Solving Multi-Agent Target Assignment and Path Finding with a Single Constraint Tree

Yimin Tang
August 01, 2023
CMU-RI-TR-23-41

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis Committee:

Prof. Katia Sycara, *chair*	The Robotics Institute, Carnegie Mellon University
Prof. Jiaoyang Li	The Robotics Institute, Carnegie Mellon University
Prof. Changliu Liu	The Robotics Institute, Carnegie Mellon University
Sha Yi	The Robotics Institute, Carnegie Mellon University

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

Multi-Agent Path Finding (MAPF) and Combined Target-Assignment and Path-Finding problem (TAPF) arise in many applications such as robotics, computer gaming, warehouse automation and traffic management at road intersections. Combined Target-Assignment and Path-Finding problem (TAPF) requires simultaneously assigning targets to agents and planning collision-free paths for agents from their start locations to their assigned targets. As a leading approach to address TAPF, Conflict-Based Search with Target Assignment (CBS-TA) leverages both K-best target assignments to create multiple search trees and Conflict-Based Search (CBS) to resolve collisions in each search tree. While being able to find an optimal solution, CBS-TA suffers from scalability due to the duplicated collision resolution in multiple trees and the expensive computation of K-best assignments. We therefore develop Incremental Target Assignment Conflict-Based Search (ITA-CBS) to bypass these two computational bottlenecks. ITA-CBS generates only a single search tree and avoids computing K-best assignments by incrementally computing new 1-best assignments during the search. We show that, in theory, ITA-CBS is guaranteed to find an optimal solution and, in practice, is computationally efficient.

Contents

Nomenclature

Ω	Constraint set in CT node
π	Path solution in CT node
π_{ta}	Task Assignment solution in CT node
A	Permissible matrix mapping agents to goal locations
a_{ij}	0 or 1, represent if agent i can go to goal location j
c	flowtime or cost in CT node
E	Edge set in G
e	Edge in E
G	Graph
g_i	Goal location for agent i
H	Constraint Tree (CT) node
I	Agent index set
M	Number of different target locations
M_c	Cost Matrix in CT node
N	Number of agents
p^i	One path for agent i
r	Root flag in CT node

s_i	Start location for agent i
T^i	The final arrive time for agent i
V	Vertex set in G
v_t^i	The position of agent i at timestep t

List of Figures

1 Introduction

1.1 Introduction

Multi-Agent Path Finding problem (MAPF) requires planning collision-free paths for multiple agents from their respective start locations to pre-assigned target locations while minimizing the sum of individual path costs [1]. Solving MAPF to optimality is NP-hard [2], and many algorithms have been developed to handle this computational challenge. Among them, Conflict-Based Search (CBS) [3] is an efficient approach that finds an optimal solution to MAPF.

This work considers a variant of MAPF that is often referred to as Combined Target-Assignment and Path-Finding problem (TAPF) [4, 5], where the target locations of the agents are not pre-assigned but need to be allocated during the computation: TAPF requires assigning each agent a unique target (location) out of a pre-specified set of candidate targets and then finds collision-free paths for the agents so that the sum of path costs is minimized. When the candidate target set of each agent contains only a single target, TAPF becomes MAPF and is thus NP-hard.

MAPF and TAPF arise in many applications such as robotics [6], computer gaming [7], warehouse automation [8], traffic management at road intersections [9]. Several attempts [5, 10] have been made to solve TAPF optimally by leveraging MAPF algorithms such as CBS [3]. Among them, a leading approach is Conflict-Based Search with Target Assignment (CBS-TA) [5], which simultaneously explores different target assignments and creates multiple search trees (i.e., a CBS forest), while planning collision-free paths with respect to each assignment.

Specifically, given a target assignment, CBS-TA leverages a CBS-like search to construct a constraint tree to resolve collisions. In the meanwhile, CBS-TA leverages a K-best target assignment technique [11, 12] to compute the next cheapest target assignment (by increasing K to K+1, intuitively speaking) during the CBS-like search when necessary, which creates a new constraint tree. Consequently, CBS-TA generates multiple constraint trees by intelligently interleaving the CBS-like search and K-best target assignment and is able to compute an optimal solution for TAPF.

CBS-TA suffers from scalability as the number of agents or targets increases for the following two reasons. First, CBS-TA may resolve the same collision in multiple search trees many times, leading to duplicated computation and low search efficiency. Second, CBS-TA involves solving a K-best target assignment problem, which is often computationally expensive. This work thus attempts to bypass these two

computational bottlenecks by exploring a new framework for integrating CBS with target assignment. The resulting algorithm is called Incremental Target Assignment Conflict-Based Search (ITA-CBS).

1.2 Thesis Contribution

In this thesis, we present a new algorithm for rapidly solving Combined Target-Assignment and Path-Finding problem (TAPF) optimally. The primary characteristics of our proposed approach are as follows:

1. ITA-CBS is complete and optimal.
2. ITA-CBS adopts a unique strategy wherein it creates only a single search tree during the search process. This attribute is thus able to avoid duplicated collision resolution in different trees as in CBS-TA.
3. ITA-CBS completely avoids solving the K-best assignment problem, and instead, ITA-CBS updates the target assignment in an incremental manner during the CBS-like search, which further reduces the computational effort.

