftarrow \pi_k$) and then updates the individual path π^i in π_{l^i} (and π^j in π_{l^j}) by calling the low-level search for agent i (and j) with the set of constraints Ω_{l^i} (and Ω_{l^j} respectively). If the low-level search fails to find a consistent path for i (or j), node P_{l^i} (or P_{l^j}) is discarded.

After conflict resolving, CBS inserts generated nodes into OPEN, which is a priority queue containing all candidate high-level nodes. CBS solves a (single-objective) MAPF problem to optimality by iteratively selecting candidate node from OPEN with the smallest g cost, detect conflicts, and then either claims success (if not conflict detected) or resolves the detected conflict which generates new candidate nodes.

Intuitively, from the perspective of the search tree $\mathcal{T}$ constructed by CBS, OPEN contains all leaf nodes in $\mathcal{T}$. In each iteration of the high-level search, a leaf node P_k is selected and checked for conflict. CBS either claims success if paths in P_k are conflict-free or generates new leaf nodes.

V. Multi-objective Conflict-based Search

The proposed Multi-Objective Conflict-Based Search (MO-CBS) is described in Alg. 1 and visualized in Fig. 2. MO-CBS generalizes CBS and has the following *key features* to handle multiple objectives.

A. Initialization

In MO-CBS, to initialize OPEN (line 1 in Alg. 1), a single-agent multi-objective planner (such as NAMOA*-dr [20]) is used for each agent $i \in I$ separately to compute all cost-unique Pareto-optimal paths, Π^i_o, for agent i. A set of joint paths Π_o is generated by taking the combination of $\Pi^i_o, \forall i \in I$, *i.e.* $\Pi_o = \{\pi_o | \pi_o = (\pi^1_o, \pi^2_o, \dots, \pi^N_o), \pi^i_o \in \Pi^i_o, \forall i \in I\}$. Clearly, the size of Π_o is $|\Pi_o| = |\Pi^1_o| \times |\Pi^2_o| \times \cdots \times |\Pi^N_o|$. For each $\pi_o \in \Pi_o$, a corresponding high-level node containing (i) π_o, (ii) the cost vector associated with π_o and (iii) an empty constraint set, is generated and added into OPEN. Intuitively, while the original CBS initializes a single root node and a single search tree $\mathcal{T}$, MO-CBS initializes a number of $R = |\Pi_o|$ root nodes and a "search forest" $\mathcal{T}_r, r \in \{1, 2, \dots, R\}$ where each tree $\mathcal{T}_r$ corresponds to a root node.[3]

In this work, let $\mathcal{C}^*$ denote the Pareto-optimal front of the given problem instance: the set of cost vectors corresponding to the Pareto-optimal set. Let $\mathcal{C}$ denote a set of cost vectors, where each cost vector corresponds to a conflict-free joint path (*i.e.* solution) that is found during the search. $\mathcal{C}$ is initialized to be an empty set (line 2).

B. Finding a Solution

For every search iteration in MO-CBS (lines 3-21), a high-level node P_k with non-dominated cost vectors among all nodes in OPEN is popped.[4] The popped node $P_k = (\pi_k, \vec{g}_k, \Omega_k)$ is first checked for dominance in procedure *Filter* (line 6), where $\vec{g}_k$ is compared with each cost vector in $\mathcal{C}$. If there exists a vector $\vec{g} \in \mathcal{C}$ such that $\vec{g} \leq \vec{g}_k$ (*i.e.* every component in $\vec{g}$ is no larger than the corresponding component in $\vec{g}_k$),[5] then node P_k cannot lead to a cost-unique Pareto-optimal solution and is thus discarded (*i.e.* filtered), and the current search iteration ends. With the *Filter* procedure, each vector in set $\mathcal{C}$ is guaranteed to be unique.

If π_k is conflict-free (lines 7-10), a solution node (*i.e.* a high-level node containing a solution) is identified, and the cost vector $\vec{g}_k$ is first used to update $\mathcal{C}$ in procedure *Update* and then added to $\mathcal{C}$. The purpose of *Update* is to ensure that an existing cost vector in $\mathcal{C}$ is removed if it is dominated by $\vec{g}_k$. Specifically, *Update*(P_k) uses the cost vector $\vec{g}_k$ in P_k to compare with all existing solution cost vectors (that have already been found during the search) in $\mathcal{C}$, and if $\vec{g}_k \succeq \vec{g}, \vec{g} \in \mathcal{C}$, then $\vec{g}$ is removed from $\mathcal{C}$. (Note that $\vec{g}_k$ cannot be equal to $\vec{g}$, since $\vec{g}_k$ would have been discarded in *Filter* otherwise.) This *Update* procedure is necessary due to the fact that a search forest $\mathcal{T}_r, r \in \{1, 2, \dots, R\}$ (rather than a single search tree) is constructed by MO-CBS. When a solution is found, it is not guaranteed to be Pareto-optimal. When a new solution π_k (in high-level node P_k) is found, *Update*(P_k) removes existing solution cost vectors in $\mathcal{C}$ with dominated cost vectors. As a result, when the algorithm terminates, $\mathcal{C}$ is guaranteed to be the same as the Pareto-optimal front $\mathcal{C}^*$. Additionally, for each $\vec{g} \in \mathcal{C}$, there exists

[3]The idea of using a CBS-like search forest to solve multi-agent path planning problems have also been investigated in [8], [10].

[4]In practice, the lexicographic order of cost vectors is often used to prioritize nodes in OPEN [20], [27], [36] and it can guarantee that every popped node has a non-dominated cost vector among all nodes in OPEN. This work follows this common practice. Other types of prioritization for the candidates in OPEN can also be used within the MO-CBS framework.

[5]Note that $\vec{g} \leq \vec{g}_k$ is equivalent to (i) $\vec{g}$ is not dominated by $\vec{g}_k$ and (ii) $\vec{g}$ is also not equal to $\vec{g}_k$.

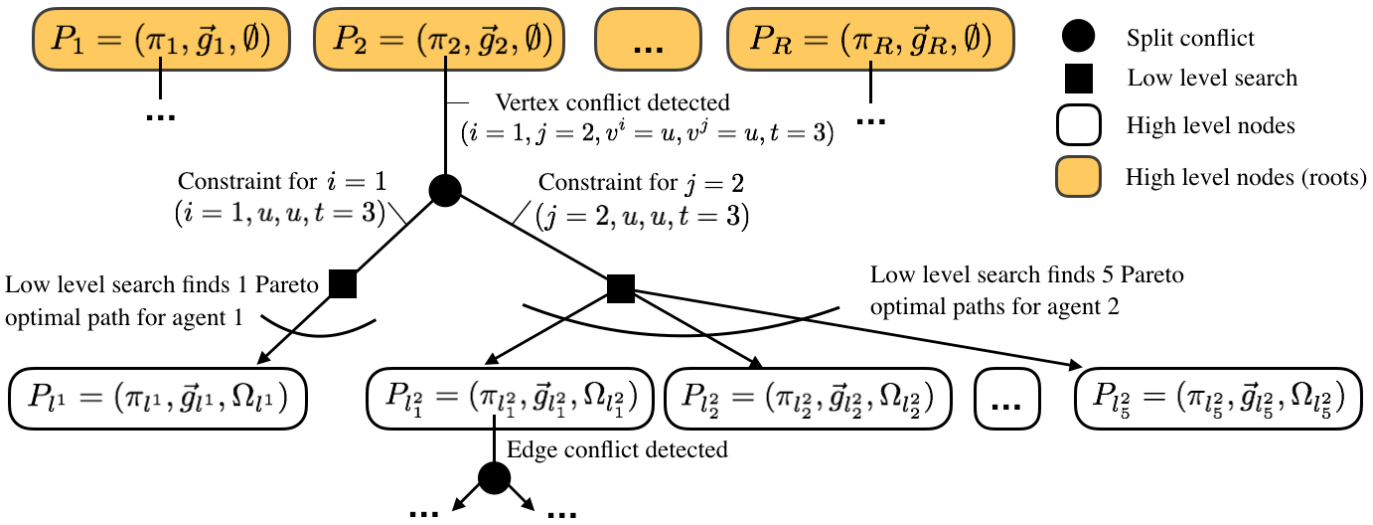


Fig. 2: An illustration of the search process of MO-CBS. MO-CBS initializes multiple root nodes. MO-CBS iteratively selects a candidate high-level node with a non-dominated cost vector from OPEN, splits the detected conflict to generate constraints, conducts low-level search subject to those constraints, and generates new high-level nodes.

a corresponding solution π that is found during the search. Thus, when Alg. 1 terminates with $\mathcal{C} = \mathcal{C}^*$, all cost-unique Pareto-optimal solution are found.

