Optimal and Bounded-Suboptimal Multi-Goal Task Assignment and Path Finding

Xinyi Zhong[1], Jiaoyang Li[2], Sven Koenig[2], Hang Ma[1]

Abstract— **We formalize and study the multi-goal task assignment and path finding (MG-TAPF) problem from theoretical and algorithmic perspectives. The MG-TAPF problem is to compute an assignment of tasks to agents, where each task consists of a sequence of goal locations, and collision-free paths for the agents that visit all goal locations of their assigned tasks in sequence. Theoretically, we prove that the MG-TAPF problem is NP-hard to solve optimally. We present algorithms that build upon algorithmic techniques for the multi-agent path finding problem and solve the MG-TAPF problem optimally and bounded-suboptimally. We experimentally compare these algorithms on a variety of different benchmark domains.**

I. Introduction

In recent years, the multi-agent path finding (MAPF) problem [1] has been well-studied in artificial intelligence and robotics due to its many applications, such as warehouse automation [2], autonomous traffic management [3], autonomous aircraft towing [4], and video games [5]. In the MAPF problem, each agent must move from its current location to its pre-assigned goal location while avoiding collisions with other agents in a known environment.

The MAPF problem has recently been extended to many real-world settings [6], [7] where goal locations are not pre-assigned to agents. For example, in a modern automated warehouse, each warehouse robot needs to pick up an inventory pod from its storage location, deliver it to the inventory stations that request one or more products stored in it, and take it back to its storage location. Such automated warehouse systems often employ a task planner to determine a set of tasks consisting of a sequence of goal locations. The problem is then to assign these tasks to the warehouse robots and plan paths for them.

We thus formalize and study the multi-goal task assignment and path finding (MG-TAPF) problem, where as many tasks as agents are given and each task consists of a sequence of goal locations. The MG-TAPF problem is to compute a one-to-one assignment of tasks to agents and plan collision-free paths for the agents from their current locations to the goal locations of their assigned tasks such that each agent visits the goal locations in the order specified by its assigned task and the flowtime (the sum of the finish times of all agents in the last goal locations of their assigned tasks) is minimized.

[1]School of Computing Science, Simon Fraser University, Canada {xinyi_zhong,hangma}@sfu.ca

[2]Department of Computer Science, University of Southern California, USA {jiaoyanl,skoenig}@usc.edu

A. Background and Related Work

Many problems that are related to our problem have been proposed and studied in recent years.

MAPF: The MAPF problem is NP-hard to solve optimally for flowtime (the sum of the finish times of all agents in the last goal locations of their assigned tasks) minimization [8] and even NP-hard to approximate within any constant factor less than 4/3 for makespan (the maximum of the finish times of all agents in their pre-assigned goal locations) minimization [9], [10]. MAPF algorithms include reductions to other well-studied optimization problems [11]–[13] and specialized rule-based, search-based and hybrid algorithms [14]–[20]. In particular, Conflict-Based Search (CBS) [19] is a popular two-level optimal MAPF algorithm that computes time-optimal paths for individual agents on the low level and performs a best-first tree search to resolve collisions among the paths on the high level. Recent research has developed several improved versions of CBS [21]–[24]. An extended version of CBS has been developed for the case where each agent has multiple pre-assigned goal locations [25]. However, the MAPF problem is insufficient for modelling real-world settings where goal locations are not pre-assigned to agents.

TAPF: In the TAPF problem [26], agents are partitioned into teams and each team is given the same number of single-goal tasks as the number of agents. The objective is to assign each agent of a team exactly one task of the team and plan collision-free paths for the agents. A special case of the TAPF problem occurs when only one team exists, known as the anonymous MAPF problem. We use TAPF to denote this special case. TAPF can be solved optimally in polynomial time for makespan minimization [27]. For flowtime minimization, its complexity remains unclear, but it can be solved with an extension of CBS, called CBS with Task Assignment (CBS-TA) [28]. CBS-TA searches in the space of all possible assignments of tasks to agents using a best-first search in a forest that consists of regular CBS search trees where each tree corresponds to a different assignment. A similar version of CBS has been developed for the case where each agent can execute more than one task, but it scales to only four agents and four tasks [29].

Lifelong TAPF: Recent research has considered the online multi-agent pickup and delivery (MAPD) problem, where tasks appear at unknown times and each task consists of a sequence of two goal locations [30]. The offline MAPD problem has also been studied for the case where all tasks are known in advance [7]. The multi-goal TAPF (MG-TAPF) problem is at the crux of two lifelong problems since state-of-the-art MAPD algorithms [7], [31] essentially decompose a lifelong instance into a sequence of MG-TAPF instances. However, the MG-TAPF instances are not solved optimally in either case. Grenouilleau et al. [31] have proposed the Multi-Label Space-Time A* algorithm (MLA*), that computes a time-optimal path for an agent that visits two goal locations and solves task assignment and path finding independently. A similar lifelong problem has also been considered where each task has a temporal constraint [6].

PC-TAPF: The precedence-constraint TAPF (PC-TAPF) problem generalizes the anonymous MAPF problem [32]. It involves sequential task assignment and incorporates temporal precedence constraints between tasks, where for example, tasks A and B must be completed before task C begins. The PC-TAPF problem is NP-hard to solve for makespan minimization. A four-level hierarchical algorithm is proposed to solve it optimally. However, the lower level path planner (third and fourth level) is incomplete. So, there is no completeness guarantee for the whole algorithm.

