Search Algorithms for Multi-Agent Teamwise Cooperative Path Finding

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

***Abstract*— Multi-Agent Path Finding (MA-PF) computes a set of collision-free paths for multiple agents from their respective starting locations to destinations. This paper considers a generalization of MA-PF called Multi-Agent Teamwise Cooperative Path Finding (MA-TC-PF), where agents are grouped as multiple teams and each team has its own objective to be minimized. For example, an objective can be the sum or max of individual arrival times of the agents. In general, there is more than one team, and MA-TC-PF is thus a multi-objective planning problem with the goal of finding the entire Pareto-optimal front that represents all possible trade-offs among the objectives of the teams. To solve MA-TC-PF, we propose two algorithms TC-CBS and TC-M*, which leverage the existing CBS and M* for conventional MA-PF. We discuss the conditions under which the proposed algorithms are complete and are guaranteed to find the Pareto-optimal front. We present numerical results for several types of MA-TC-PF problems.**

I. Introduction

Multi-Agent Path Finding (MA-PF) seeks to find collision-free paths for multiple agents from their respective start to goal locations, which has been widely studied over the last decade [16]. This problem often requires optimizing a single objective, such as min-sum, i.e., minimizing the sum of individual path costs [14] or min-max, i.e., minimizing the maximum of individual costs of the agents [17]. The objective is typically defined over all the agents and hence the name cooperative path finding [15]. In this paper, we are interested in a variant of MA-PF where agents are grouped into multiple teams, where each team seeks to optimize its own objective (Fig. 1).

We formulate a new problem called Multi-Agent Teamwise Cooperative Path Finding (MA-TC-PF). In MA-TC-PF, each agent has its own start and goal locations, while belonging to at least one team, and teams are not required to be mutually disjoint to each other. Each team has its own objective to be minimized such as min-sum or min-max, and MA-TC-PF seeks to minimize an objective vector, where each component of the vector corresponds to the objective of a team. In the presence of multiple objectives, in general, there does not exist a single solution that can simultaneously minimize all the objectives; therefore, we aim to find a set of Pareto-optimal solutions for the MA-TC-PF. A solution is Pareto-optimal if one cannot improve over one objective without deteriorating another objective. MA-TC-PF differs from the existing Multi-Agent Multi-Objective Path Finding (MA-MO-PF) [11], [19], self-interested MA-PF [2] and adversarial MA-PF [7], and we elaborate in Sec. II.

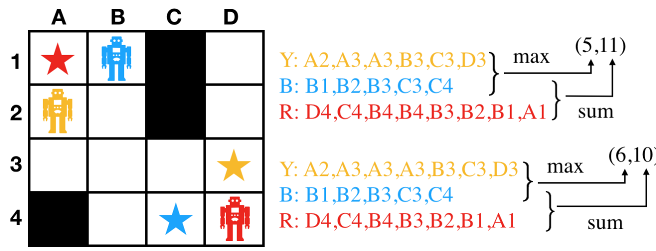


Fig. 1: An illustration of MA-TC-PF with two teams, where team 1 includes the yellow (Y) and blue (B) agents while team 2 includes the blue (B) and red (R) agents. Team 1 aims to minimize the maximum arrival times of both agents (so that they can collaboratively start a task for example), while team 2 aims to minimize the sum of arrival times (since the agents are equipped with some fuel-consuming devices and the total fuel usage is to be minimized for example).

To solve MA-TC-PF, we adapt CBS [14] and M* [18], and propose TC-CBS and TC-M*. On the one hand, TC-CBS and TC-M* leverage the conflict resolution technique in CBS and M* by coupling agents together for planning only when the agents are in conflict. On the other hand, TC-CBS and TC-M* leverage the dominance principles [3] to identify and compare candidate solutions, and are guaranteed to find the entire Pareto-optimal front. We discuss the applicability of each algorithm to different problem variants of MA-TC-PF, and our approaches are tested with up to 20 agents in various maps. Finally, we showcase a possible usage of MA-TC-PF to provide an "explanation" of MA-PF solutions, a notion that arises in explainable and trustworthy AI [1], [9].

II. Related Work

MA-PF [16] often requires optimizing a single-objective, such as min-sum (also called min-flowtime) or min-max (also called min-makespan). It can be regarded as a special case of MA-TC-PF where there is only one team that includes all agents. To solve MA-PF problems to optimality, various methods have been developed, which focus on either min-sum [14], [18] or min-max [17], [20] criteria. The proposed TC-CBS and TC-M* can be used to simultaneously handle the min-sum and min-max bi-objective problems by finding a set of Pareto-optimal solutions.