For readers that are familiar with CBS: while CBS terminates when the first solution is identified, MO-CBS continues to search when a solution is identified and terminates only when OPEN is empty in order to identify all cost-unique Pareto-optimal solutions.

C. Conflict Resolution

When a node $P_k = (\pi_k, \vec{g}_k, \Omega_k)$ is popped from OPEN, if π_k contains a conflict, just as in CBS, the detected conflict is split into two constraints and a new set of constraints Ω_l is generated correspondingly. Given an agent i and a constraint set Ω_l, the low-level search (which is explained next) is invoked to compute individual Pareto-optimal paths that are consistent with respect to Ω_l for agent i.

Given Ω_l and an agent i, while in CBS, only one individual optimal path for agent i (that is consistent with Ω_l) is computed, in MO-CBS, there can be multiple consistent Pareto-optimal individual paths for agent i. To find all of them, the low-level search employs a single-agent multi-objective planner (Sec. VI-B) to search a time-augmented graph $G^t = (V^t, E^t) = G \times \{0, 1, \dots, T\}$, where each vertex in $v \in V^t$ is defined as $v = (u, t), u \in V, t \in \{0, 1, \dots, T\}$ and T is a predefined time horizon (a large positive integer). Edges within G^t is represented as $E^t = V^t \times V^t$ where $(u_1, t_1), (u_2, t_2)$ is connected in G^t if $(u_1, u_2) \in E$ and $t_2 = t_1 + 1$. Wait in place is also allowed in G^t (*i.e.* $(u, t_1), (u, t_1 + 1), u \in V$ is connected in G^t). In addition, all vertices and edges in G^t that correspond to vertex constraints and edge constraints in $\Omega_l^i \subseteq \Omega_l$ are removed from the time augmented graph G^t.

The low-level search guarantees to return a set of consistent Pareto-optimal individual paths Π_*^i for agent i subject to the given constraint set. For each path $\pi_*^i \in \Pi_*^i$ computed by the low-level search, a corresponding joint path π_l is generated by first making a copy of π_k and then update the individual path π_l^i in π_l with π_*^i (lines 16-17). If the cost vector of π_l is neither dominated by nor equal to the cost vector of any solution cost vector in $\mathcal{C}$ (line 20), a new node $P_l = (\pi_l, \vec{g}_l, \Omega_l)$ is generated and inserted into OPEN.

D. Relationship to CBS

With only one objective (*i.e.* $M = 1$), MO-CBS is equivalent to CBS in the following sense. Dominance between vectors becomes the "less than" comparison between scalars and the candidate with the minimum g cost in OPEN is popped in each iteration. When the first solution with the minimum cost g^* is found, all other nodes in OPEN must have a cost value no less than g^*, and are thus discarded by the *Filter* procedure, which makes OPEN empty and leads to the termination of MO-CBS. Additionally, the low-level search returns an individual optimal path for an agent when invoked. Only one root node is generated at the initialization step and there is one corresponding search tree built during the search.

VI. Variants of MO-CBS

A. Tree-By-Tree Expansion for the High-Level Search

In MO-CBS, a node with a non-dominated cost vector is selected from OPEN and expanded (conflict checking and splitting). This expansion strategy has two drawbacks. First, all root nodes need to be generated so that a non-dominated one can be selected. Considering an example with ten agents and each agent has ten individual Pareto-optimal paths, MO-CBS needs to generate 10^{10} root nodes, which is computationally prohibitive. The second drawback is that nodes are selected in a "breadth-first" manner in a sense that the selected nodes can belong to different trees. As the number of agents (or objectives) increases, this expansion strategy may lead to a large number of expansions before finding the first solution.

Here we propose a new expansion strategy to bypass these limitations. Let candidates in OPEN be sorted by the tree

$\mathcal{T}_r$ they belong to, and let OPEN$_r$ denote the open list that contains only candidate nodes in tree $\mathcal{T}_r, r \in \{1, 2, \dots, R\}$. Clearly, OPEN$= \bigcup_{r\in 1,2,\dots,R}$OPEN$_r$. Instead of selecting a non-dominated node in OPEN as MO-CBS does, here, only nodes in OPEN$_1$ are considered for selection at first. The selected node is then expanded in the same manner as MO-CBS does. As any newly generated nodes belong to $\mathcal{T}_1$, these nodes must be inserted into OPEN$_1$. Only when OPEN$_1$ depletes, the algorithm then selects candidates from OPEN$_2$ (and then OPEN$_3$, and so on) for expansion. The algorithm terminates when OPEN$_R$ is depleted. We denote MO-CBS with such a "tree-by-tree" (abbreviated as "-t") node selection strategy as MO-CBS-t. MO-CBS-t enables *on-demand* generation of roots and performs a "depth-first" like search by exhaustively examining one tree after another. This allows MO-CBS-t to start the search without initializing all the roots (which is verified in Sec. VIII-C).

B. Different Low-Level Planners

MO-CBS is a search framework in a sense that different low-level planners can be used within the framework, as long as the low-level planner can find all individual Pareto-optimal paths in a time-augmented graph G^t as described in Sec. V-C.

Among the existing single-agent multi-objective search algorithms, NAMOA* [17] is a popular A*-like multi-objective planner. NAMOA*-dr [20] is an improved version of NAMOA* with the so-called "dimensionality reduction" (-dr) technique. Both NAMOA* and NAMOA*-dr can handle an arbitrary number of objectives. Recently, NAMOA*-dr is further expedited by BOA* [36] when there are only two objectives. We refer the reader to the BOA* paper [36] for a detailed discussion about the technical difference between those algorithms.

All these algorithms can be applied to search the aforementioned time-augmented graph G^t and be used as the low-level planner of MO-CBS. We use notation BOA*-st and NAMOA*-dr-st (-st stands for space-time) to indicate that the planner is applied to the time-augmented graph. In this work, to handle an arbitrary number of objectives, we use NAMOA*-dr-st as the low-level planner of MO-CBS and denote the corresponding algorithm MO-CBS-n. When there are only two objectives, we use BOA*-st as the low-level planner of MO-CBS and denote the corresponding algorithm MO-CBS-b. If the aforementioned tree-by-tree expansion strategy is used, we add "-t" to denote the corresponding variant (e.g. MO-CBS-tb, MO-CBS-tn).

VII. Analysis

A. Pareto-optimality

Let Π_* denote the set of all Pareto-optimal (solution) joint paths for a given MOMAPF problem instance. Note that for two solutions $\pi, \pi' \in \Pi_*$, it's possible that their cost vectors $\vec{g}(\pi), \vec{g}(\pi')$ are the same. At any time of the search, define $\Pi_*|\mathcal{C} := \{\pi : \pi \in \Pi_*, \vec{g}(\pi) \notin \mathcal{C}\}$. Intuitively, $\Pi_*|\mathcal{C}$ is the subset of Π_* whose cost vectors have not yet been included in $\mathcal{C}$ during the search. Additionally, "expanding" a high-level node means checking for conflicts and splitting the detected conflict as aforementioned in Sec. V.