B. Contributions

In this paper, we study the general version of the TAPF problem, where each task consists of a sequence of multiple ordered goal locations, called MG-TAPF problem.

We formalize the MG-TAPF problem as an extension of the TAPF problem that aims to minimize the flowtime. We prove that it is NP-hard to solve optimally. The proof is based on a reduction [10] from a specialized version of the Boolean satisfiability problem [33] to the MG-TAPF problem.

We present an Conflict-Based Search with Task Assignment with Multi-Label A* algorithm (CBS-TA-MLA) that solves the MG-TAPF problem optimally for flowtime minimization. CBS-TA-MLA is a hierarchical algorithm. It uses a best-first search algorithm CBS-TA [28] on the high level to search over all possible assignments of tasks to agents and resolve collisions among paths and MLA* [31] on the low level to compute a time-optimal path for each agent that visits the goal locations of its assigned task in sequence. We prove that CBS-TA-MLA is correct, complete and optimal.

We develop three admissible heuristics for the high-level search of CBS-TA-MLA based on the existing admissible heuristics [22] for CBS for the MAPF problem and generalize Multi-Valued Decision Diagrams (MDDs) from the case of one goal location to the case of multiple goal locations. We also extend CBS-TA-MLA to a bounded-suboptimal version, called ECBS-TA-MLA, using ideas from the bounded-suboptimal version of CBS [34]. We experimentally compare the proposed algorithms for a variety of benchmark domains.

II. Problem Definition

The MG-TAPF problem instance consists of (1) an undirected graph $G = (V, E)$, where V is the set of locations and E is the set of unit-weight edges connecting locations, (2) m agents $\{a_1, a_2, \dots, a_m\}$, and for each agent a_i, there is a start location $s_i \in V$, and (3) m tasks $\{\boldsymbol{g}_1, \boldsymbol{g}_2, \dots, \boldsymbol{g}_m\}$, where each task $\boldsymbol{g}_j$ is characterized by a sequence of K_j goal locations $\boldsymbol{g}_j = \langle \boldsymbol{g}_j[1], \dots, \boldsymbol{g}_j[K_j] \rangle$. Each agent a_i can be assigned any task $\boldsymbol{g}_j$.

Let $\pi_i(t)$ denote the location of agent a_i at time t. A path $\pi_i = \langle \pi_i(0), \dots, \pi_i(T_i), \pi_i(T_i + 1), \dots \rangle$ for agent a_i is a sequence of locations that satisfies the following conditions: (1) The agent starts at its start location, that is $\pi_i(0) = s_i$; (2) At each timestep t, the agent either moves to a neighboring location $\pi_i(t+1) \in V$ where $(\pi_i(t), \pi_i(t+1)) \in E$, or stays in its current locations, that is $\pi_i(t) = \pi_i(t+1)$; and (3) The agent visits all goal locations of its assigned task $\boldsymbol{g}_j$ in sequence and remains in the final goal location at the *finish time* T_i, which is the minimum time T_i such that $\pi_i(t) = \boldsymbol{g}_j[K_j]$ for all times $t = T_i, \dots, \infty$.

Agents need to avoid collisions while moving to their goal locations. A collision between agents a_i and a_j is either: (1) a vertex collision $\langle a_i, a_j, u, t \rangle$, where two agents a_i and a_j are in the same location $u = \pi_i(t) = \pi_j(t)$ at time t; or (2) an edge collision $\langle a_i, a_j, u, v, t \rangle$ where two agents a_i and a_j traverse the same edge (u, v) in opposite directions $u = \pi_i(t) = \pi_j(t+1)$ and $v = \pi_i(t+1) = \pi_j(t+1)$ at timestep t. A *plan* consists of an assignment of tasks to agents and a path for each agent. A *solution* is a plan whose paths are collision-free. The flowtime $\sum_{i=1}^{m} T_i$ of a plan is the sum of the finish times of all agents. The problem of MG-TAPF is to find a solution that minimizes the flowtime. In this paper, we only consider the flowtime objective even though many of our results could be easily generalized to other objectives, such as makespan (the maximum of the finish times $\max_{1 \le i \le m} T_i$ of all agents) minimization.

III. Complexity

We show that the MG-TAPF problem is NP-hard to solve optimally for flowtime minimization, even when each task has only two goal locations. Similar to [10] and [35], we use a reduction from $\le$3,=3-SAT [33], an NP-complete version of the Boolean satisfiability problem. A $\le$3,=3-SAT instance consists of N Boolean variables $\{X_1, \dots, X_N\}$ and M disjunctive clauses $\{C_1, \dots, C_M\}$, where each variable appears in exactly three clauses, uncomplemented at least once, and complemented at least once, and each clause contains at most three literals. Its decision question asks whether there exists a satisfying assignment. We first show a constant-factor inapproximability result for makespan minimization.

Theorem 1: For any $\epsilon > 0$, it is NP-hard to find a $(4/3 - \epsilon)$-approximate solution to the MG-TAPF problem for makespan minimization, even if each task has exactly two goal locations.

Proof: We use a reduction similar to that used in the proof of Theorem 3 in [10] to construct an MG-TAPF instance with $m = M + 2N$ agents and the same number of tasks that has a solution with makespan three if and only if a given $\le$3,=3-SAT instance with N variables and M clauses is satisfiable.