MA-MO-PF [10], [11], [19] differs from MA-PF by associating a vector-cost (rather than a scalar-cost) to the action of an agent, where each component of the cost vector represents an objective to be minimized, such as arrival time

[1] Zhongqiang Ren, Chaoran Zhang and Howie Choset are at Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA. Emails: {zhongqir, chaoranz, choset}@andrew.cmu.edu

[2] Sivakumar Rathinam is with the Department of Mechanical Engineering, Texas A&M University, College Station, TX 77843-3123. Email: srathinam@tamu.edu

and path risk. MA-MO-PF requires minimizing the sum of accumulated cost vectors over all agents along their paths. The MA-TC-PF differs from MA-MO-PF, since the action cost of each agent is a scalar, and there are multiple teams where each team has its own objective.

Other variants of MA-PF related to this paper include self-interested MA-PF [2], where each agent aims to find its individually min-cost path and the goal is to design a taxation scheme so that all agents become cooperative after adding an additional tax-cost to the agents' paths. In MA-TC-PF, a similar notion of the self-interested agent arises, when each agent itself forms a team. However, the goal of MA-TC-PF is to compute the entire Pareto-optimal front, which identifies possible trade-offs between teams' objectives. Finally, adversarial MA-PF [7] divides agents into mutually disjoint teams, and aims to find a policy for a selected team so that the agents in the selected team can navigate to their goals subject to any actions other teams can take. In contrast, MA-TC-PF does not consider an adversary.

III. Problem Statement

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 $u, v \in V$ is denoted as $(u, v) \in E$ and the cost of $e \in E$ is a finite positive real number $cost(e) \in \mathbb{R}^+$. Let $v_o^i, v_d^i \in V$ respectively denote the start and goal location of agent i.

Let a superscript $i \in I$ over a variable 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 cost value of the path, which is the sum of the cost of all the edges present in the path, i.e., $g^i(\pi^i(v_1^i, v_\ell^i)) = \Sigma_{j=1,2,\dots,\ell-1} cost(v_j^i, v_{j+1}^i)$. For presentation purposes, we denote $\pi^i(v_1^i, v_\ell^i)$ simply as π^i when there is no confusion.

All agents share a global clock and they start the paths at time $t = 0$. Each action of an agent, either wait or move, requires one unit of time. Any two agents 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 $\{T_j, j = 1, 2, \dots, M\}$ denote a set of M *teams*, where each team $T_j \subseteq I$. Each agent belongs to at least one team and teams are not required to be mutually disjoint to each other. Let π^{T_j} denote a joint path, which is a set of individual paths $\{\pi^i, \forall i \in T_j\}$. Let g^{T_j} denote the *objective* value of team T_j that is to be minimized, which is either the sum or the maximum of the individual path cost of all the agents in the team T_j (i.e., $g^{T_j} := \sum_{i \in T_j} g(\pi^i)$ or $g^{T_j} := \max_{i \in T_j} g(\pi^i)$). Let π (without any superscript) denote a joint path of all the agents, which is also referred to as a solution. Let $\vec{g}(\pi) := \{g(\pi^{T_j}), j = 1, 2, \dots, M\}$ denote an *objective vector* of length M, where each component corresponds to the objective of a team.

To compare two solutions, we compare the objective 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 [3]): 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 $a(m) < b(m)$, $\exists m \in \{1, 2, \dots, M\}$.

Any two solutions are non-dominated with respect to each other if the corresponding objective 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, and the corresponding set of objective vectors is called the Pareto-optimal front. In this paper, 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 objective vector.

Remark 1: A MA-TC-PF problem is called *fully cooperative* if each team $T_j, j = 1, 2 \dots, M$ contains all agents (i.e., $T_j = I$). Otherwise (i.e., there exists a team that does not include all agents), the MA-TC-PF is not fully cooperative.

IV. TC-CBS

We first review Conflict-Based Search (CBS) [14] and then describe our method Teamwise Cooperative Conflict-Based Search (TC-CBS). We then discuss the relationship of TC-CBS to Multi-Objective CBS (MO-CBS) [11] and discuss the cases where TC-CBS is incomplete.

A. Review of Conflict-Based Search

Conflict-Based Search (CBS) [14] is a two-level search algorithm that computes a conflict-free solution to a MA-PF problem. On the high-level, every search node P is defined as a tuple of (π, g, Ω), where:

- $\pi = (\pi^1, \pi^2, \dots, \pi^N)$ is a joint path that connects starts and goals of agents respectively.
- g is the scalar cost value of π (i.e., $g = g(\pi) = \Sigma_{i \in I} g^i(\pi^i)$).
- Ω is a set of constraints. Each constraint is of form (i, v, t) (or i, e, t), which indicates agent i is forbidden to enter node v (or edge e) at time t.