Definition 2 (CVset): Given a high-level node $P = (\pi, \vec{g}, \Omega)$, let $CV(P)$ be the set of all joint paths that are (i) consistent with Ω, and (ii) conflict-free ($i.e.$ valid).

Correspondingly, if $\pi' \in CV(P)$, we say P *permits* π'. Intuitively, each joint path in $CV(P)$ is a (conflict-free) solution joint path that satisfies all constraints in Ω.

Definition 3: For each $\pi_* \in \Pi_*|\mathcal{C}$, let $\mathcal{P}(\pi_*)$ denote a high-level search node $(\pi, \vec{g}, \Omega)$ such that (i) $\mathcal{P}(\pi_*)$ permits π_* and (ii) for each agent $i \in I$, $\vec{g}(\pi^i) \leq \vec{g}(\pi^i_*)$.

Correspondingly, if a node P_k satisfies Def. 3 for some $\pi_* \in \Pi_*|\mathcal{C}$, we say P_k is a $\mathcal{P}$-node of π_*.

Corollary 1: For a $\pi_* \in \Pi_*|\mathcal{C}$ and a corresponding $\mathcal{P}$-node of π_*, which is denoted as $(\pi, \vec{g}, \Omega)$, we have $\vec{g}(\pi) \leq \vec{g}(\pi_*)$.

Lemma 1: During the search iterations, for any $\pi_* \in \Pi_*|\mathcal{C}$, if $\mathcal{P}(\pi_*)$ exists and is popped from OPEN for expansion, $\mathcal{P}(\pi_*)$ will not be filtered in the procedure *Filter*.

Proof 1: We prove this Lemma by contradiction. By Def. 3, node $\mathcal{P}(\pi_*)$ has a cost vector $\vec{g} \leq \vec{g}(\pi_*)$. If $\mathcal{P}(\pi_*)$ is removed by the procedure *Filter*, there must exist a feasible solution π' with cost vector $\vec{g}(\pi') \in \mathcal{C}$ such that $\vec{g}(\pi') \leq \vec{g}$. Hence, we have $\vec{g}(\pi') \leq \vec{g}(\pi_*)$. This implies $\vec{g}(\pi') = \vec{g}(\pi_*)$ because π and π_* are feasible solutions, and π_* is Pareto-optimal. However, $\vec{g}(\pi') = \vec{g}(\pi_*)$ is not possible because if $\vec{g}(\pi') \in \mathcal{C}$, then by definition, $\pi_* \notin \Pi_*|\mathcal{C}$. Hence proved. ■

Lemma 2: Let a $\pi_* \in \Pi_*|\mathcal{C}$ be such that there exists a $\mathcal{P}$-node of π_* (denoted as $P_k = (\pi_k, \vec{g}_k, \Omega_k)$) in OPEN, and let π_k have conflicts. Then, if P_k is popped from OPEN and expanded (lines 11-21), there still exists a $\mathcal{P}$-node of π_* after expansion in OPEN.

Proof 2: If π_k has a conflict between agent $i, j \in I$ (line 11), during the expansion, MO-CBS splits the conflict, generates two constraints ω_i, ω_j and invokes the low-level planner (line 14) to find individual cost-unique Pareto-optimal paths subject to the new set of constraints $\Omega_k \bigcup \{\omega_i\}$ or $\Omega_k \bigcup \{\omega_j\}$, which results in two sets $\{P_{l^i}\}, \{P_{l^j}\}$ of new high-level nodes (line 19). Since π_* must satisfy at least one of the constraints ω_i, ω_j, at least one set of nodes $\{P_{l^i}\}, \{P_{l^j}\}$ must permit π_*.

Without losing generality, let $\{P_{l^i}\}$ be a set of nodes that permits π_*, and let Π^i_l denote a set of all individual cost-unique Pareto-optimal path for agent i that is computed by the low-level planner after adding constraint ω_i. Note that there is a one-one correspondence between nodes in $\{P_{l^i}\}$ and individual paths in Π^i_l (lines 16-19). So, there exists at least one individual path $\pi^i_l \in \Pi^i_l$ such that $\vec{g}(\pi^i_l) \leq \vec{g}(\pi^i_*)$ because otherwise π^i_* is non-dominated by any solution in Π^i_l. It means π^i_* is a cost-unique Pareto-optimal path that satisfies all constraints in $\Omega_k \bigcup \{\omega_i\}$ and the low-level planner does not find it, which is impossible. Let $(\pi_l, \vec{g}_l, \Omega_l) \in \{P_{l^i}\}$ denote the generated high-level node corresponding to π^i_l, then $\vec{g}(\pi^j_l) = \vec{g}(\pi^j_k) \leq \vec{g}(\pi^j_*), \forall j \in I, j \neq i$ (by lines 16-17 in MO-CBS, and note that P_k is a $\mathcal{P}$-node of π_*). So, there exists a node in $\{P_{l^i}\}$ that is a $\mathcal{P}$-node of π_*.

Finally, by Lemma 1, $\mathcal{P}$-node of π_* cannot be filtered (line 20), and is thus added to OPEN. ■

Lemma 3: During any iteration of the algorithm, if $\Pi_*|\mathcal{C}$ is non-empty, then for each $\pi_* \in \Pi_*|\mathcal{C}$, there exists at least one $\mathcal{P}(\pi_*)$ in OPEN.

Proof 3: We show this Lemma by mathematical induction.

Base case: During the initialization step of MO-CBS, all individual Pareto-optimal paths of each agent are computed and all possible combinations are enumerated to generate initial joint paths and root nodes. Each root node has an empty constraint set and permits all $\pi_* \in \Pi_*$. Thus, right after initialization ($i.e.$ after line 1), this Lemma holds.

Assumption: Assume the Lemma holds at the start of the k-th iteration of the *while* loop of MO-CBS.

Induction: During the k-th iteration of the *while* loop, let $P_k = (\pi_k, \vec{g}_k, \Omega_k)$ denote a node that is popped from OPEN (line 4). For any $\pi_* \in \Pi_*|\mathcal{C}$, if P_k is not a $\mathcal{P}$-node of π_*, then by assumption, there must exist another node P'_k in OPEN, which is a $\mathcal{P}$-node of π_*. Since P'_k is not popped from OPEN during the k-th iteration, P'_k is still in OPEN and the Lemma holds. Hence, we only need to consider the case where the popped node is a $\mathcal{P}$-node of π_*. By Lemma 1, P_k is not removed during the filtering step (line 6) of the algorithm. Now, π_k must either be conflict-free or have conflicts:

- If π_k is conflict-free (line 7), since $\vec{g}(\pi_k) \leq \vec{g}(\pi_*)$, π_k must also be Pareto-optimal and $\vec{g}(\pi_k) = \vec{g}(\pi_*)$. Since $\vec{g}(\pi_k)$ is added to $\mathcal{C}$ (line 9), by definition, π_* does not belong to $\Pi_*|\mathcal{C}$ any more.
- If π_k has a conflict, as shown in Lemma 2, there is still a $\mathcal{P}$-node in OPEN after the expansion of P_k.

Therefore, at the end of the k-th iteration of MO-CBS, the Lemma holds. Hence, proved. ■

Theorem 1 (Pareto-optimality): For a given problem instance, MO-CBS finds the entire Pareto-optimal front $\mathcal{C}^*$, if it exists.