We follow the notations used in the proof of Theorem 3 in [10] and point out the differences here: For each variable X_i in the $\le$3,=3-SAT instance, we construct two "literal" agents a_{iT} and a_{iF} with start locations s_{iT} and s_{iF}, and two tasks $\boldsymbol{g}_{iT}$ and $\boldsymbol{g}_{iF}$, each with two goal locations. We set $\boldsymbol{g}_{iT}[1] = s_{iT}$, $\boldsymbol{g}_{iT}[2] = t_{iT}$, $\boldsymbol{g}_{iF}[1] = s_{iF}$, and $\boldsymbol{g}_{iF}[2] = t_{iF}$. For each clause C_j in the ≤3,=3-SAT instance, we construct a "clause" agent a_j with start location c_j and a task $\boldsymbol{g}_j$ with two goal locations $\boldsymbol{g}_j[1] = c_j$ and $\boldsymbol{g}_j[2] = d_j$. Therefore, any optimal solution must assign every task to the agent whose start location is the first goal location of the task and let the agent execute the task.

Using the same arguments as in the proof of Theorem 3 in [10], we can show that the constructed MG-TAPF instance has a solution with makespan three if and only if the ≤3,=3-SAT instance is satisfiable, and always has a solution with makespan four, even if the ≤3,=3-SAT instance is unsatisfiable. For any $\epsilon > 0$, any MG-TAPF algorithm with approximation ratio $4/3 - \epsilon$ thus computes a solution with makespan three the ≤3,=3-SAT instance is satisfiable and thus solves ≤3,=3-SAT problem. ■

In the proof of Theorem 1, the MG-TAPF instance reduced from the given ≤3,=3-SAT instance has the property that the length of every path from the start location of every agent to the final goal location of the task assigned to the agent is at least three. Therefore, if the makespan is three, every agent arrives at the final goal location of its assigned task in exactly three timesteps, and the flowtime is $3m$. Moreover, if the makespan exceeds three, the flowtime exceeds $3m$, yielding the following theorem.

Theorem 2: It is NP-hard to find the optimal solution to the MG-TAPF problem for flowtime minimization, even if each task has exactly two goal locations.

IV. CBS-TA-MLA

The CBS-TA-MLA algorithm is a two-level search algorithm, where the low-level MLA* algorithm plans an optimal path for each agent based on the task assignment and the constraints provided by the high-level CBS-TA algorithm.

A. High Level: CBS-TA

Conflict-Based Search with Task Assignment (CBS-TA) is a best-first search algorithm, which was initially designed to solve the TAPF problems [28]. We extend it to solving MG-TAPF problem by replacing the low-level search algorithm with MLA*. Algorithm 1 shows the pseudo-code. CBS searches a binary tree, called constraint tree (CT), while CBS-TA searches a forest that contains multiple CTs. Each tree corresponds to a different task assignment. Each node, called CT node, in the CT contains (1) a Boolean value $root$, indicating whether the node is a root node of a CT; (2) an $assignment$, which is the task assignment of the node; (3) a set of $constraints$, where a vertex constraint $\langle a_i, u, t\rangle$ prohibits agent a_i from being at location u at time t and an edge constraint $\langle a_i, u, v, t\rangle$ prohibits agent a_i from moving along from u to v at timestep t; (4) a set of $paths$ with respect to the task assignment and the constraints; and (5) a $cost$, which is the flowtime of the paths [Lines 2-7].

An $m \times m$ cost matrix C stores the distances from the start locations of all agents to the final goal locations of all tasks where all intermediate goal locations of the task

Algorithm 1: High Level of CBS-TA-MLA

```
1  OPEN ← $\emptyset$
   // initialize first root node R
2  R.root ← True
3  R.assignment ← firstAssignment()
4  R.constraints ← $\emptyset$
5  for each agent $a_i$ do
6      R.paths[$a_i$] ← MLA*($a_i$, R.assignment[$a_i$], R.constraints)
7  R.cost ← flowtime(R.paths)
8  R.collisions ← findCollisions(R)
9  OPEN ← OPEN ∪ {$R$}
10 while OPEN ≠ $\emptyset$ do
11     N ← lowest cost node from OPEN
12     OPEN ← OPEN \ {N}
13     if N.paths do not have collisions then
14         Return N.assignment, N.paths
15     if N.root is True then
           // initialize new root node R with next-best task assignment
16         R.root ← True
17         R.assignment ← nextAssignment()
18         R.constraints ← $\emptyset$
19         for each agent $a_i$ do
20             R.paths[$a_i$] ← MLA*($a_i$, R.assignment[$a_i$], R.constraints)
21         R.cost ← flowtime(R.paths)
22         R.collisions ← findCollisions(R)
23         OPEN ← OPEN ∪ {$R$}
24     $\langle a_i, a_j, u, t\rangle / \langle a_i, a_j, u, v, t\rangle$ ← chooseCollision(N)
       // generate child nodes
25     for agent $a_k$ in {$a_i, a_j$} do
26         Q.root ← False
27         Q.assignment ← N.assignment
28         Q.constraints ← N.constraints ∪ {$\langle a_k, u, t\rangle / \langle a_k, u, v, t\rangle$}
29         Q.paths[$a_k$] ← MLA*($a_k$, Q.assignment[$a_k$], Q.constraints)
30         if Q.paths[$a_k$] is None then
31             continue to the next iteration
32         Q.cost ← flowtime(Q.paths)
33         Q.collisions ← findCollisions(Q)
34         OPEN ← OPEN ∪ {Q}
35 Return No Solution
```