CBS constructs a search tree with the root node $P_{root} = (\pi_o, g(\pi_o), \emptyset)$, where the joint path π_o is constructed by running the low-level (single-agent) planner, such as A*, for every agent respectively with an empty set of constraints while ignoring any other agents. P_{root} is added to OPEN, a queue that prioritizes nodes based on their g-values.

In each search iteration, a node $P = (\pi, g, \Omega)$ with the minimum g-value is popped from OPEN for expansion. To expand P, every pair of individual paths in π is checked for vertex conflict (i, j, v, t) (and edge conflict (i, j, e, t)). If no conflict is detected, π is conflict-free and is returned as an optimal solution. Otherwise, the detected conflict (i, j, v, t) is *split* into two constraints (i, v, t) and (j, v, t) respectively and two new constraint sets $\Omega \bigcup \{i, v, t\}$ and $\Omega \bigcup \{j, v, t\}$ are generated. (Edge conflict is handled in a similar manner and is thus omitted.) Then, for the agent i in each split constraint (i, v, t) and the corresponding newly generated constraint set $\Omega' = \Omega \bigcup \{i, v, t\}$, the low-level planner is invoked to plan an individual minimum cost path π'^i of agent i subject to all constraints related to agent i in Ω'. The low-level planner typically runs A*-like search in a time-augmented graph with constraints marked as obstacles. A new joint path π' is then formed by first copying π and then updating agent i's individual path π^i with π'^i. Finally, for each of the two split constraints, a corresponding high-level node is generated and added to OPEN for future expansion. CBS terminates when the first conflict-free joint path is found which is guaranteed to be the min-cost solution.

B. TC-CBS Algorithm

As shown in Alg. 1, the proposed TC-CBS algorithm follows a similar workflow as CBS. The main differences are the following. **First**, given a high-level node P_k and its corresponding joint path π_k, TC-CBS computes an objective vector $\vec{g}(\pi_k)$ based on the teams, instead of computing a scalar cost value g as in CBS. This arises in lines 1 and 14, when generating the root node and a new high-level node respectively. Consequently, high-level nodes are organized in lexicographic (abbreviated as lex.) order in OPEN, and in each iteration, a lex. min node is popped from OPEN for processing (line 4). **Second**, since there are multiple Pareto-optimal solutions in general, TC-CBS stores all Pareto-optimal solutions found during the search in a set $\mathcal{C}$ (line 7). For presentation purposes, we denote $\mathcal{C}$ as a set of objective vectors. Each vector in $\mathcal{C}$ identifies a unique high-level node and thus a unique conflict-free solution. **Third**, to find all cost-unique Pareto-optimal solutions, TC-CBS terminates when OPEN depletes, while CBS terminates when the first conflict-free solution is found. Additionally, whenever a node P_k is popped from OPEN (line 4) or newly generated (line 15), P_k is tested for filtering, i.e., P_k is discarded if the objective vector in P_k is dominated by or equal to any existing objective vectors in $\mathcal{C}$.

C. Discussion and Properties of TC-CBS

The high-level search in TC-CBS is the same as MO-CBS [11], while the low-level search is different. In MO-CBS, each low-level search requires solving a multi-objective shortest path problem subject to constraints [6], [12], [13], while the low-level search in TC-CBS is single-objective.

A problem instance is feasible if there exists a feasible solution. Given a feasible instance, TC-CBS is said to be *complete* if it terminates in finite time. For fully cooperative MA-TC-PF, TC-CBS is guaranteed to be complete, and is guaranteed to find all cost-unique Pareto-optimal solutions. The analysis in MO-CBS [11] can be applied to TC-CBS for fully cooperative MA-TC-PF problems, since TC-CBS has the same high-level search as MO-CBS.