Our experimental results show significant improvement in efficiency: ITA-CBS is faster than CBS-TA in 96.1% testcases, 5 times faster in 38.7% testcases, and 100 times faster in 5.6% testcases than CBS-TA among 6,334 effective testcases which are solved by at least one algorithm.

2 Background

2.1 Multi-Agent Path Finding Problem (MAPF)

Multi-Agent Path Finding problem (MAPF) can be viewed as a special case of TAPF where each agent can be assigned to only one target location. MAPF has a long history [13, 14] and remains an active research problem [15, 16]. A variety of methods are developed to address MAPF, trading off completeness and optimality for runtime efficiency. These methods range from decoupled methods [14, 17, 18], which plan a path for each agent independently and synthesize the paths, to coupled methods [1], where all agents are planned together. Other methods [3, 19] consider agents that are planned independently at first and then together only when needed in order to resolve agent-agent conflicts. Conflict-Based Search (CBS) [3] is optimal with respect to flowtime and forms the foundation of this paper.

CBS is a two-level search algorithm that finds an optimal solution with minimum flowtime. Its low level plans a shortest path for an agent from its start location to target location. Its high level searches a binary Constraint Tree (CT). Each CT node $H = (c, \Omega, \pi)$ includes a scalar flowtime(cost) c, constraint set Ω and plan π which is a set of paths for all agents from their start locations to target locations, satisfying Ω. It is obvious that root CT node has an empty constraint set Ω and has the lowest c among all CT nodes within a single Constraint Tree.

In each H, CBS only select and resolve the first conflict, even when multiple collisions occur in the plan. To resolve a conflict in H, we can formulate two constraints, wherein each constraint prohibits one agent from executing its originally intended action at timestep t, and then add them individually to two successor CT nodes. Here we also define two types of constraints, namely vertex constraint (i, v, t) that prohibits agent i from occupying vertex v at timestep t and edge constraint (i, u, v, t) that prohibits agent i from going from vertex u to vertex v at timestep t. These two child CT nodes will be added to a priority queue sorted by cost. By maintaining this priority queue, it can be proved that CBS is optimal with respect to the flowtime.

2.2 Assignment Problem

Given N agents, M tasks, and a $N \times M$ matrix M_c denoting the corresponding assignment cost of each task to each agent, the task assignment problem [20–22] seeks to allocate the tasks to agents such that each agent is assigned to a unique task and

the total assignment cost is minimized. The assignment problem is also known as the maximum weighted bipartite matching problem and it is a widely-studied problem applicable to many domains [20]. Popular methods to address this problem include Hungarian algorithm [21, 22] and Successive Shortest Path (SSP) algorithm [23, 24]. Additionally, the Dynamic Hungarian algorithm [25] seeks to quickly re-compute an optimal assignment based on the existing assignment, when some entries change in matrix M_c.

The Hungarian algorithm formulates the assignment problem as a bipartite graph matching problem and solves it by finding a minimum-cost perfect matching with a runtime complexity of $O(n^3)$, where $n = \max(N, M)$. In our TAPF problem setting, the time complexity is $O(M^3)$. SSP algorithm formulates the assignment problem as a minimum-cost flow problem and solves it via the well-known Dijkstra algorithm with a runtime complexity of $O(fn^2)$, f is the max flow which is equal to N in Assignment Problem, so the complexity is $O(NM^2)$. Dynamic Hungarian runs faster than computing an assignment from scratch after matrix M_c changes [25]. And its time complexity is $O(kn^2)$, where k is the number of changed row/column in M_c.

2.3 Target-Assignment and Path-Finding Problem (TAPF)

Target-Assignment and Path-Finding (TAPF) can be viewed as a combination of the MAPF problem and the assignment problem. While conventional MAPF has a pre-defined target location for each agent, TAPF and its variants [5, 26–32] seek to simultaneously allocate the targets to agents and find conflict-free paths for the agents. Of close relevance to this work is CBS-TA [5], which is a leading algorithm in the literature that solves TAPF to optimality respect to flowtime. Some work [26, 33–36] follows the similar CBS forest idea, but none of them is designed to solve TAPF optimally.

CBS-TA operates on this principle: a fixed Target Assignment solution transforms a TAPF problem into a MAPF problem, and each MAPF problem has a binary Constraint Tree (CT). CBS-TA maintains one priority queue to store nodes from all CTs and generates root CT nodes for different target assignments lazily. Since a root CT node has the lowest flowtime for the given TA, if it is higher than all CT nodes in

our priority queue, we do not expand it. Therefore, CBS-TA generates root CT nodes in ascending order of their costs and only needs to generate the next root CT node if the current one is expanded. That is, given a root CT node with a TA, CBS-TA needs to compute the next best TA efficiently. So CBS-TA can efficiently explores all nodes of various CTs (CBS forest) by enumerating every TA solution.