Proof 4: During the search of MO-CBS, by Lemma 3, for each $\pi_* \in \Pi_*$, either π_* is permitted by some high-level node $\mathcal{P}(\pi_*)$ in OPEN, or $\vec{g}(\pi_*) \in \mathcal{C}$. Therefore, until a Pareto-optimal solution with a cost vector equal to $\vec{g}(\pi_*)$ is added to $\mathcal{C}$, some high-level node $\mathcal{P}(\pi_*)$ will exist in OPEN. MO-CBS terminates only when OPEN depletes, which means all nodes in OPEN are either filtered or expanded. Therefore, MO-CBS will find the entire Pareto-optimal front. ■

B. Completeness

A MOMAPF problem instance is *feasible* if there exists at least one feasible ($i.e.$ conflict-free) joint path for all agents. A MOMAPF problem instance is infeasible otherwise. An algorithm is *complete* if:

- (Statement-1) The algorithm returns a solution in finite time, if the given problem instance is feasible.
- (Statement-2) The algorithm reports failure in finite time, if the given problem instance is infeasible.

We first consider (Statement-1).

Lemma 4: MO-CBS terminates in finite time, if the given MOMAPF problem instance is feasible.

Proof 5: $\mathcal{C}^*$ contains a finite set of Pareto-optimal cost vectors. MO-CBS never expands a high-level node P with a cost vector $\vec{g} \geq \vec{g}^*, \exists \vec{g}^* \in \mathcal{C}^*$, ($i.e.$ every component in $\vec{g}$ is no less than the corresponding component in $\vec{g}^*$), since such a node P is removed by the *Filter* procedure. Graph G is finite ($i.e.$ has finite number of vertices and edges). Each edge[6] in the graph has cost$(e) \in (0, \infty)^M$. Hence, there are only a finite number of joint paths π connecting the starts and destinations of all agents such that $\vec{g}(\pi) \ngeq \vec{g}^*, \exists \vec{g}^* \in \mathcal{C}^*$. In each search iteration, MO-CBS either identifies a feasible solution (a conflict-free joint path), or detects a conflict and generates new constraints which prevents at least one joint path from being generated (lines 12-19) in subsequent search iterations. Hence, MO-CBS terminates in finite time. ■

Theorem 2 (Completeness): MO-CBS finds a solution in finite time, if the given MOMAPF problem instance is feasible.

Proof 6: By Lemma 4, MO-CBS terminates in finite time. By Theorem 1, MO-CBS finds a solution at termination. ■

We now discuss (Statement-2). If the given MOMAPF problem instance is infeasible, then MO-CBS may not terminate. To overcome this issue, similar to [30], we can run some feasibility checking before running MO-CBS. Specifically, given a MOMAPF instance, a corresponding MAPF instance is generated by assigning each edge in graph G a (scalar) unit cost value. Then the generated MAPF instance is verified in polynomial time with the method in [42] to check whether this MAPF instance is feasible. It's obvious that the given MOMAPF instance is feasible if and only if the generated MAPF instance is feasible.

Remark. The definition of completeness in this work is the same as the one in [30]. The proofs are applicable to the basic version of MO-CBS as well as its variants. Specifically, the variant MO-CBS-t differs from MO-CBS in a sense that it re-orders the expansions during the search, and both theorems still hold. For MO-CBS variants that use different low-level planners, since those low-level planners are all guaranteed to find all individual cost-unique Pareto-optimal paths, Lemma 2 is still correct. Consequently, both theorems still hold.

VIII. Numerical Results

A. Test Settings, Implementation and Baseline

We implement all four variants MO-CBS-n, MO-CBS-tn, MO-CBS-b, MO-CBS-tb in C++. We test on a Ubuntu 20.04 laptop with an Intel Core i7-11800H 2.40GHz CPU and 16 GB RAM without multi-threading or compiler optimization. For comparison, we implement the recent MOM* [25] in C++ as a baseline, which can also guarantee finding all cost-unique Pareto-optimal solutions as MO-CBS does. Another method to compute all cost-unique Pareto-optimal solutions is applying a single-agent multi-objective planner to search the joint graph of all agents. This method has been shown to be computationally inefficient, as the size of the joint graph grows exponentially with respect to the number of agents [25], which is thus omitted in this article.

In our implementation, for each agent, the heuristic vector is computed by running M exhaustive backwards Dijkstra

[6]Note that wait in place actions are represented as self-loops in the graph, which are also included in the edge set E.

search from that agent's destination: the m-th Dijkstra search ($m = 1, 2, \ldots, M$) uses edge cost values $c_m(e), \forall e \in E$ ($i.e.$ the m-th component of the cost vector $\vec{c}(e)$ of all edges). In our implementation, for the high-level search, all nodes in OPEN are prioritized in the lexicographic order based on their $\vec{g}$-vectors and the minimum one is popped from OPEN in each search iteration. This implementation guarantees that every popped high-level node has a non-dominated cost vector among all nodes in OPEN.

We select (grid) maps of different types from a MAPF data set [33]. For each map, an un-directed graph G is generated by making each grid four-connected. To assign cost vectors to edges in G, we follow the convention in [20] by assigning each edge an M-dimensional cost vector with each component being an integer randomly sampled from $[1, C_{max}]$, where C_{max} takes different values in the following sections. We use the start-goal pairs from the "random" category in the data set [33], and for each map, there are 25 instances. We set a runtime limit of 300 seconds for each instance.

B. MO-CBS Low-Level Search

We begin by investigating different low-level planners within the framework of MO-CBS. We fix $M = 2$ and compare MO-CBS-b and MO-CBS-n. These two planners expand nodes in the same order for the high-level search, and the only difference between them is the low-level search.

1) Different C_{max}: First, we set $M = 2, N = 4$ (fixed) and vary C_{max} in an empty 16×16 map. Let $\bar{t}$ denote the average runtime (in micro-seconds) of the low-level planner per call during the MO-CBS search. As shown in Table I (a), the low-level planner of MO-CBS-b ($i.e.$ BOA*-st) runs up to twice as fast as the low-level planner of MO-CBS-n ($i.e.$ NAMOA*-dr-st), which can be observed by comparing 15.2ms against 32.8ms in the $C_{max} = 8$ row. The advantage of BOA*-st over NAMOA*-dr-st is more obvious as C_{max} increases. We discuss the reason for this in the ensuing paragraphs. We also show the corresponding number of expansions (#Exp) of the low-level planners in Table I (b). Note that #Exp is not an accurate indicator to compare the computational efforts of BOA*-st and NAMOA*-dr-st, since the computational effort of each expansion in BOA*-st is in general cheaper than NAMOA*-dr-st due to the improved dominance checks. More details can be found in [36].

2) Different Maps: We then show how the size of the map affects the low-level planner. Table II (a) shows the minimum, median and maximum $\bar{t}$ over all instances. As the map size increases, both planners need more runtime per call in general. In the map den312d of size 65x81, NAMOA*-dr-st takes up to around 3 seconds per call while BOA*-st needs around 2 seconds. Considering that MO-CBS needs to iteratively invoke the low-level planner, a speed-up in the low-level planner can help with the overall MO-CBS search, which is verified in the resulting success rates of MO-CBS-b and MO-CBS-n in the test with the den312d map: out of the 25 instances, MO-CBS-b succeeds 20 instances while MO-CBS-n succeeds 18 instances.