Algorithm 1 Pseudocode for TC-CBS

```
1: Compute P_root and insert into OPEN.
2: C ← ∅
3: while OPEN not empty do
4:     P_k = (π_k, g⃗_k, Ω_k) ← OPEN.pop()
5:     if Filter(P_k) then continue        ▷ End of iteration
6:     if no conflict detected in π_k then
7:         add g⃗_k to C
8:         continue                        ▷ End of iteration
9:     Ω ← split detected conflict
10:    for all ω^i ∈ Ω do
11:        Ω_l = Ω_k ∪ {ω^i}
12:        π^i_* ← LowLevelSearch(i, Ω_l)
13:        π_l ← π_k, replace π^i_l (in π_l) with π^i_*
14:        g⃗_l ← g⃗(π_l)         ▷ Computed based on teams.
15:        P_l = (π_l, g⃗_l, Ω_l)
16:        if not Filter(P_l) then
17:            add P_l to OPEN
18: return C
```

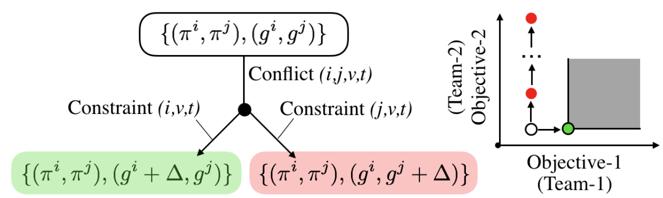


Fig. 2: An illustrative case where TC-CBS is incomplete. The grey area show the set of objective vectors dominated by the green solution. Details can be found in the text.

However, for MA-TC-PF that is not fully cooperative, TC-CBS may not be complete (i.e., incomplete): TC-CBS fails to terminate in finite time even if the problem instance is feasible. Same as discussed in [11], the condition for TC-CBS being complete is: there is a finite number of joint paths whose objective vectors are non-dominated by the Pareto-optimal front. This condition may not hold for MA-TC-PF that is not fully cooperative. We illustrate with an example as shown in Fig. 2: there are two agents $I = \{i, j\}$ and two teams $T_1 = \{i\}, T_2 = \{j\}$; the objective vector is $(g^{T_1}, g^{T_2}) = (g^i, g^j)$. Consider the case where a conflict $(i = 1, j = 2, v, t)$ is detected, and is split into constraints (i, v, t) and (j, v, t) during the search, which results in two new high-level nodes (red and green). For either of the two nodes, one agent's path cost may increase (as a constraint is added), while the other agent's path cost remains the same. Consider the case where the green node leads to the only conflict-free Pareto-optimal solution, and the red node still contains conflicts and leads to further conflict resolution.

Algorithm 2 Pseudocode for TC-M*

1: initialize OPEN with $l_o = (v_o, \vec{h}(v_o))$
2: $\mathcal{L} \leftarrow \emptyset$, $I_C(l_o) \leftarrow \emptyset$, $\alpha(v_o) \leftarrow \{l_o\}$
3: **while** OPEN not empty **do**
4:     $l \leftarrow$ OPEN.pop()
5:     **if** *SolutionFilter*(l) **then continue**
6:     **if** $v(l) = v_d$ **then**
7:         add l to $\mathcal{L}$ and then **continue**
8:     $Ngh(l) \leftarrow$ *GetLimitedNgh*(l)
9:     **for all** $l' \in Ngh(l)$ **do**
10:         $I_C(l') \leftarrow I_C(l') \bigcup \Psi(v(l), v(l'))$
11:         *BackProp*(l, $I_C(l')$)
12:         **if** $\Psi(v(l), v(l')) \neq \emptyset$ **then continue**
13:         **if** *DomCheck*(l') **then**
14:             *DomBackProp*(l, l')
15:             **continue**
16:         $\vec{f}(l') \leftarrow \vec{g}(l') + \vec{h}(v(l'))$
17:         add l' to OPEN, add l' to $\alpha(v(l'))$
18:         add l to back_set(l'), parent(l') $\leftarrow l$
19: **return** $\mathcal{L}$

Algorithm 3 Pseudocode for BackProp

INPUT: l, $I_C(l')$
if $I_C(l') \nsubseteq I_C(l)$ **then**
    $I_C(l) \leftarrow I_C(l') \bigcup I_C(l)$
    if $l \notin$ OPEN **then**
        add l to OPEN
    for all $l'' \in$ back_set(l) **do**
        BackProp(l'', $I_C(l)$)

As a result, there can be an infinite number of joint paths[1] whose objective vectors are non-dominated by the Pareto-optimal front, and TC-CBS never terminates since OPEN never depletes.

V. TC-M*

In contrast to TC-CBS, the proposed TC-M* in this section is complete for all variants of MA-TC-PF. We begin with a full description of TC-M*, and then discuss its properties and the relationship to the existing M* [18] and MOM* [10].

A. Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}) = G \times G \times \cdots \times G$ denote the joint graph which is the Cartesian product of N copies of G, where each vertex $v \in \mathcal{V}$ represents a joint vertex, and $e \in \mathcal{E}$ represents a joint edge that connects a pair of joint vertices. The joint vertices corresponding to the start and goal vertices of all the agents are $v_o = (v_o^1, v_o^2, \cdots, v_o^N)$ and $v_d = (v_d^1, v_d^2, \cdots, v_d^N)$ respectively.

[1] An example is that agent i has reached its destination which blocks the only path for agent j to reach its destination v_d^j. In this case, an infinite number of high-level nodes will be generated. It remains an open question whether we can design a mechanism to detect all the corner cases and make TC-CBS complete for all variants of MA-TC-PF.

Algorithm 4 Pseudocode for DomBackProp

INPUT: l, l' $\triangleright$ l' is a successor of l
for all $l'' \in \alpha(v(l'))$ **do**
    if $\vec{g}(l'') \succeq \vec{g}(l')$ or $\vec{g}(l'') = \vec{g}(l')$ **then**
        BackProp(l, $I_C(l'')$)
        add l to back_set(l'')

There can be multiple non-dominated joint paths from v_o to any other joint vertex v in $\mathcal{G}$. To distinguish these paths, let $l := (v, \vec{g})$ denote a *label*, which is a tuple of a joint vertex v and an objective vector $\vec{g}$. Each label identifies a unique joint path $\pi(v_o, v)$ from v_o to v with objective vector $\vec{g} = \vec{g}(\pi(v_o, v))$. To simplify notations, let $v(l), \vec{g}(l)$ denote the joint vertex and the objective vector related to label l, and let $v^i(l)$ denote the vertex of agent i contained in $v(l)$. To keep track of multiple joint paths at each joint vertex v, let $\alpha(v)$ denote a set of labels l with $v(l) = v$.

Similarly to A* [5], let heuristic vector $\vec{h}(v)$ denote an underestimate of the cost-to-go from joint vertex v, which is an M-dimensional vector, and define f-vector as $\vec{f}(l) := \vec{g}(l) + \vec{h}(v(l))$. Let OPEN denote a list of candidate labels to be expanded during the search, where labels are prioritized in the lex. order based on their f-vectors.

Additionally, let $\phi^i : V \to V$ denote an individual optimal policy, which maps the current vertex of an agent to the next vertex along some individual optimal path towards its goal. ϕ^i can be constructed via a pre-processing step, where the shortest paths from any vertex in G to v_d^i for each agent $i \in I$ are found via an exhaustive backwards A* search from v_d^i to any other vertices in G. Finally, let $\Psi : \mathcal{V} \times \mathcal{V} \to 2^I$ (2^I stands for the power set of I) denote a conflict checking function, which takes two adjacent joint vertices $u, v \in \mathcal{G}$ and returns a subset of agents that are in conflict when transiting from u to v. Let $I_C(l) \subseteq I$ denote a collision set of label l. Intuitively, it stores the subset of agents that can run into conflicts during the search process.

B. TC-M* Algorithm

Intuitively, TC-M* begins by searching a sub-graph embedded in $\mathcal{G}$ by letting agents follow their individual policies, and dynamically growing the sub-graph based on agent-agent conflicts (i.e., collision sets I_C) until all cost-unique conflict-free Pareto-optimal joint paths from v_o to v_d are found.

Specifically, as shown in Alg. 2, TC-M* first adds the initial label $l_o := (v_o, \vec{h}(v_o))$ into OPEN and initializes $\mathcal{L}$ to be an empty set, which will be used to store labels that identify cost-unique Pareto-optimal solutions found during the search. Additionally, at any time during the search, the collision set of a label that is newly generated is initialized to be an empty set.

In each iteration (lines 4-18), a label l with the lex. min f-vector in OPEN is popped and processed as follows. First, procedure *SolutionFilter* discards l (line 5), if $\vec{f}(l)$ is dominated by or equal to the f-vector of any existing solutions in $\mathcal{L}$ (i.e., there exists $l^* \in \mathcal{L}$ such that $\vec{f}(l^*) \succeq \vec{f}(l)$ or $\vec{f}(l^*) = \vec{f}(l)$, and note that $\vec{f}(l^*) = \vec{g}(l^*)$ since