are visited in sequence while ignoring collisions with the other agents. CBS-TA starts with a single root node with the best task assignment (the task assignment with the lowest flowtime which ignoring collisions among agents). The best task assignment is calculated by applying the Hungarian method [36] to the cost matrix C. Once the task assignment is calculated, the corresponding paths of the agents are planned by the low-level MLA* search algorithm . CBS-TA then finds collisions among the planned paths, stores the number of collisions in the node and adds the node to the OPEN list [Lines 8-9]. A new root node with the next-best task assignment is created if the currently expanded node is a root node [Lines 15-23]. We use the next-best task assignment algorithm in [28]. See [28] for details.

CBS-TA removes a node N with the lowest cost $N.cost$ from the OPEN list to expand (breaking ties in favor of the paths in node with the smallest number of collisions) [Lines 11-12]. First, it checks whether the number of collisions is 0. If so, N is declared a goal node, and $N.assignment$ and $N.paths$ are returned [Lines 13-14]. Otherwise, CBS-TA resolves an earliest vertex collision $\langle a_i, a_j, u, t\rangle$ (or edge collision $\langle a_i, a_j, u, v, t\rangle$) [Line 24] by generating two child nodes. Child nodes inherit the constraint set and paths from N [Lines 25-29]. CBS-TA adds constraint $\langle a_i, u, t\rangle$ (or $\langle a_i, u, v, t\rangle$) to the constraint set of one child node, and adds constraint $\langle a_j, u, t\rangle$ (or $\langle a_j, v, u, t\rangle$) to that of the other child node. It then calls the low-level MLA* search algorithm to replan the path of a_i (or a_j) to satisfy the new constraint set.

If such a path does not exist, CBS-TA prunes the child node [Lines 30-31]. Otherwise, CBS-TA updates the cost and the number of collisions between the newly planned path and the existing paths of the other agents and adds the child node to the OPEN list [Lines 32-34]. Once the OPEN list is empty, CBS-TA terminates the search unsuccessfully [Line 35].

B. Low Level: MLA*

Multi-Label Space-Time A* (MLA*) finds a time-optimal path for an agent a_i (a path with the smallest finish time T_i) that visits all goal locations of its assigned task $\boldsymbol{g}_j$ in sequence and obeys a set of constraints. MLA* was first introduced for two goal locations [31] and then extended to more than two goal locations [37]. MLA* extends Space-Time A* [38] by adding a label indicating the different segments between the goal locations, where label k indicates that the next goal location to visit is $\boldsymbol{g}_j[k]$.

We now formally describe MLA*. MLA* is an A* search whose states are tuples of a location, a time and a label. It starts with state $\langle s_i, 0, 1\rangle$, indicating agent a_i being at location s_i at time 0 with label 1. A directed edge exists from state $\langle u, t, k\rangle$ to state $\langle v, t+1, k'\rangle$ if and only if (1) $u = v$ or $(u, v) \in E$ and (2) $k' = k + 1$ if $v = \boldsymbol{g}_j[k]$ and $k' = k$ otherwise. To obey the constraints, the set of states $\{\langle v, t, k\rangle \mid k = 1, \ldots, K_j + 1\}$ is removed from the state space of agent a_i if and only if there is a vertex constraint $\langle a_i, v, t\rangle$, and the set of edges $\{(\langle u, t, k\rangle, \langle v, t+1, k'\rangle) \mid k = 1, \ldots, K_j\}$ is removed if and only if there is an edge constraint $\langle a_i, u, v, t\rangle$. If MLA* expands a goal state $\langle \boldsymbol{g}_j[K_j], t, K_j + 1\rangle$ and the agent can stay at the goal location forever (without violating any vertex constraints), it terminates and returns the path from the start state to the goal state.

During the search, the h-value of each state $\langle v, t, k\rangle$ is set to dist$(v, \boldsymbol{g}_j[k]) + \sum_{k'=k}^{K_j-1}$ dist$(g_j[k'], g_j[k'+1])$, that is, the shortest distance from location v to visit all unvisited goal locations in task $\boldsymbol{g}_j$ in sequence. The distances dist$(v, \boldsymbol{g}_j[k])$ from each location $v \in V$ to all goal locations $\boldsymbol{g}_j[k]$ with $j = 1, \ldots, m$ and $k = 1, \ldots, K_j$ are pre-computed by searching backward once from each goal location $\boldsymbol{g}_j[k]$ on graph G.

C. Properties of CBS-TA-MLA

We now show that CBS-TA-MLA is complete and optimal.

Theorem 3: CBS-TA-MLA is guaranteed to find an optimal solution if the given MG-TAPF instance is solvable and correctly identifies an unsolvable MG-TAPF instance with an upper bound of $\mathcal{O}(|V|^3 \cdot \sum_{j=1}^{m} K_j)$ on the finish time T_i of any agent at the final goal location of its assigned task.