(a)	min. / median / max. low-Level RT (unit: ms)	
C_{max}	MO-CBS-b	MO-CBS-n
2	1.9 / 2.9 / 6.2	2.0 / 3.8 / 7.9
5	2.1 / 5.0 / 17.7	2.2 / 6.6 / 26.4
8	2.1 / 6.3 / 15.2	2.2 / 8.2 / 32.8

(b)	min. / median / max. low-Level #Exp.	
C_{max}	MO-CBS-b	MO-CBS-n
2	11.0 / 43.1 / 157	9.8 / 41.7 / 154
5	12.3 / 112 / 533	10.5 / 107 / 534
8	13.8 / 158 / 460	12.0 / 151 / 718

TABLE I: Runtime (RT) data of the low-level planner of MO-CBS-b ($i.e.$ BOA*) and MO-CBS-n ($i.e.$ NAMOA*-dr) with $M = 2, N = 4$ (fixed) and varying C_{max} in the empty 16×16 map. Let $\bar{t}$ denote the average runtime (in microseconds) of the low-level planner per call during the MO-CBS search. Table (a) shows the minimum, median and maximum of $\bar{t}$ over all instances. Table (b) shows the same statistics of the number of expansions (#Exp.) of the low-level planner per call. BOA*-st runs up to twice as fast as NAMOA*-dr-st, and the advantage of BOA*-st is more obvious as C_{max} increases.

(a)	min. / median / max. low-Level RT (unit: ms)	
Map	MO-CBS-b	MO-CBS-n
empty 16x16	1.9 / 2.9 / 6.2	2.0 / 3.8 / 7.9
random 32x32	4.4 / 9.8 / 57.7	4.6 / 11.5 / 78.1
den312d 65x81	22.9 / 142.3 / 2267.4	24.7 / 188.7 / 3062.2

(b)	min. / median / max. low-Level #Exp.	
Map	MO-CBS-b	MO-CBS-n
empty 16x16	11.0 / 43.1 / 157	9.8 / 41.7 / 154
random 32x32	10.3 / 197 / 2098	9.0 / 195 / 2195
den312d 65x81	281 / 3925 / 68476	278 / 3916 / 64232

TABLE II: Similarly to Table I, this table reports the runtime (RT) data of the low-level planners when $M = 2, N = 4, C_{max} = 2$ (fixed) in maps of different types and sizes. Table (a) and (b) show the statistics of $\bar{t}$ and of #Exp. respectively over all instances. BOA*-st runs faster than NAMOA*-dr-st in all the maps. In the last den312d map, MO-CBS-n solves 18 instances while MO-CBS-b solves 20 ($i.e.$ two more) instances because of the faster low-level planner BOA*-st.

3) Discussion and Summary: Finally, we report the statistics of the number of root nodes (#Root) of MO-CBS over all instances corresponding to the tests in Table I and II. Note that #Root is the product of the numbers of Pareto-optimal individual paths of each agent, and the number of agents is fixed ($N = 4$) in this experiment. The geometric mean of #Root over agents is an indicator of the number of individual Pareto-optimal paths for each agent. From Table III, as C_{max} increases or the size of the map increases, #Root grows correspondingly, and it indicates that each agent tends to have more individual Pareto-optimal paths. Combined with Table I and II, it shows that, finding more Pareto-optimal paths burdens a low-level planner in general.

To summarize, first, instances with larger C_{max} and larger maps tend to have more Pareto-optimal individual paths, and it takes the low-level planner more time and expansions to find those Pareto-optimal paths in general. Second, BOA*-st

	min. / median / max. #Root
empty 16x16, $C_{max} = 2$	2 / 12 / 108
empty 16x16, $C_{max} = 5$	6 / 150 / 1458
empty 16x16, $C_{max} = 8$	12 / 330 / 2205
random 32x32, $C_{max} = 2$	2 / 30 / 720
den312d 65x81 $C_{max} = 2$	42 / 1920 / 24948

TABLE III: The minimum, median and maximum number of roots (#Root) of the MO-CBS search with $N = 4$ (fixed) and varying C_{max} in various maps. As C_{max} increases or the size of the map increases, #Root grows correspondingly, and it indicates that each agent tends to have more individual Pareto-optimal paths, which burdens the low-level planner.

clearly outperforms NAMOA*-dr-st in terms of the runtime (when $M = 2$). Therefore, for the rest of the experiments when $M = 2$, we limit our focus to MO-CBS-b.

C. MO-CBS High-Level Search

1) Success Rates: We then investigate different high-level search strategies of MO-CBS. We fix $M = 2$ and compare MO-CBS-b (without the tree-by-tree expansion) and MO-CBS-tb (with the tree-by-tree expansion). We test both algorithms in four maps of various sizes with varying N ranging from 2 to 10 with a step size of 2. As shown in Fig. 3, MO-CBS-b slightly outperforms MO-CBS-tb in terms of the success rate in all four maps. An intuitive explanation is that MO-CBS-b generates *all* the root nodes at initialization and inserts them into OPEN for search, which makes the search process more informed. Different from MO-CBS-b, MO-CBS-tb generates the root nodes in a tree-by-tree manner during the search, and greedily search one tree after another, which makes the search process less informed. To verify the reason, we conduct the following comparison.

2) Number of Conflicts and Filtered Nodes: First, we show in Fig. 4 the statistics about the numbers of conflicts (#Conflict) resolved by both algorithms ($i.e.$ count the times when Alg. 1 reaches line 11) over all instances. We can observe that MO-CBS-b in general needs to resolve less conflicts than MO-CBS-tb, which indicates the search process of MO-CBS-b is more efficient than the one of MO-CBS-tb (given that MO-CBS-b has higher success rates).

Second, we look at the statistics about the number of filtered nodes of both algorithms. The number of filtered nodes (#Filter) is defines as the times when the *Filter* procedure returns true (line 6 and 20), which means a candidate node is discarded. In the map random 32x32, as shown in Table. IV, MO-CBS-b tends to filter fewer nodes than MO-CBS-tb. Combined with Fig. 4, we can observe that, with similar success rates, MO-CBS-b resolves less conflicts and filters less nodes than MO-CBS-b does, which indicates that MO-CBS-b can search more efficiently than MO-CBS-tb in general.

3) Memory Issue: Although MO-CBS-b can search more efficiently than MO-CBS-tb in general, MO-CBS-b have to generate all the root nodes for initialization, which can consume a lot of memory. In the den312d map when $N = 6$, MO-CBS-b runs out of the 16GB memory and fails to initialize for some of the instances, while MO-CBS-tb bypasses this

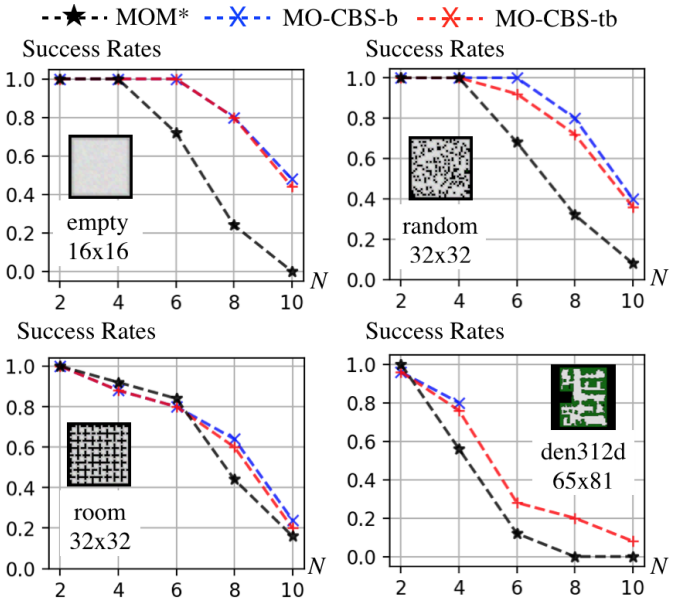


Fig. 3: Success rates of MOM* (baseline), MO-CBS-b (this work) and MO-CBS-tb (this work) in four maps of different sizes. MO-CBS based algorithms outperforms the baseline in general. The maximum enhancement of the success rate (around 60%) can be observed at $N = 8$ in the empty map. In den312d when $N \geq 6$, MO-CBS-b runs out of memory at initialization for some of the instances and is thus omitted.

	min./median/max. #Filter	
N	MO-CBS-b	MO-CBS-tb
2	0 / 2 / 74	0 / 6 / 118
4	0 / 51 / 18052	1 / 111 / 33051
6	3 / 578 / 74390	10 / 1358 / 72175
8	4 / 8881 / 93308	13 / 22050 / 116700
10	0 / 23698 / 122478	1333 / 41222 / 254374

TABLE IV: The minimum, median and maximum of the number of filtered nodes per instance (#Filter) by MO-CBS-b and MO-CBS-tb in the random 32x32 map with varying N. In general, MO-CBS-b filters less number of nodes than MO-CBS-tb. Combined with Fig. 4, with similar success rates, MO-CBS-b resolves less conflicts and filters less nodes than MO-CBS-b does, which indicates that MO-CBS-b can search more efficiently than MO-CBS-tb in general.

memory issue due to the tree-by-tree expansion strategy. As shown in Table V, when $N \geq 6$, the number of roots grows up to millions, which makes MO-CBS-b run out of memory to initialize all the root nodes.