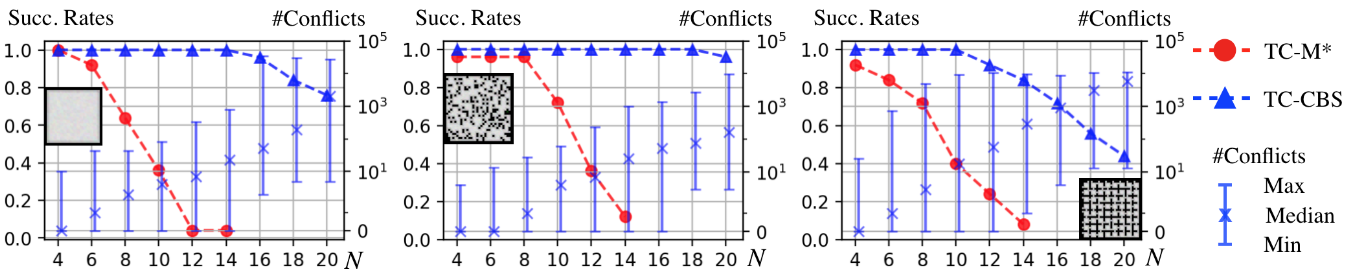


Fig. 3: Numerical results of our algorithms TC-CBS and TC-M* for fully cooperative MA-TC-PF problems with min-sum and min-max as the two objectives (type-1 problem). The horizontal axis shows the number of agents (N), the left vertical axis shows the success rates (Succ. Rates) while the right axis shows the number of conflicts resolved (#Conflicts). Three maps are of size 16x16 empty, 32x32 random and 32x32 room from the left to the right. TC-CBS outperforms TC-M* and can address up to 20 agents.

$h(v(l^*)) = h(v_d) = 0$). If l is not filtered, the algorithm checks if $v(l) = v_d$. If yes, a new cost-unique Pareto-optimal solution is found, the label l is thus added to $\mathcal{L}$ and the current iteration ends. If $v(l) \neq v_d$, l is then expanded by generating its "limited neighbor" set [18] as follows.

The limited neighbors $Ngh(l)$ is a set of successor labels of l (line 8). For each agent i, if $i \notin I_C(l)$, agent i is only allowed to follow its individual policy $\phi^i(v^i(l))$. If $i \in I_C(l)$, agent i is allowed to visit any adjacent vertices of $v^i(l)$ in G. Formally,

$$v^i(l') \leftarrow \begin{cases} \phi^i(v^i(l)) & \text{if } i \notin I_C(l) \\ v^i(l') \mid (v^i(l), v^i(l')) \in E & \text{if } i \in I_C(l) \end{cases} \quad (1)$$

Limited neighbors of a label l varies once $I_C(l)$ changes, which dynamically modifies the sub-graph embedded in $\mathcal{G}$ that can be reached from l.

After generating $Ngh(l)$, TC-M* loops over each of the labels $l' \in Ngh(l)$ (lines 10-18). Collision checking is conducted for the transition from $v(l)$ to $v(l')$, which returns a set of agents that are in conflict (line 10), and is unioned with the current collision set $I_C(l')$. If l' has never been generated before, $I_C(l')$ is first initialized to be an empty set before the union operation.

Then (line 11), the collision set of label l' is back-propagated via the *Backprop* procedure as shown in Alg. 3. To support the collision set back-propagation, a data structure "back_set" is defined for every label. Intuitively, the back_set of label l contains all parent labels from which l is ever reached during the search. $I_C(l')$ is used to update the collision set of all parent labels recursively (lines 2-7 in Alg. 3), and labels, whose collision sets are enlarged, are re-inserted into OPEN for re-expansion (line 5 in Alg. 3).

After back-propagating the collision set, if there is no conflict during the transition from $v(l)$ to $v(l')$, label l' is checked for dominance in procedure *DomCheck* (line 13). Specifically, *DomCheck* returns true if there exists an objective vector $\vec{g}(l'')$ of an existing label $l'' \in \alpha(v(l'))$ that dominates or is equal to $\vec{g}(l')$. If *DomCheck* returns true, label l' can not lead to a cost-unique Pareto-optimal solution is thus pruned (line 15). Before being pruned (line 14), another procedure *DomBackProp* is invoked over label l' and its parent l so that the collision sets of ancestor labels of l' can still be updated after l' is pruned. If *DomCheck* returns false, label l' is added to $\alpha(v(l'))$ and OPEN for future expansion (lines 16-18). When the algorithm terminates, the set of solution labels $\mathcal{L}$ is returned.

C. Discussion and Properties of TC-M*

Similarly to M* [18], TC-M* leverages the notion of individual policies, collision sets and back-propagation. Additionally, TC-M* borrows the technique of handling multiple non-dominated joint paths from v_o to any other joint vertex as in MOM* [10], which includes the dominance comparison, *SolutionFilter* and *DomBackProp*.