Proof: The proof of the optimality of CBS-TA-MLA is trivial as CBS-TA and MLA* have been proved to be optimal in [28] and [31], respectively. As for the completeness, consider an arbitrary optimal solution to the given MG-TAPF instance with paths π_i. The solution can be divided chronologically into at most $\mathcal{K} = \sum_{j=1}^{m} K_j$ segments at breakpoints $t^{(0)} = 0, t^{(1)}, \ldots, t^{\mathcal{K}} = \max_i T_i$ where the label of at least one agent changes (because it reaches a new goal location of its assigned task) at each agent $t^{(\kappa)}$. Since there exists a solution with at most $U = \mathcal{O}(|V|^3)$ (U is a constant depending on V only) agent movements (edge traversals) to any solvable MAPF instance [39], there exist collision-free paths for all agents with makespan at most U that move each agent a_i from $\pi(t^{(\kappa-1)})$ to $\pi(t^{(\kappa)})$ and thus $t^{(\kappa)} - t^{(\kappa-1)} \leq U, \forall \kappa \geq 1$. Therefore, $t^{(\mathcal{K})} \leq U \cdot \sum_{j=1}^{m} K_j$. CBS-TA-MLA can thus safely prune any state whose time is larger than $U \cdot \sum_{j=1}^{m} K_j$ on the low level and terminate on Line 35 for any given unsolvable MG-TAPF instance when OPEN eventually becomes empty in finite time. ■

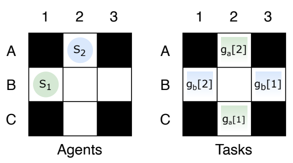

Fig. 1: An example instance with agents and tasks.

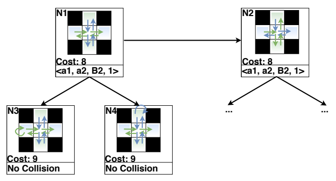

Fig. 2: The search forest created by the high-level search of CBS-TA-MLA for the example from Figure 1.

D. Example

Consider the example in Figure 1 with two agents a_1 and a_2 located at $s_1 = B1$ and $s_2 = A2$ respectively. Two tasks $\boldsymbol{g}_a$ and $\boldsymbol{g}_b$ will be assigned to two agents, where $\boldsymbol{g}_a[1] = C2$, $\boldsymbol{g}_a[2] = A2$, $\boldsymbol{g}_b[1] = B3$ and $\boldsymbol{g}_b[2] = B1$. The corresponding high-level forest of CBS-TA-MLA is shown in Figure 2. The first root CT node N_1 assigns $\boldsymbol{g}_b$ to a_1 and $\boldsymbol{g}_a$ to a_2. CBS-TA-MLA chooses to expand the node with the minimum cost, which is N_1. It detects two collisions $\langle a_1, a_2, B2, 1\rangle$ and $\langle a_1, a_2, B2, 3\rangle$. Since N_1 is a root CT node, the second root node N_2 with next-best task assignment $\boldsymbol{g}_a$ to a_1 and $\boldsymbol{g}_b$ to a_2 is created and added to the OPEN list. CBS-TA-MLA resolves the earliest collision $\langle a_1, a_2, B2, 1\rangle$ by creating two child nodes N_3 and N_4, where a_1 is prohibited from being in location $B2$ at time 1 in N_3 by adding $\langle a_1, B2, 1\rangle$ to $N_3.constraint$ and a_2 is prohibited from being in location $B2$ at time 1 in N_4. As the low-level search can find paths for the replanned agents, both child nodes are added to the OPEN list. In the next iteration, CBS-TA-MLA picks N_2 for expansion, but does not create a root node since there are only two possible task assignments. The paths in N_2 have collisions, and thus CBS-TA-MLA generates two child nodes and adds them to the OPEN list. Then, it selects N_3 which has no collisions. So, it declares N_3 the goal node and returns the task assignment and paths.

V. Extensions of CBS-TA-MLA

This section introduces three extensions of CBS-TA-MLA, namely an improved optimal version (with heuristics), a bounded-suboptimal version and a greedy version.

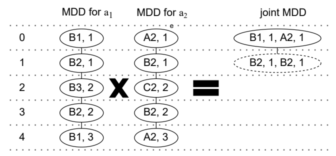

Fig. 3: The MDDs and joint MDD for the example instance in Figure 1. Levels of MDD nodes are shown on the left.

A. CBS-TA-MLA with Heuristics (CBSH-TA-MLA)

CBS with heuristics [22] introduces three admissible heuristics (namely CG, DG and WDG) for the high-level search of CBS, which reduce the number of expanded CT nodes. The collisions among paths of a CT node are classified in three types [21]: cardinal collisions if both of the resulting child nodes have a larger cost than the node itself, semi-cardinal collisions if only one of its child nodes has a larger cost than the node itself, and non-cardinal collisions if both of the child nodes have the same cost as the node itself.

The technique to classify collisions is a Multi-Valued Decision Diagrams (MDDs) [21]. An MDD for agent a_i at CT node N is a directed acyclic graph consisting of all possible cost-optimal paths of a_i with respect to the task assignment and constraints of N. Each MDD node consists of a location v and a level/time t. A collision between agents a_i and a_j at time t is cardinal iff the contested vertex/edge is the only vertex/edge at level t of the MDDs of both agents ($t = 1$ in Figure 3). To make the MDDs applicable in our case in which the cost-minimal paths contain all goal locations of the assigned task in the correct sequence, we add a label to each MDD node (see Figure 3).