4) Number of Pareto-optimal Solutions: This section reports the statistics of the number of Pareto-optimal solutions (#Sol) and the number of root nodes (#Root) over all succeeded instances ($i.e.$ all Pareto-optimal solutions are found) in the map random 32x32. As shown in Table VI, both the number of root nodes and the number of Pareto-optimal solutions grow as N increases.[7] The discrepancy between the

[7]When $N = 8, 10$, since the success rates are not 100%, there is a bias towards easy instances that have fewer Pareto-optimal solutions. When $N = 2, 4, 6$, since the success rates are 100%, there is no such a bias.

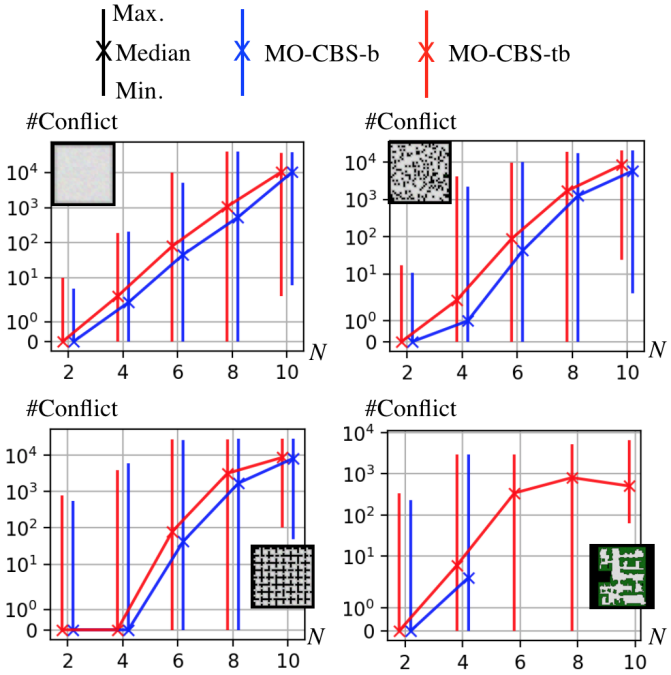


Fig. 4: A comparison of the minimum, median and maximum number of conflicts resolved per instance by MO-CBS-b and MO-CBS-tb in various maps. In general, MO-CBS-tb needs to resolve more conflicts than MO-CBS-b during the search.

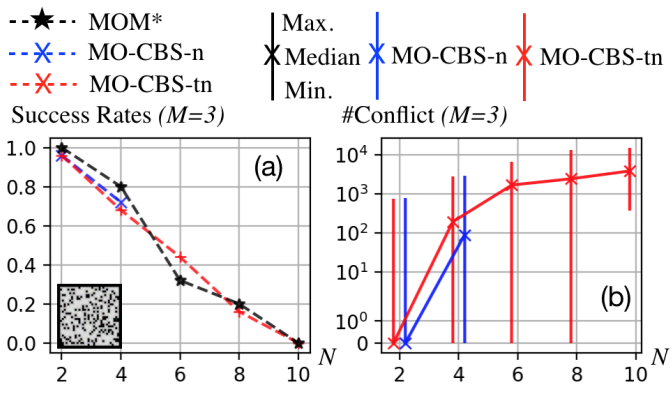
(den312d)	min./median max. #Root				
N	2	4	6	8	10
min.	7	42	504	2,160	11,664
median	36	1,920	39,600	2,566,080	136,080,000
max	216	24,948	2,649,536	558,379,008	17,868,128,256

TABLE V: The statistics of the number of root nodes (#Root) of all instances in the den312d 65x81 map with varying N. As N increases, #Root grows too large for MO-CBS-b to initialize with the 16GB RAM memory, while MO-CBS-tb bypasses this issue due to the tree-by-tree expansion strategy.

#Root and #Sol indicates that a large number of root nodes are filtered instead of leading to Pareto-optimal solutions. It implies a possible future work direction: one can develop new methods for the initialization step of MO-CBS to improve the computational efficiency.

	min./median/max. #Sol and #Root	
N	#Sol	#Root
2	1 / 5 / 14	1 / 6 / 44
4	2 / 9 / 20	2 / 30 / 720
6	5 / 13 / 24	8 / 249 / 5292
8	6 / 16 / 28	10 / 792 / 51840
10	11 / 16 / 29	140 / 804 / 97200

TABLE VI: The minimum, median and maximum of (i) the number of Pareto-optimal solutions (#Sol) and (ii) the number of root nodes (#Root) per instance in MO-CBS-b. The statistics are computed over all succeeded instances ($i.e.$ all Pareto-optimal solutions are found).



(c)	Min. / Median / Max. #Root				
N	2	4	6	8	10
Min.	2	2	80	160	33,600
Median	32	1,740	67,860	2,322,430	25,401,600
Max.	684	89,376	5,898,820	290,304,000	6,584,090,000

Fig. 5: Comparison among MO-CBS-n, MO-CBS-tn, MOM* with $M = 3$ (fixed) and varying N in the random 32x32 map. When M increases from 2 to 3, the problem instances become more challenging and the success rates decrease for all three planners.

D. MO-CBS and MOM*, $M = 2, 3$

1) Two Objectives: We compare MO-CBS-b, MO-CBS-tb and MOM* with $M = 2$ in four different maps. When $M = 2$, as shown Fig. 3, MO-CBS-b and MO-CBS-tb both achieves higher success rates than MOM* in general. The maximum enhancement of the success rate (around 60%) can be observed at $N = 8$ in the empty map. In the room map, MOM* has slightly higher success rates than the MO-CBS based algorithms. In general, it's not obvious under what circumstances MO-CBS is guaranteed to outperform MOM*. Empirically, there is no leading algorithm that outperforms the other method in all settings. More discussion can be found in the following paragraphs.

2) Three Objectives: We also compare MO-CBS-n, MO-CBS-tn and MOM* with $M = 3$ in the random map. As shown in Fig. 5, all three planners achieve similar success rates, which is lower than the corresponding success rates when $M = 2$ in Fig. 3. MO-CBS-n fails to initialize due to the large number of root nodes for some instances when $N \geq 6$ and is thus omitted. In general, when M increases from 2 to 3, the problem becomes more challenging and the success rates decrease for all three planners.

3) Discussion: Since MOM* and MO-CBS are two algorithms that search over different spaces, it's not obvious when one planner is guaranteed to outperform the other. Intuitively speaking, MOM* searches in the joint graph ($i.e.$ the Cartesian product of individual graphs) with a varying branching factor that is determined by the "collision set" [25], the subset of agents that are in conflict. MO-CBS searches in a different space by detecting and splitting conflicts between agents and the number of conflicts is the decisive factor of the computational efficiency of MO-CBS.