In contrast to TC-CBS, TC-M* is complete for all variants of MA-TC-PF and is guaranteed to find all cost-unique Pareto-optimal solutions. Intuitively, TC-M* searches the joint graph $\mathcal{G}$ (which has a finite size) by first exploring a low-dimensional sub-graph and iteratively enlarging the sub-graph being searched. In the worst case, TC-M* runs A*-like (or Multi-Objective A*-like) search over the entire $\mathcal{G}$ and will terminate when $\mathcal{G}$ is exhaustively searched. We refer the reader to [10], [18] for more details.

VI. Numerical Results

A. Test Settings

We implemented our TC-CBS and TC-M* in Python and tested on a laptop with Core i7-11800H 2.40GHz CPU and 16 GB RAM. A possible baseline approach that can solve MA-TC-PF with solution quality guarantees is to run MOA* search [6], [13] directly in the joint graph $\mathcal{G}$. However, the size of $\mathcal{G}$ grows exponentially with respect to the number of agents, which limits the scalability of this baseline approach [10], [11], [18]. We thus omit this baseline.

We leverage an online dataset for MA-PF [16], which contains grid-like maps and test instances (i.e., pairs of v_o and v_d). We set a runtime limit of five minutes for each instance. We test the following four types of problem instances with the number of agents N ranging from 4 to 20. In each map, there are 25 instances for each N. The type-1 problem has two teams and each team includes all the agents. One team has the min-sum objective while the other

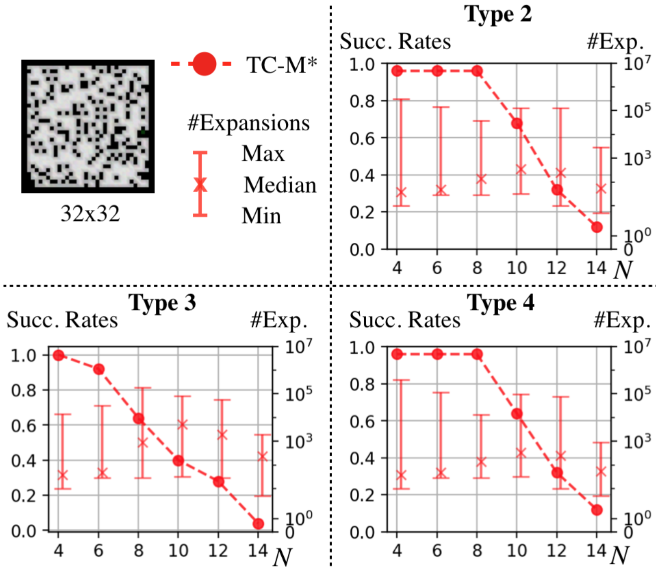


Fig. 4: Numerical results of our algorithm TC-M* for different types of MA-TC-PF problems. The horizontal axis shows the number of agents (N), the left vertical axis shows the success rates (Succ. Rates) while the right axis shows the number of expansions (#Exp.). TC-M* can handle up to ten agents in general.

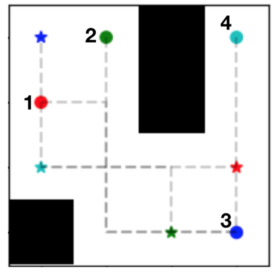
team has the min-max objective. Type-2 divides all agents into two disjoint teams of equal size, and both teams have the min-sum objective. Type-3 divides agents into disjoint teams, where each team contains two agents and has the min-max objective. Type-4 treats each agent as a team.

B. *MA-PF with Both Min-sum and Min-max Objectives*

We begin with the type-1 problem, which can be solved by both TC-M* and TC-CBS. As shown in Fig. 3, TC-CBS achieves higher success rates than TC-M*, and is tested with up to 20 agents. Although TC-CBS is incomplete for general MA-TC-PF problems, it is computationally more efficient than TC-M* in general. We report the corresponding statistics of the number of Pareto-optimal solutions over succeeded instances here: for all three maps and all Ns that are tested, the minimum and median number of solutions is one, and the maximum number of solutions is up to three. It indicates that, in these instances, the min-sum and min-max objectives can often be optimized at the same time.

C. *Other Variants of MA-TC-PF*

We then investigate problems of type-2,3,4, which can be handled by TC-M* with completeness guarantees. As shown in Fig. 4, TC-M* can in general handle up to 10 agents for these problems. For type-4 problem where each agent is a team, we further provide an example as shown in Fig. 5 with four agents. In this example, there are eight Pareto-optimal solutions, which identifies all possible trade-offs between all agents. It can be easily proved (by contradiction) that this set of solutions contains both the min-sum solution and the min-max solution of all the agents.