CG heuristic only considers the cardinal collisions among paths. DG heuristic considers the dependency among agents and WDG heuristic considers the extra cost that each pair of dependent agents contributes to the total cost. WDG dominates DG, which dominates CG. See [22] for details.

We adopt the techniques of CBS-TA-MLA for the CBS-TA-MLA with Heuristics algorithm (CBSH-TA-MLA). We maintain a new variable min_f_val for the minimum f-value of all nodes in the OPEN list. Each node N has two additional fields, namely $N.h\_val$ to represent the admissible h-value and $N.f\_val = N.cost + N.h\_val$ to represent the priority in the OPEN list. See [22] for details of $N.h\_val$ computation method. The chooseCollision(N) function in Algorithm 1 chooses cardinal collisions first, semi-cardinal collisions next and non-cardinal collisions last (breaking ties in favor of the earliest collision).

B. Enhanced CBS-TA-MLA (ECBS-TA-MLA)

Similar to Enhanced CBS (ECBS) [34], ECBS-TA-MLA is a bounded-suboptimal algorithm for MG-TAPF. It uses a focal search on both high and low levels. A focal search maintains two lists: OPEN and FOCAL. The OPEN list is sorted in increasing order of the f-values. The best node N_{best} in the OPEN list is the node with the minimum f-value, which is denoted by f_{best}. The FOCAL list contains that subset of the nodes in the OPEN list whose f-values are at most $\omega \cdot f_{best}$. The FOCAL list is sorted according to

(a) Dense map (b) Sparse map

Fig. 4: Two maps used for the experiments.

some other heuristic. The FOCAL search guarantees to find solutions that are a factor of at most ω worse than optimal by always expanding the best node in the FOCAL list.

Low-level focal search: The low-level focal search prioritizes nodes in the OPEN list with f-values and nodes in the FOCAL list with the number of collisions in paths between the current agent a_i and the other agents in the CT node. When it finds a solution, it returns the path and the f-value of the best node n in the OPEN list, which is the lower bound on the cost of the time-optimal path, denoted by $f_{best}(a_i)$.

High-level focal search: The high-level focal search sorts CT nodes in the OPEN list in increasing order of the sum of the lower bounds of all agents $LB(N) = \sum_{i=1}^{m} f_{best}(a_i)$. Let N_{best} denote the node N in the OPEN list with the minimum $LB(N)$. The FOCAL list contains that subset of CT nodes N with $N.cost \leq \omega \cdot LB(N_{best})$. The nodes in the FOCAL list are sorted in increasing order of the number of collisions among $N.paths$. Since $LB(N_{best})$ is provably a lower bound on the optimal flowtime C^*, the cost of any CT node in the FOCAL list is at most $\omega \cdot C^*$. As a result, once a solution is found, its flowtime is at most $\omega \cdot C^*$, so it is bounded-suboptimal with suboptimality factor ω.

C. Greedy CBS-TA-MLA (TA+CBS-MLA)

The TA+CBS-MLA performs best task assignment (TA) followed by the CBS-MLA algorithm. It starts with the root node with the best task assignment and does not generate any other root nodes. TA+CBS-MLA provides no optimality or completeness guarantee.

VI. Experiments

This section describes our experimental results on a 2.3GHz Intel Core i5 laptop with 16GB RAM. The algorithms are implemented in Python and tested on three maps, namely (1) a dense map, which is a 20×20 warehouse map with 30% obstacles (Figure 4a), (2) a sparse map, which is a 32×32 random map with 10% obstacles (Figure 4b), and (3) a 32×32 empty map, all with a time limit of 120 seconds unless otherwise specified.

A. CBSH-TA-MLA

We evaluate CBSH-TA-MLA using two test sets. In the first set, we use the dense map with 10 agents/tasks with randomly generated locations and report the success rate, the average number of expanded CT nodes and the average runtime over 100 instances. In the second test set, we use the sparse map and the empty map and report the above three values over 50 instances with a time limit of 300 seconds. The last two values are averaged over instances that are successfully solved by CBSH-TA-MLA with all four

TABLE I: Results for CBSH-TA-MLA using different heuristics on different maps with different numbers of agents/tasks, where each task consists of two goal locations.

Map	Agents	Heuristics	Success Rate	Nodes Expanded	Runtime (s)
Dense Map	10	No	98/100	34.18	2.54
		CG	98/100	28.86	2.23
		DG	98/100	25.54	**2.09**
		WDG	97/100	**8.98**	3.53
Sparse Map	20	No	44/50	42.59	15.68
		CG	44/50	34.65	13.21
		DG	46/50	30.58	**11.8**
		WDG	46/50	**4.63**	13.17
Empty Map	20	No	46/50	23.02	6.96
		CG	46/50	12.08	4.03
		DG	50/50	3.48	**1.52**
		WDG	48/50	**2.11**	2.82
Empty Map	30	No	40/50	20.225	9.55
		CG	40/50	14.75	7.15
		DG	46/50	7.375	**4.24**
		WDG	46/50	**4.375**	4.52

TABLE II: Results for ECBS-TA-MLA with different ω on different maps with different numbers of agents/tasks and different numbers of goal locations per task.