Each CT node in CBS-TA, denoted $H = (c, \Omega, \pi, r, \pi_{ta})$, has two extra fields compared to CBS algorithm: a root flag r signifying if the node is a root, and a TA solution π_{ta}. CBS-TA keeps a priority queue for storing H from all CTs and lazily generates root H for varying π_{ta}. Motivated by K-best task assignment algorithms [11, 12] and SSP with Dijkstra algorithm, CBS-TA finds the succeeding optimal TA with $O(N^2M^2)$.

3 Problem Definition and Method

3.1 Problem Definition

We define the Combined Target-Assignment and Path-Finding problem (TAPF) as follows. Let $I = \{1, 2, \cdots, N\}$ denote a set of N agents. Let $G = (V, E)$ denote an undirected graph, where each vertex $v \in V$ represents a possible location of an agent in the workspace, and each edge $e \in E$ is a unit-length edge between two vertices that moves an agent from one vertex to the other. Self-loop edges are allowed, which represent "wait-in-place" actions. Each agent $i \in I$ has a unique start location $s_i \in V$. Let $\{g_j \in V | j \in \{1, 2, ..., M\}\}$, $M \geq N$, denote the set of all M target locations. Let A denote a binary $N \times M$ matrix, where each entry a_{ij} (the i-th row and j-th column in A) is one if agent i is eligible to be assigned to target g_j and zero otherwise. Our task is to assign each agent i a unique target g_j while ensuring $a_{ij} = 1$ and plan corresponding collision-free paths.

Each action of agents, either waiting in place or moving to an adjacent vertex, takes a time unit. Let $p^i = [v^i_0, v^i_1, ..., v^i_{T^i}], v^i_k \in V$ denote a path of agent i from v^i_0 to $v^i_{T^i}$ with the arrival time T^i. This work considers two types of agent-agent conflicts along their paths. The first type is the *vertex conflict*, where two agents i, j occupy the same vertex at the same time. The second type is the *edge conflict*, where two agents go through the same edge from opposite directions at the same time (i.e. $v^i_t = v^j_{t+1}$ and $v^i_{t+1} = v^j_t$).

The goal of the TAPF problem is to find a set of paths $\{p^i | i \in I\}$ for all agents such that, for each agent i:

1. $v^i_0 = s_i$ (i.e., agent i starts from its start location);
2. $v^i_t = g_j, \forall t \in [T^i, \max\{T^k | \forall k \in I\}]$ and $a_{ij} = 1$ (i.e., agent i stops at a target location g_j, which is eligible to be assigned to i, until all agents reach their goals);
3. Every pair of adjacent vertices in path p^i is either identical or connected by an edge (i.e., $v^i_k = v^i_{k+1} \vee (v^i_k, v^i_{k+1}) \in E, \forall k \in [0, T^i - 1]$);
4. $\{p^i | i \in I\}$ is conflict-free;
5. The flowtime $\sum_{i=1}^{N} T^i$ is minimized.

Note that no two agents can share the same target location, otherwise their paths must have conflicts.

3.2 Incremental Target Assignment Conflict-Based Search (ITA-CBS)

Our ITA-CBS, as shown in Algorithms 1 and 2, has the same low-level shortest path search as CBS and CBS-TA algorithm, but its high-level search is different. Each CT node $H = (c, \Omega, \pi, \pi_{ta}, M_c)$ in ITA-CBS has two additional fields compared to that in CBS: a TA (i.e., Target Assignment) solution and a $N \times M$ cost matrix M_c, where each entry describes the length of the shortest path from the corresponding start to target locations that satisfies the constraint set Ω. ITA-CBS begins by creating the first CT node with an empty Ω and the corresponding M_c and π_{ta} (Algorithm 1; Line 1-6). ITA-CBS maintains one priority queue to store all CT nodes that are generated during the search (Algorithm 1; Line 7-9, 24). ITA-CBS selects a CT node H_{cur} with the minimum cost from the priority queue and checks if it includes a conflict-free solution. If so, ITA-CBS is guaranteed to find an optimal solution (Algorithm 1; Line 10-13). Otherwise, ITA-CBS uses the first detected conflict to create two new constraints (Algorithm 1; Line 14) as in CBS. Then ITA-CBS creates two child nodes identical to the current node H and adds each constraint respectively into the constraint set of the two child nodes (Algorithm 1; Line 15-21).

For each new node Q (with a constraint on agent i added), the low-level search is invoked for agent i to recompute its optimal paths subject to the new constraint set from its start to all possible targets (Algorithm 2). The cost of these planned paths are then used to update the cost matrix M_c in Q. Since M_c changes, the TA solution π_{ta} should also be updated. We use dynamic Hungarian algorithm [25] to get the assignment solution more efficiently, and compute the solution path and total cost (Algorithm 1; Line 22-24).