Agents	1	2	3	4	Sum	Max
Solution 1	4	6	6	7	23	7
Solution 2	6	6	6	6	24	**6**
Solution 3	4	5	7	7	23	7
Solution 4	6	7	6	5	24	7
Solution 5	6	5	7	6	24	7
Solution 6	6	5	8	5	24	8
Solution 7	7	4	6	5	**22**	7
Solution 8	6	5	6	7	24	7
Idvl. Optm.	4	4	6	5	19	6

Fig. 5: An example for type 4 problem, where each agent (circle) needs to move to its destination (star), and the goal is to find all trade-offs between agents. In this example, Solution 7 (highlighted in the table in blue) has the minimum sum of individual arrival times. This table allows us to answer explanatory questions about the solutions. More discussion can be found in the text.

D. *Example: Explanation for MA-PF Solutions*

Finally, MA-TC-PF has the potential to answer explanatory questions about MA-PF solutions. For example, regarding the instance shown in Fig. 5, consider a possible question raised by the user of MA-PF planners: among all conflict-free solutions, can agent 1's arrival time be reduced without worsening the min-sum objective of all agents? The table computed by our approach can provide the answer to the question (which is NO in this case). Answering explanatory questions may increase trust of users and transparency of intelligent systems [1], [9].

VII. Conclusion and Future Work

This paper proposes a new problem formulation MA-TC-PF, which generalizes the conventional MA-PF from one team to multiple teams, and is able to describe the min-sum and min-max bi-objective MA-PF problem. To solve MA-TC-PF, while relying heavily on existing algorithms, we develop two algorithms TC-CBS and TC-M*, and discuss their properties. We present and discuss the numerical results of the proposed algorithms for several different types of MA-TC-PF problems in various maps. Finally, we showcase a possible usage of MA-TC-PF.

There are several directions for future work. One can investigate if the existing improving techniques (e.g. [4], [8], [21]) can be leveraged to improve the scalability of TC-CBS and TC-M*. One can also investigate variants of MA-TC-PF and design algorithms for trustworthy and explainable AI.

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] Shaull Almagor and Morteza Lahijanian. Explainable multi agent path finding. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 34–42, 2020.

[2] Zahy Bnaya, Roni Stern, Ariel Felner, Roie Zivan, and Steven Okamoto. Multi-agent path finding for self interested agents. In *International Symposium on Combinatorial Search*, volume 4, 2013.

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

[4] Cornelia Ferner, Glenn Wagner, and Howie Choset. Odrm* optimal multirobot path planning in low dimensional search spaces. In *2013 IEEE International Conference on Robotics and Automation*, pages 3854–3859. IEEE, 2013.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[6] Carlos Hernández, William Yeoh, Jorge A. Baier, Han Zhang, Luis Suazo, Sven Koenig, and Oren Salzman. Simple and efficient bi-objective search algorithms via fast dominance checks. *Artif. Intell.*, 314:103807, 2023.

[7] Marika Ivanová and Pavel Surynek. Adversarial multi-agent path finding is intractable. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 481–486. IEEE, 2021.

[8] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, Hang Ma, Graeme Gange, and Sven Koenig. Pairwise symmetry reasoning for multi-agent path finding search. *Artificial Intelligence*, 301:103574, 2021.

[9] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019.

[10] 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.

[11] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. A conflict-based search framework for multiobjective multiagent path finding. *IEEE Transactions on Automation Science and Engineering*, pages 1–13, 2022.

[12] Zhongqiang Ren, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Multi-objective safe-interval path planning with dynamic obstacles. *IEEE Robotics and Automation Letters*, 7(3):8154–8161, 2022.

[13] Zhongqiang Ren, Richard Zhan, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Enhanced multi-objective A* using balanced binary search trees. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, pages 162–170, 2022.

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

[15] David Silver. Cooperative pathfinding. In *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, volume 1, pages 117–122, 2005.

[16] 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.

[17] Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Modifying optimal sat-based approach to multi-agent path-finding problem to suboptimal variants. 07 2017.

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

[19] 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.

[20] Jingjin Yu and Steven M LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.

[21] Han Zhang, Jiaoyang Li, Pavel Surynek, T. K. Satish Kumar, and Sven Koenig. Multi-agent path finding with mutex propagation. *Artificial Intelligence*, 311:103766, 2022.