Map	Agents	Goal Locations	ω	Success Rate	Nodes Expanded	Runtime (s)	Cost
Dense Map	2	2	1.00	100/100	22.68	2.20	**144.69**
			1.05	100/100	6.00	0.70	145.57
			1.10	100/100	3.27	0.32	146.77
			1.30	100/100	**0.85**	**0.08**	148.19
	20	2	1.00	29/100	247.46	28.47	**249.46**
			1.05	78/100	20.42	2.22	255.75
			1.10	97/100	6.68	0.83	260.21
			1.30	**100/100**	**3.46**	**0.58**	262.32
	30	2	1.00	0/100	/	/	/
			1.05	14/100	/	/	/
			1.10	48/100	/	/	/
			1.30	**96/100**	/	/	/
Sparse Map	10	2	1.00	100/100	3.14	0.58	**304.79**
			1.05	100/100	0.47	0.12	305.64
			1.10	100/100	0.36	0.11	306.32
			1.30	100/100	**0.35**	**0.10**	306.79
	20	2	1.0	76/100	28.34	9.22	**537.49**
			1.05	100/100	1.96	0.79	541.38
			1.1	100/100	1.48	**0.69**	543.28
			1.3	100/100	**1.35**	0.73	544.36
	10	10	1.00	73/100	9.33	13.35	**1846.46**
			1.05	**88/100**	0.68	4.52	1856.49
			1.10	86/100	**0.65**	3.56	1858.03
			1.30	85/100	**0.65**	**0.75**	1858.03

TABLE III: Results for different numbers of goal locations per task for ECBS-TA-MLA with different ω on the sparse map. The number of goal locations differs in different tasks.

Goal Locations	ω	Success Rate	Nodes Expanded	Runtime (s)
2-5	1.1	95/100	1.26	1.96
	1.3	95/100	1.26	2.13
6-10	1.1	72/100	2.04	3.51
	1.3	72/100	2.00	3.61
11-15	1.1	80/100	7.03	18.36
	1.3	83/100	7.20	19.46
16-20	1.1	78/100	14.61	48.11
	1.3	83/100	15.15	51.37

heuristics. Table I shows that WDG always results in the smallest number of expanded nodes, while DG always results in the smallest average runtime on all instances. This is so because WDG needs to compute the weights of the edges of the pairwise dependency graph, which requires executing the CBS-TA-MLA algorithm for two agents repeatedly.

B. *ECBS-TA-MLA*

We evaluate ECBS-TA-MLA with different suboptimality factors ω on the dense and the sparse maps with different numbers of agents/tasks (for both maps) and different numbers of goal locations in each task (for the sparse map). We report the success rate, the average number of expanded CT nodes, the average runtime and the average cost over 100 instances in Table II. The last three values are averaged over instances that are successfully solved by ECBS-TA-MLA with all four ω. As expected, ECBS-TA-MLA achieves high success rates on the instances with small numbers of agents/tasks. The success rates only drop a bit with $\omega = 1.3$ when the number of agents increases up to 30. In addition, when we increase ω, the average number of expanded CT nodes and the average runtime decrease, while the average cost increases.

We test ECBS-TA-MLA when tasks have different number of goal locations with 10 agents/tasks and report the same values as in Table I in Table III. The experiment shows that, with an appropriate ω, the success rate is still over 70% within two minutes, even with a maximum of 20 goal locations per task.

TABLE IV: Results for TA+CBS-MLA and CBS-TA-MLA on the dense map with 10 agents/tasks.

	Success Rate	Runtime (s)	Cost
TA+CBS-MLA	**100/100**	**1.92**	148.16
CBS-TA-MLA	99/100	3.04	**146.63**

C. *TA+CBS-MLA*

We compare TA+CBS-MLA against CBS-TA-MLA on the dense map and report the success rate, the average cost, and the average runtime with 10 agents/tasks, where each task consists of two goal locations, over 100 instances in Table IV. CBS-TA-MLA outperforms TA+CBS-MLA in the average cost, but TA+CBS-MLA outperforms CBS-TA-MLA in both the success rate and the average runtime. The reason for this is that the number of possible task assignments is large, namely $10! \approx 3$ millions, so CBS-TA-MLA spends a significant amount of time computing task assignments.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented the CBS-TA-MLA algorithm to solve the MG-TAPF problem optimally. We presented three enhanced variants of CBS-TA-MLA, namely (1) optimal variant CBSH-TA-MLA, which speeds up CBS-TA-MLA by adding a heuristic, (2) bounded-suboptimal variant ECBS-TA-MLA, which speeds up CBS-TA-MLA by using focal search and (3) greedy variant TA+CBS-MLA, which commits to the most promising task assignment without exploring other assignments. We conducted experiments to evaluate these algorithms in different settings. It is future research to incorporate additional enhancements (such as disjoint splitting for the high-level search and incremental A* for the low-level search) into CBS-TA-MLA to improve its efficiency.

ACKNOWLEDGMENT

The research at Simon Fraser University was supported by the Natural Sciences and Engineering Research Council of Canada under grant number RGPIN2020-06540. The research at the University of Southern California was supported by the National Science Foundation under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, and 2112533 as well as a gift from Amazon.

References

[1] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *SoCS*, 2019, pp. 151–159.

[2] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, pp. 9–20, 2008.

[3] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of Artificial Intelligence Research*, vol. 31, pp. 591–656, 2008.