Additionally, in MAPF, all edges are often associated with the *same* unit scalar cost [30], [33], while in MOMAPF, all edges are associated with different cost vectors. This makes it hard to predict the difficulty of an instance for MOM* or MO-CBS by only looking at the topology of the map without investigating the cost structure. From our experimental results, one possible indicator about the difficulty of an instance for MO-CBS is the number of root nodes, which reflects the number of individual Pareto-optimal paths of agents, and takes both the topology and the cost structure of the map into consideration.

E. Discussion: MO-CBS and CBS

While (single-objective) CBS can solve up to 21 agents in an empty 8×8 four-connected grid and up to hundred of agents in large maps [30], MO-CBS can solve obviously fewer agents for the following reasons. **First**, the low-level planner of MO-CBS solves a single-agent multi-objective path planning problem, which is computationally more expensive than the single-objective path planning problem solved by the low-level planner of CBS, especially when there are many individual Pareto-optimal paths to find (as discussed in Sec. VIII-B). **Second**, in CBS, each agent has only *one* individual optimal path, and CBS terminates when all the conflicts along those individual paths are resolved. In MO-CBS, each agent has *multiple* individual Pareto-optimal paths, and MO-CBS needs to resolve all the conflicts for *any possible combination* of the individual paths, which is computationally much more expensive (Fig. 4). In other words, the high-level search of MO-CBS needs to search over multiple trees rather than searching a single tree as CBS does (Sec. VIII-C). **Third**, for conventional CBS, larger maps often lead to less conflicts between agents which allows CBS to handle a large number of agents. For MO-CBS, larger maps can lead to a larger number of individual Pareto-optimal paths (Table. II), which then slows down the low-level planner and leads to more potential conflicts to be resolved by MO-CBS.

F. Construction Site Path Planning

This section demonstrates an application example of MO-CBS for practitioners. We consider multiple agents transporting materials in a construction site [12], [29], [31]. We focus on planning collision-free paths for a set of agents from their starts to goals while optimizing both the sum of individual arrival times and the sum of individual path risks. We use a simplified risk model as shown on the left in Fig. 6. We select a random 32x32 map from [33] and compute the corresponding risk map as shown in Fig. 6 as follows. The risk score of each cell equals one plus the number of black cells in the 8 neighbors around it, where the black cells represent some semi-constructed architecture. The risk here is possibly due to the falling items from the architecture or the collision with the architecture. Similar to the previous tests, each agent can either wait or move to one of the four cardinal adjacent cells. Each action of the agent incurs a cost vector of length two, where the first component indicates the action time which is always one, and the second component is the risk cost of the arrival cell as aforementioned. If an agent waits in a cell, the risk cost incurred is the risk score of that cell.

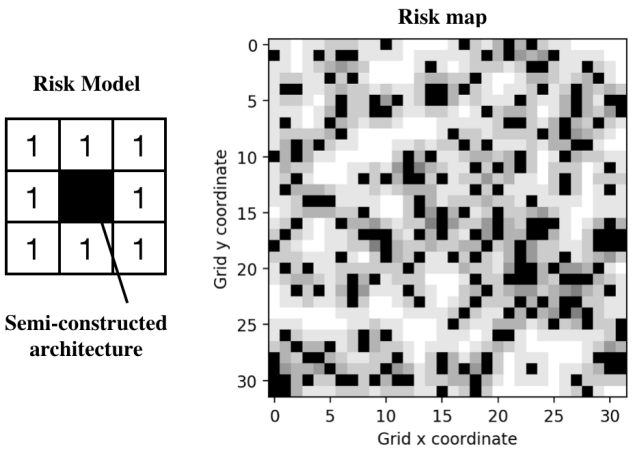


Fig. 6: (Left) Risk model. (Right) A risk map where black cells represent semi-constructed architecture and the darkness of a grey cell indicates the risk score of that cell. More details can be found in the text.

As shown in Fig. 7 (a), the set of Pareto-optimal solutions trades off between the arrival time and path risk. In solution (joint path) S_1 (Fig. 7 (b)), all agents take shortcuts regardless of the risks. For example, the blue agent in S_1 passes through many risky cells by following a shortest path. In solution S_2 (Fig. 7 (c)), all agents follow the safest paths. For example, in the lower right corner of S_2, the light green agent takes a detour to avoid the brown agent to make both of them safe along their respective paths. The solution S_3 (Fig. 7 (d)) visualizes a Pareto-optimal solution in the "middle", where arrival time and path risk are balanced in some way.

IX. Conclusion and Future Work

In this article, we develop a new algorithm called Multi-Objective Conflict-Based Search (MO-CBS) to solve Multi-Objective Multi-Agent Path Finding (MOMAPF) problems with optimality guarantees. We also develop several variants of MO-CBS by using various low-level planners and different high-level expansion strategies. We analyze the properties of MO-CBS and show that the method is able to find all cost-unique Pareto-optimal solutions. Numerical results show that MO-CBS outperforms the baseline in terms of success rates under a runtime limit. We also show an application example for practitioners.

There are several directions for future work. One can consider improving MO-CBS by expediting its initialization, conflict splitting and dominance checks. Additionally, instead of finding an exact set of Pareto-optimal solutions, one can focus on approximating the Pareto-optimal solutions with faster computational speed and better scalability, in terms of both the number of agents and the number of objectives. For example, one can consider leveraging evolutionary algorithms [4] or approximated single-agent multi-objective planners [7] to expedite the computation. One can also develop other types of MOMAPF algorithms by leveraging other (single-objective)

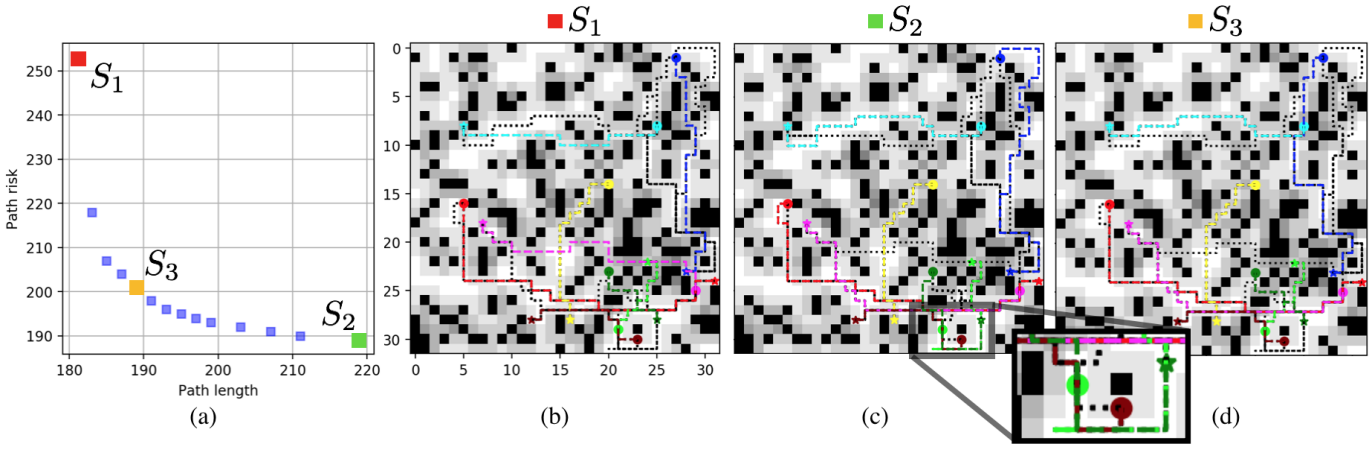


Fig. 7: (a) shows the Pareto-optimal front of the construction site example. (b), (c) and (d) show three Pareto-optimal solution joint paths corresponding to the red, green and orange solution in (a) respectively. In (b), (c) and (d), the colored dotted paths show the individual paths that constitute the corresponding joint path, while the black dotted paths show the individual paths in other Pareto-optimal solutions. For solution S_1, all agents take shortcut and go through risky zones while for solution S_2, all agents are being conservative and go through safe zones. The solution S_3 balances the two objectives. Finding and visualizing a Pareto-optimal set of solutions can potentially help the human decision maker to understand the underlying trade-off between conflicting objectives and thus make more informed decisions.