3.3 ITA-CBS Example

An example of our algorithm is shown in Figure 3.1. The map has 5 vertices, a, b, c, d, e, and there are 2 agents 1 and 2. Agent 1's target location set is $\{d, e\}$ and agent 2's set is $\{c, e\}$. Each blue rounded rectangle in our figure represents a CT node

Algorithm 1 ITA-CBS Algorithm

Input: Graph, start and target locations
Output: Optimal path for each agent

OPEN = PriorityQueue()
$\Omega_0 = \emptyset$
M_c^0 = findAllShortestPath(Ω_0)
π_{ta}^0 = assignAlgorithm(M_c^0)
c_0, π_0 = getSolutionPath(π_{ta}^0, M_c^0)
$H_0 = \{c_0, \Omega_0, \pi_0, \pi_{ta}^0, M_c^0\}$
Insert H_0 to OPEN
while OPEN not empty **do**
    H_{cur} = OPEN front node; OPEN.pop()
    Validate the paths in H_{cur} until a conflict occurs
    if H_{cur} has no conflict **then**
        return $H_{cur}.\pi$
    Conflict = (i, j, v_i^{t-1}, v_i^t, v_j^{t-1}, v_j^t, t) from H_{cur}
    for each agent i in Conflict **do**
        Q = H_{cur}
        if Conflict is vertex collision **then**
            Q.Ω = Q.Ω $\cup$ (i, v_i^t, t)
        else
            Q.Ω = Q.Ω $\cup$ (i, v_i^{t-1}, v_i^t, t)
        Q.M_c = updateCostMatrix(Q.M_c, Q.Ω)
        Q.π_{ta} = assignAlgorithm(Q.M_c)
        $Q.c, Q.\pi$ = getSolutionPath(Q.π_{ta}, Q.M_c)
        Insert Q to OPEN
return No valid solution

Algorithm 2 updateCostMatrix

Input: costMatrix M_c^{in}, constraint set Ω
Output: M_c^{out}

idx = Ω.last.i
$M_c^{out} = M_c^{in}$
for each target location j **do**
    M_c^{out}[idx][j] = shortestPathSearch(idx, j)
return M_c^{out}

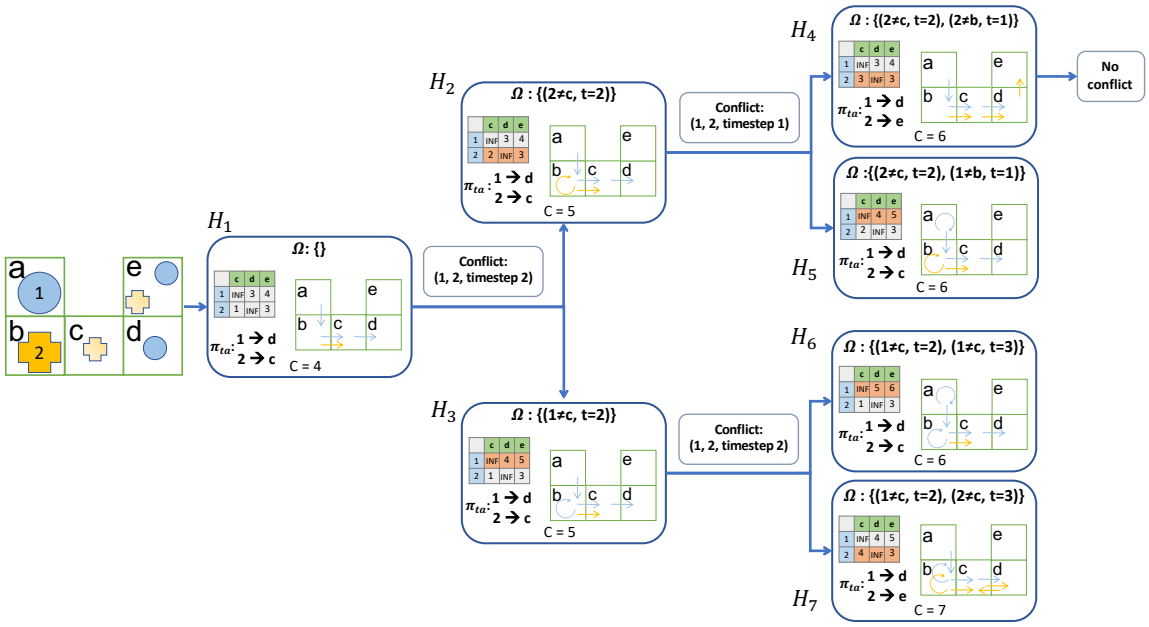


Figure 3.1: (1) Leftmost: A simple map with 5 cells (a, b, c, d, e) and 2 agents $(1, 2)$. Agent 1's target location set is $\{d, e\}$ and agent 2's set is $\{c, e\}$. (2) Each blue rounded rectangle represents a CT node H. Within each CT node, we have: a constraint set Ω, a cost matrix M_c in the upper left corner, which has been updated with Ω, a TA result π_{ta} calculated from the cost matrix, a path diagram in the right corner representing a possible path solution, and the total cost c on the bottom.