[4] R. Morris, C. Pasareanu, K. Luckow, W. Malik, H. Ma, T. K. S. Kumar, and S. Koenig, "Planning, scheduling and monitoring for airport surface operations," in *AAAI Workshop on Planning for Hybrid Systems*, 2016.

[5] J. Li, K. Sun, H. Ma, A. Felner, T. K. S. Kumar, and S. Koenig, "Moving agents in formation in congested environments," in *AAMAS*, 2020, pp. 726–734.

[6] V. Nguyen, P. Obermeier, T. C. Son, T. Schaub, and W. Yeoh, "Generalized target assignment and path finding using answer set programming," in *IJCAI*, 2017, pp. 1216–1223.

[7] M. Liu, H. Ma, J. Li, and S. Koenig, "Task and path planning for multi-agent pickup and delivery," in *AAMAS*, 2019, pp. 2253–2255.

[8] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *AAAI*, 2013, pp. 1444–1449.

[9] P. Surynek, "An optimization variant of multi-robot path planning is intractable," in *AAAI*, 2010, pp. 1261–1263.

[10] H. Ma, C. Tovey, G. Sharon, T. K. S. Kumar, and S. Koenig, "Multi-agent path finding with payload transfers and the package-exchange robot-routing problem," in *AAAI*, 2016, pp. 3166–3173.

[11] J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *ICRA*, 2013, pp. 3612–3617.

[12] P. Surynek, "Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally," in *IJCAI*, 2015, pp. 1916–1922.

[13] E. Erdem, D. G. Kisa, U. Oztok, and P. Schueller, "A general formal framework for pathfinding problems with multiple agents," in *AAAI*, 2013, pp. 290–296.

[14] R. Luna and K. E. Bekris, "Push and Swap: Fast cooperative path-finding with completeness guarantees," in *IJCAI*, 2011, pp. 294–300.

[15] K. Wang and A. Botea, "MAPP: A scalable multi-agent path planning algorithm with tractability and completeness guarantees," *Journal of Artificial Intelligence Research*, vol. 42, pp. 55–90, 2011.

[16] B. de Wilde, A. W. ter Mors, and C. Witteveen, "Push and Rotate: Cooperative multi-agent path planning," in *AAMAS*, 2013, pp. 87–94.

[17] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 195, pp. 470–495, 2013.

[18] G. Wagner, "Subdimensional expansion: A framework for computationally tractable multirobot path planning," Ph.D. dissertation, Carnegie Mellon University, 2015.

[19] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[20] E. Lam, P. Le Bodic, D. Harabor, and P. J. Stuckey, "Branch-and-cut-and-price for multi-agent pathfinding," in *IJCAI*, 2019, pp. 1289–1296.

[21] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and S. E. Shimony, "ICBS: Improved conflict-based search algorithm for multi-agent pathfinding," in *IJCAI*, 2015, pp. 740–746.

[22] J. Li, A. Felner, E. Boyarski, H. Ma, and S. Koenig, "Improved heuristics for multi-agent path finding with conflict-based search," in *IJCAI*, 2019, pp. 442–449.

[23] G. Gange, D. Harabor, and P. J. Stuckey, "Lazy CBS: Implicit conflict-based search using lazy clause generation," in *ICAPS*, 2019, pp. 155–162.

[24] E. Boyarski, A. Felner, D. Harabor, P. J. Stuckey, L. Cohen, J. Li, and S. Koenig, "Iterative-deepening conflict-based search," in *IJCAI*, 2020, pp. 4084–4090.

[25] P. Surynek, "Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering," in *AAAI*, 2021.

[26] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," in *AAMAS*, 2016, pp. 1144–1152.

[27] J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," in *Algorithmic Foundations of Robotics X, Springer Tracts in Advanced Robotics*, E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, Eds., 2013, vol. 86, pp. 157–173.

[28] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *AAMAS*, 2018.

[29] C. Henkel, J. Abbenseth, and M. Toussaint, "An optimal algorithm to solve the combined task allocation and path finding problem," in *IROS*, 2019, pp. 4140–4146.

[30] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *AAMAS*, 2017, pp. 837–845.

[31] F. Grenouilleau, W.-J. van Hoeve, and J. N. Hooker, "A multi-label A* algorithm for multi-agent pathfinding," in *ICAPS*, 2019, pp. 181–185.

[32] K. Brown, O. Peltzer, M. A. Sehr, M. Schwager, and M. J. Kochenderfer, "Optimal sequential task assignment and path finding for multi-agent robotic assembly planning," in *ICRA*, 2020, pp. 441–447.

[33] C. A. Tovey, "A simplified NP-complete satisfiability problem," *Discrete Applied Mathematics*, vol. 8, pp. 85–90, 1984.

[34] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *SoCS*, 2014, pp. 19–27.

[35] H. Ma, G. Wagner, A. Felner, J. Li, T. K. S. Kumar, and S. Koenig, "Multi-agent path finding with deadlines: Preliminary results," in *AAMAS*, 2018, pp. 2004–2006.

[36] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[37] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *AAMAS*, 2020, pp. 1898–1900.

[38] D. Silver, "Cooperative pathfinding," in *AIIDE*, 2005, pp. 117–122.

[39] J. Yu and D. Rus, "Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms," in *Algorithmic Foundations of Robotics XI, Springer Tracts in Advanced Robotics*, 2015, vol. 107, pp. 729–746.