MAPF methods to handle agents that move with different speeds [1], [21], [39] or targets that need allocation [10], [26].

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 2120219 and 2120529. 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

[1] Anton Andreychuk, Konstantin Yakovlev, Pavel Surynek, Dor Atzmon, and Roni Stern. Multi-agent pathfinding with continuous time. *Artificial Intelligence*, page 103662, 2022.

[2] Liron Cohen, Tansel Uras, TK Satish Kumar, and Sven Koenig. Optimal and bounded-suboptimal multi-agent motion planning. In *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[3] Matthias Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

[4] Michael TM Emmerich and André H Deutz. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural computing*, 17(3):585–609, 2018.

[5] Erhan Erkut, Stevanus A Tjandra, and Vedat Verter. Hazardous materials transportation. *Handbooks in operations research and management science*, 14:539–621, 2007.

[6] Meir Goldenberg, Ariel Felner, Roni Stern, Guni Sharon, Nathan Sturtevant, Robert C Holte, and Jonathan Schaeffer. Enhanced partial expansion A*. *Journal of Artificial Intelligence Research*, 50:141–187, 2014.

[7] Boris Goldin and Oren Salzman. Approximate bi-criteria search by efficient representation of subsets of the pareto-optimal frontier. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 149–158, 2021.

[8] Nir Greshler, Ofir Gordon, Oren Salzman, and Nahum Shimkin. Cooperative multi-agent path finding: Beyond path planning and collision avoidance. In *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 20–28. IEEE, 2021.

[9] Pierre Hansen. Bicriterion path problems. In *Multiple criteria decision making theory and application*, pages 109–127. Springer, 1980.

[10] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph Durham, and Nora Ayanian. Conflict-based search with optimal task assignment. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2018.

[11] Edward Lam, Pierre Le Bodic, Daniel Harabor, and Peter Stuckey. Branch-and-cut-and-price for multi-agent pathfinding. pages 1289–1296, 08 2019.

[12] Edward Lam, Peter J Stuckey, Sven Koenig, and TK Satish Kumar. Exact approaches to the multi-agent collective construction problem. In *International Conference on Principles and Practice of Constraint Programming*, pages 743–758. Springer, 2020.

[13] Hang Ma, Wolfgang Hönig, TK Satish Kumar, Nora Ayanian, and Sven Koenig. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7651–7658, 2019.

[14] Hang Ma and Sven Koenig. Optimal target assignment and path finding for teams of agents. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1144–1152, 2016.

[15] Hang Ma, Jiaoyang Li, T K Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Conference on Autonomous Agents & Multiagent Systems*, 2017.

[16] Sebastian Mai and Sanaz Mostaghim. Modeling pathfinding for swarm robotics. In *International Conference on Swarm Intelligence*, pages 190–202. Springer, 2020.

[17] Lawrence Mandow and José Luis Pérez De La Cruz. Multiobjective A* search with consistent heuristics. *Journal of the ACM (JACM)*, 57(5):1–25, 2008.

[18] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[19] Justin Montoya, Sivakumar Rathinam, and Zachary Wood. Multi-objective departure runway scheduling using dynamic programming. *IEEE Transactions on Intelligent Transportation Systems*, 15(1):399–413, 2013.

[20] Francisco-Javier Pulido, Lawrence Mandow, and José-Luis Pérez-de-la Cruz. Dimensionality reduction in multiobjective shortest path search. *Computers & Operations Research*, 64:60–70, 2015.

[21] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. Loosely synchronized search for multi-agent path finding with asynchronous actions. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2021.

[22] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. MS*: A new exact algorithm for multi-agent simultaneous multi-goal sequencing and path finding. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[23] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. Multi-objective conflict-based search for multi-agent path finding. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[24] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. Multi-objective conflict-based search using safe-interval path planning. *arXiv preprint arXiv:2108.00745*, 2021.

[25] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. Subdimensional expansion for multi-objective multi-agent path finding. *IEEE Robotics and Automation Letters*, 6(4):7153–7160, 2021.

[26] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. Conflict-Based Steiner Search for Multi-Agent Combinatorial Path Finding. In *Proceedings of Robotics: Science and Systems*, New York City, NY, USA, June 2022.

[27] Zhongqiang Ren, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Multi-objective path-based D* lite. *IEEE Robotics and Automation Letters*, 7(2):3318–3325, 2022.

[28] Subrata Saha, Alex Elkjær Vasegaard, Izabela Nielsen, Aneta Hapka, and Henryk Budzisz. Uavs path planning under a bi-objective optimization framework for smart cities. *Electronics*, 10(10):1193, 2021.

[29] Guillaume Sartoretti, Yue Wu, William Paivine, TK Satish Kumar, Sven Koenig, and Howie Choset. Distributed reinforcement learning for multi-robot decentralized collective construction. In *Distributed autonomous robotic systems*, pages 35–49. Springer, 2019.

[30] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[31] AR Soltani and T Fernando. A fuzzy based multi-objective path planning of construction sites. *Automation in construction*, 13(6):717–734, 2004.

[32] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[33] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[34] Bradley S. Stewart and Chelsea C. White. Multiobjective A*. *J. ACM*, 38(4):775814, October 1991.

[35] Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Efficient sat approach to multi-agent path finding under the sum of costs objective. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, pages 810–818, 2016.

[36] Carlos Hernández Ulloa, William Yeoh, Jorge A Baier, Han Zhang, Luis Suazo, and Sven Koenig. A simple and fast bi-objective search algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 143–151, 2020.

[37] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

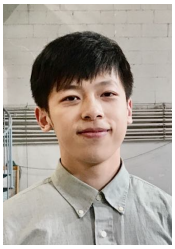
[38] J. Weise, S. Mai, H. Zille, and S. Mostaghim. On the scalable multi-objective multi-agent pathfinding problem. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2020.

[39] Jens Weise and Sanaz Mostaghim. A scalable many-objective pathfinding benchmark suite. *IEEE Transactions on Evolutionary Computation*, 2021.

[40] Jie Xu, Andrew Spielberg, Allan Zhao, Daniela Rus, and Wojciech Matusik. Multi-objective graph heuristic search for terrestrial robot design. 2021.

[41] Jingjin Yu and Steven M LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[42] Jingjin Yu and Daniela Rus. Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *Algorithmic foundations of robotics XI*, pages 729–746. Springer, 2015.



Zhongqiang (Richard) Ren (Student Member, IEEE) received the M.S. degree from Carnegie Mellon University, Pittsburgh, PA, USA, and the dual B.E. degree from Tongji University, Shanghai, China, and FH Aachen University of Applied Sciences, Aachen, Germany. He is currently a Ph.D. candidate at Carnegie Mellon University.

Sivakumar Rathinam (Senior Member, IEEE) received the Ph.D. degree from the University of California at Berkeley in 2007. He is currently a Professor with the Mechanical Engineering Department, Texas A&M University. His research interests include motion planning and control of autonomous vehicles, collaborative decision making, combinatorial optimization, vision-based control, and air traffic control.

Howie Choset (Fellow, IEEE) received the undergraduate degrees in computer science and business from the University of Pennsylvania, Philadelphia, PA, USA, and the M.S. and Ph.D. degrees in mechanical engineering from Caltech, Pasadena, CA, USA.
He is a Professor in the Robotics Institute, Carnegie Mellon, Pittsburgh, PA, USA.