H. Within each H, we have a constraint set Ω, a cost matrix M associated with Ω, a TA solution π_{ta}, and the total cost (flowtime) c.

Initially, we create the first node H_1. Conflicts can arise in our initial solution, so we use the first conflict, where agent 1 and agent 2 collide at timestep 2, to establish 2 constraints. Then we create 2 new CT nodes (H_2, H_3) and add these 2 constraints into each constraint set separately and update each cost matrix with the new constraint. The new cost matrix only has one row different from the previous cost matrix. Because the cost matrix changed, we will obtain a new TA result by dynamic Hungarian algorithm. Then we push new H into our priority queue.

In Fig. 3.1, both new nodes H_2 and H_3 have the same total cost. Consider H_2 is first selected from the priority queue for expansion. Two new nodes H_4, H_5 are generated from H_2. Among $\{H_3, H_4, H_5\}$, H_3 has the smallest flowtime and is thus selected for expansion, which leads to H_6, H_7. Now, the priority queue has 4 nodes: $\{H_4, H_5, H_6, H_7\}$, and H_4, H_5, H_7 have the same lowest flowtime 6.

When H_4 is selected for expansion, it has 2 equal TAs: $\{1 \to d, 2 \to c\}$ and $\{1 \to d, 2 \to e\}$. In ITA-CBS, ties are broken at random and consider the case $\{1 \to d, 2 \to e\}$ without losing generality. In this case, there is no conflict, and ITA-CBS returns the solution: $\{1 \to d, 2 \to e\}$ with flowtime is 6, which is an optimal solution.

3.4 Incremental Target Assignment

During the search, when a new constraint is added to an agent i, only the row in the cost matrix corresponding to agent i may change. One can run Hungarian algorithm from scratch based on the new cost matrix to compute the assignment. However, it's too costly for ITA-CBS to execute the algorithm at each CT node. To expedite the computation, we employ the dynamic Hungarian algorithm [25, 37] to reuse previous assignment and quickly update the assignment after cost matrix changes.

Specifically, Hungarian algorithm assigns each vertex i a value $l(i)$ which should satisfy $M_c(u, v) \leq l(u)+l(v)$, where u, v are different vertices, M_c is the cost matrix. A special subgraph is formed that includes all vertices and edges meeting the condition $M_c(u, v) = l(u) + l(v)$. [22] proved that if the special subgraph's matching is a perfect matching, this matching is the optimal matching in original weight graph.

Hungarian algorithm aims to adjust vertex values to achieve a perfect match in the special subgraph. For dynamic Hungarian algorithm, if k rows and columns are changed, these k affected vertexes will be unmatched. Then dynamic Hungarian algorithm will adjust the vertex value $l(i)$ for each affected vertex i, ensuring that $M_c(u, v) \leq l(u)+l(v)$ still holds. The complexity will be $O(kM^2)$ to get a new optimal matching. In ITA-CBS, time complexity is $O(M^2)$ since a new conflict only impacts one row in M_c, which is faster than original Hungarian algorithm with $O(M^3)$.

3.5 Properties of ITA-CBS

This section shows that ITA-CBS is guaranteed to find an optimal TAPF solution if one exists.

Lemma 3.5.1. *The cost of each CT node is a lower bound on the flowtime of all solutions that satisfy the node's constraints.*

Proof Sketch. Since the entries of the cost matrix of a CT node correspond to the shortest paths that ignore collisions, for any solution that satisfies the node's constraints, its flowtime cannot be smaller than the flowtime of its corresponding target assignment. Since we find the best target assignment at each node, its flowtime is a lower bound on the flowtime of all solutions that satisfy the node's constraints. It is easy to prove that the cost of a CT node is equal to the flowtime of its best target assignment. Therefore, the lemma holds. □

Lemma 3.5.2. *Every collision-free set of paths that satisfies the constraints of a CT node must also satisfy at least one of its child nodes' constraints.*

Proof Sketch. We prove by contradiction and assume that there is a collision-free solution $\{p^i\}$ that satisfies the constraints of a node H_x but does not satisfy the constraints of either child node. Suppose the collision chosen to resolve in H_x is between agents i and j at vertex v (or edge e) at timestep t. Since each child node has only one additional constraint compared to node H_x, we know that $\{p^i\}$ violates both additional constraints. That is, both path p^i and path p^j visit vertex v (or edge e) at timestep t, which leads to a collision and contradicts the assumption that $\{p^i\}$ is collision-free. Therefore, the lemma holds. □

Lemma 3.5.3. *At any iteration of the high-level search, every collision-free solution must satisfy at least one CT node's constraints in the OPEN list.*

Proof Sketch. Since the root CT node has no constraints, all solutions satisfy the constraints of the root CT node. When we pop a CT node from the OPEN list, we will insert its child nodes back into the OPEN list. According to Lemma 3.5.3, this lemma holds. □

Theorem 3.5.4. *ITA-CBS guarantees to find an optimal TAPF solution if exists.*

Proof Sketch. According to Lemmas 3.5.1 and 3.5.3, the cost of the CT node with the smallest cost in the OPEN list is a lower bound on the flowtime of all collision-free solutions. Therefore, when ITA-CBS terminates, its returned solution is guaranteed to be optimal. □

4 Results and Discussion

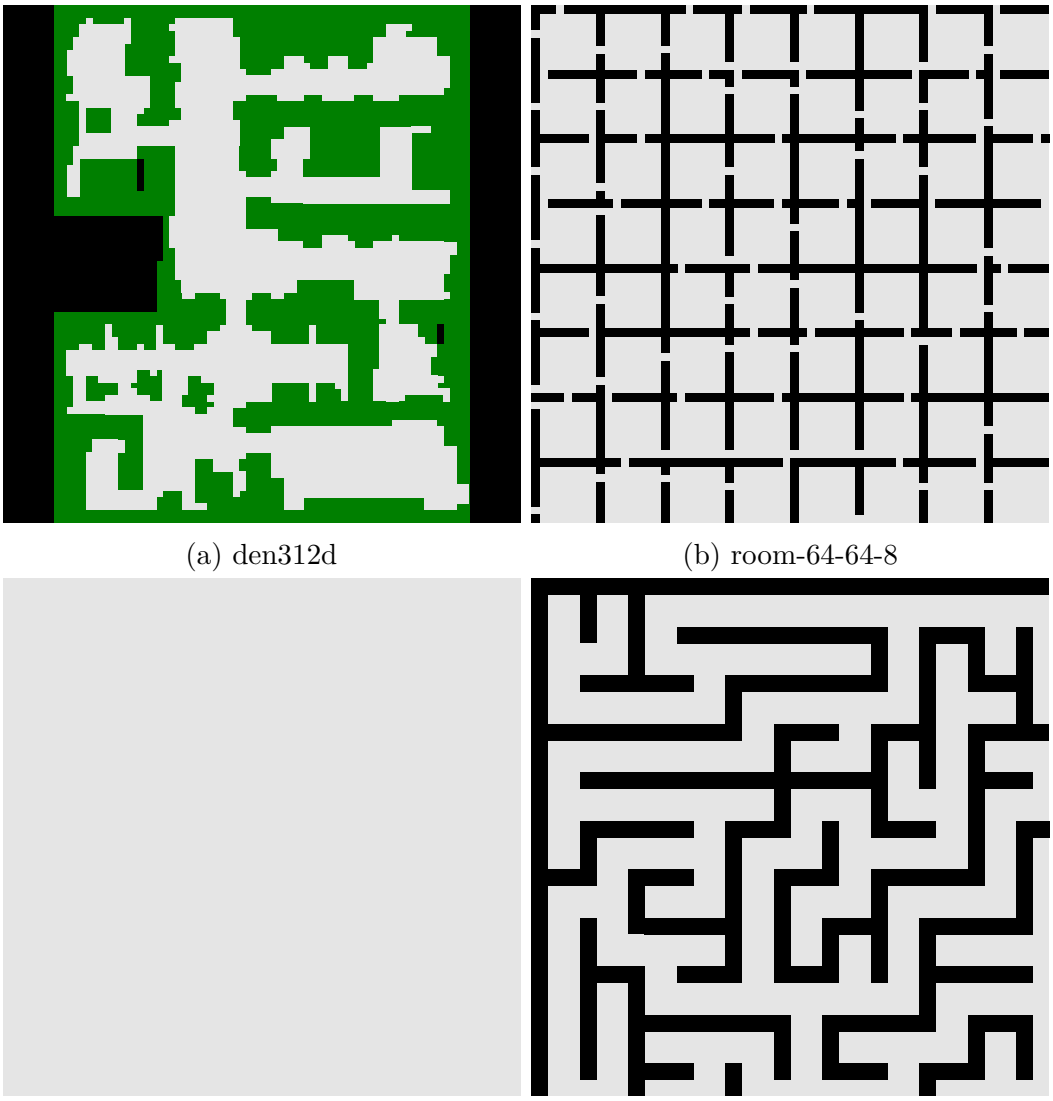
We evaluate the performance of ITA-CBS and CBS-TA. We implement ITA-CBS and CBS-TA in C++ based on the existing CBS-TA implementation.[1] Our CBS-TA implementation outperforms the original based on our tests. To our best knowledge, CBS-TA is the only existing work that solves TAPF optimally for flowtime, and thus we only compare ITA-CBS with CBS-TA in our experiments. We classify a testcase as a failure if no solution is found within 30 seconds and we mark the runtime for this testcase as 30 seconds. All experiments were executed on a computer with Ubuntu 20.04.1, AMD Ryzen 3990X 64-Core Processor, 64G RAM with 2133 MHz.

We use 8 different maps from MAPF Benchmark Sets [38]: (1) den312d is from video game Dragon Age Origins (DAO), (2) random-32-32-10 and empty-32-32 are open grids with and without random obstacles, (3) maze-32-32-2 is a maze-like grid, (4) room-64-64-8 is a room-like grid, (5) warehouse-10-20-10-2-1 is inspired by real-world autonomous warehouse applications and (6) orz900d and Boston-0-256 are the first and second largest maps among all benchmark map files. All maps are shown in Figure 4.1 and Figure 4.2.

4.1 Test Scenarios

We develop 2 test scenarios: (1) Group Test: We divide all agents into groups, and each group shares the same target location set. (2) Common Target Test: Each agent receives a target set of equal size. All agents have some common target locations. We evaluate the performance of the ITA-CBS and CBS-TA algorithms by altering the proportion of common targets in target sets. In each testcase, we randomly select the start and goal locations for every agent. We generate a set of 20 testcases for a given map using a specific test configuration. The success rate is calculated as the percentage of completed tests out of the total 20 test cases.

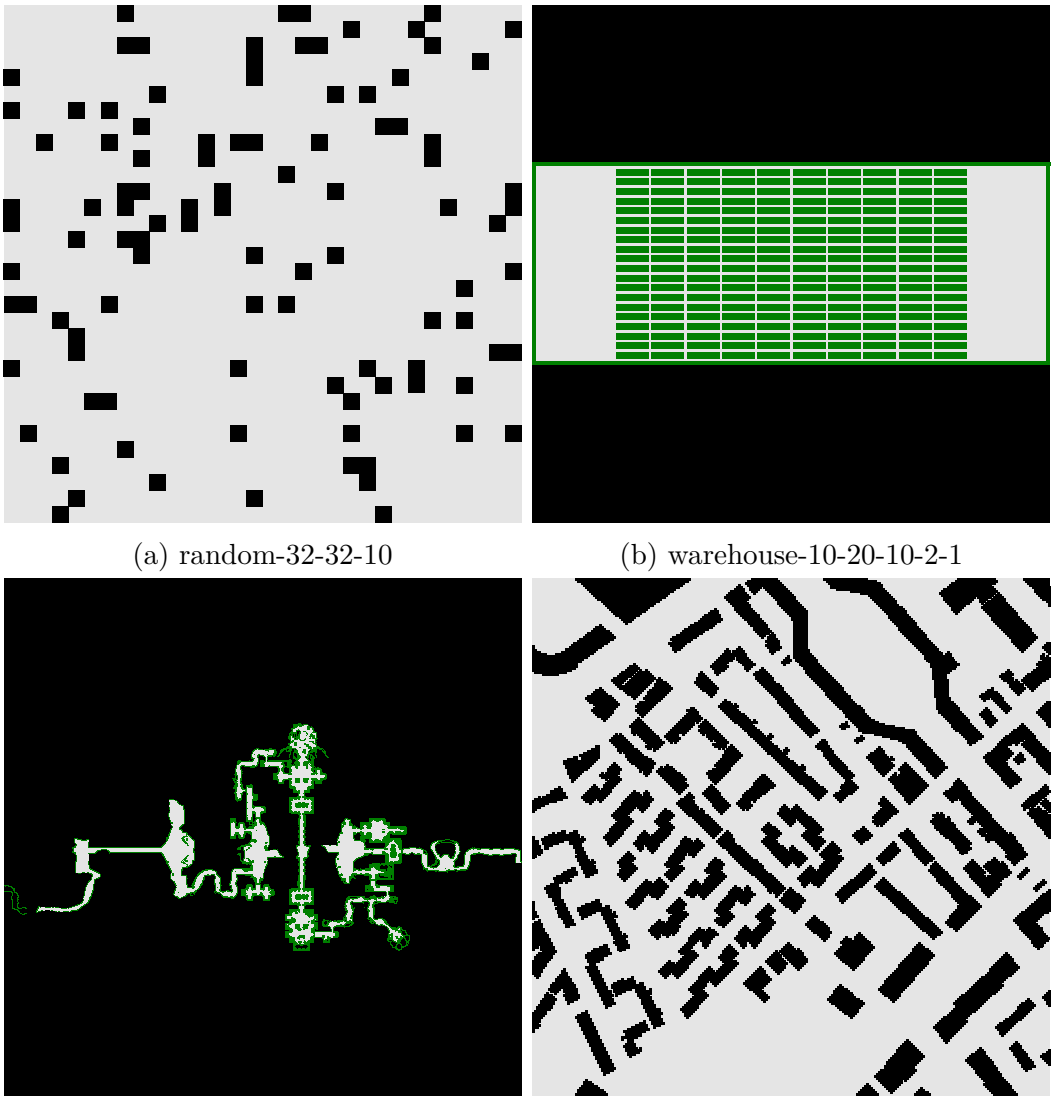
[1]The CBS-TA source code is publicly available at `https://github.com/whoenig/libMultiRobotPlanning`. We will open source our code at `https://github.com/TachikakaMin`.



(a) den312d (b) room-64-64-8

(c) empty-32-32 (d) maze-32-32-2

Figure 4.1: (1) den312d is from video game Dragon Age Origins (DAO), (2) random-32-32-10 is open grids with random obstacles, (3) empty-32-32 is open grids without any obstacle, (4) maze-32-32-2 is a maze-like grid



(a) random-32-32-10

(b) warehouse-10-20-10-2-1

(c) orz900d

(d) Boston-0-256

Figure 4.2: (1) room-64-64-8 is a room-like grid, (2) warehouse-10-20-10-2-1 is inspired by real-world autonomous warehouse applications, (3) orz900d is the largest map among all benchmark map files, (4) Boston-0-256 is a city-like grid and is the second largest map in mapf benchmark dataset

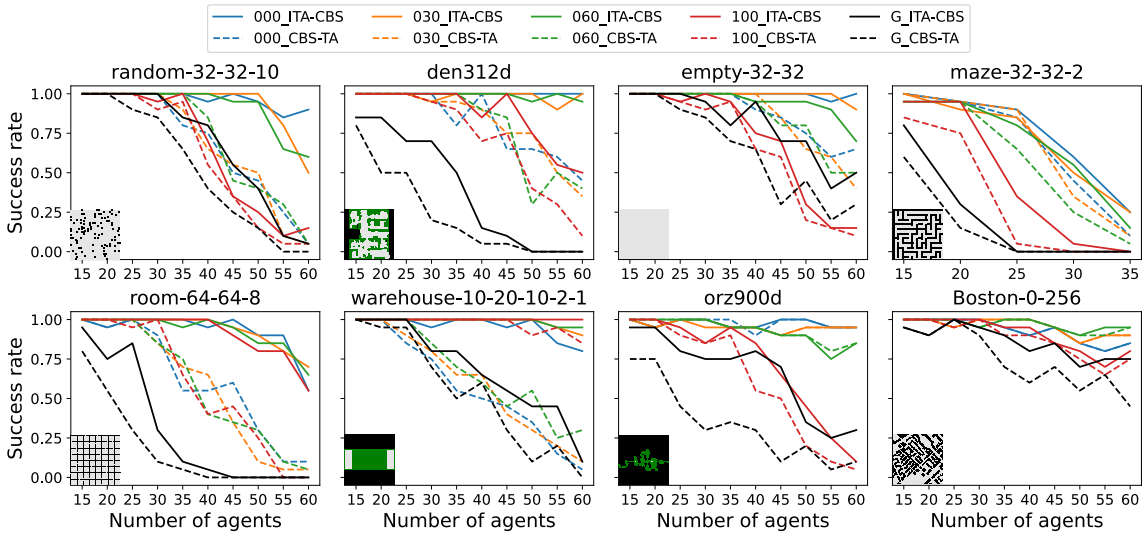


Figure 4.3: Result for group test and common target test. G_ITA-CBS and G_CBS-TA represent group test result using black lines. Others are common target test results. Numbers before algorithm names in legends are common target percentages. 000 represents there is no common target for all agents and 100 represents all agents share the same target set. The map is in the bottom left corner of each subgraph. The X-axis is agent number and the Y-axis is success rate of algorithms. For most test scenarios, ITA-CBS outperforms CBS-TA.

4.1.1 Group Test

In this test scenario, we put every 5 agents into one group, and the agents in each group share 5 different target locations. Different groups have different target locations. We increase the agent number with 5 intervals and all numbers can be found in Figure 4.3. Since groups do not share the same target locations, testcases grow increasingly complicated as the number of agents increases. The black lines reflect the success rates of both algorithms. Figure 4.3 shows that ITA-CBS outperformed CBS-TA on all test maps.

4.1.2 Common Target Test

In this test scenario, we give each agent one target set with a fixed size and adjust the proportion of common targets. The size of the fixed target set is determined by dividing the total valid grid count of the map by the maximum number of agents. For the maze map, agent numbers vary from 15 to 35 with an increment of 5. For other maps, the agent number is from 15 to 60 with an increment of 5. Correspondingly, the target set sizes are {15, 15, 80, 40, 15, 50, 20, 20} for {empty, random, warehouse, den312d, maze, room, orz900d, boston}. The percentage of common targets among all targets are: 0, 30%, 60% and 100%. Figure 4.3 shows that as common targets increase, the total success rates decrease, and ITA-CBS outperformed the CBS-TA under most proportions.

4.2 Test Overall Situation

We also show all testcases in Figure 4.4. The X-axis represents ITA-CBS running time in seconds, and the Y-axis represents CBS-TA algorithm running time. We have a total of 7,600 testcases, including 5,134 testcases both algorithms solved, 1,191 testcases ITA-CBS solved only, 9 testcases CBS-TA solved only and 1266 testcases both algorithms failed.. For the 6,334 effective testcases which are solved by at least one algorithm, ITA-CBS is faster in 96.1% testcases, 5 times faster in 38.7% testcases, and 100 times faster in 5.6% testcases than CBS-TA.

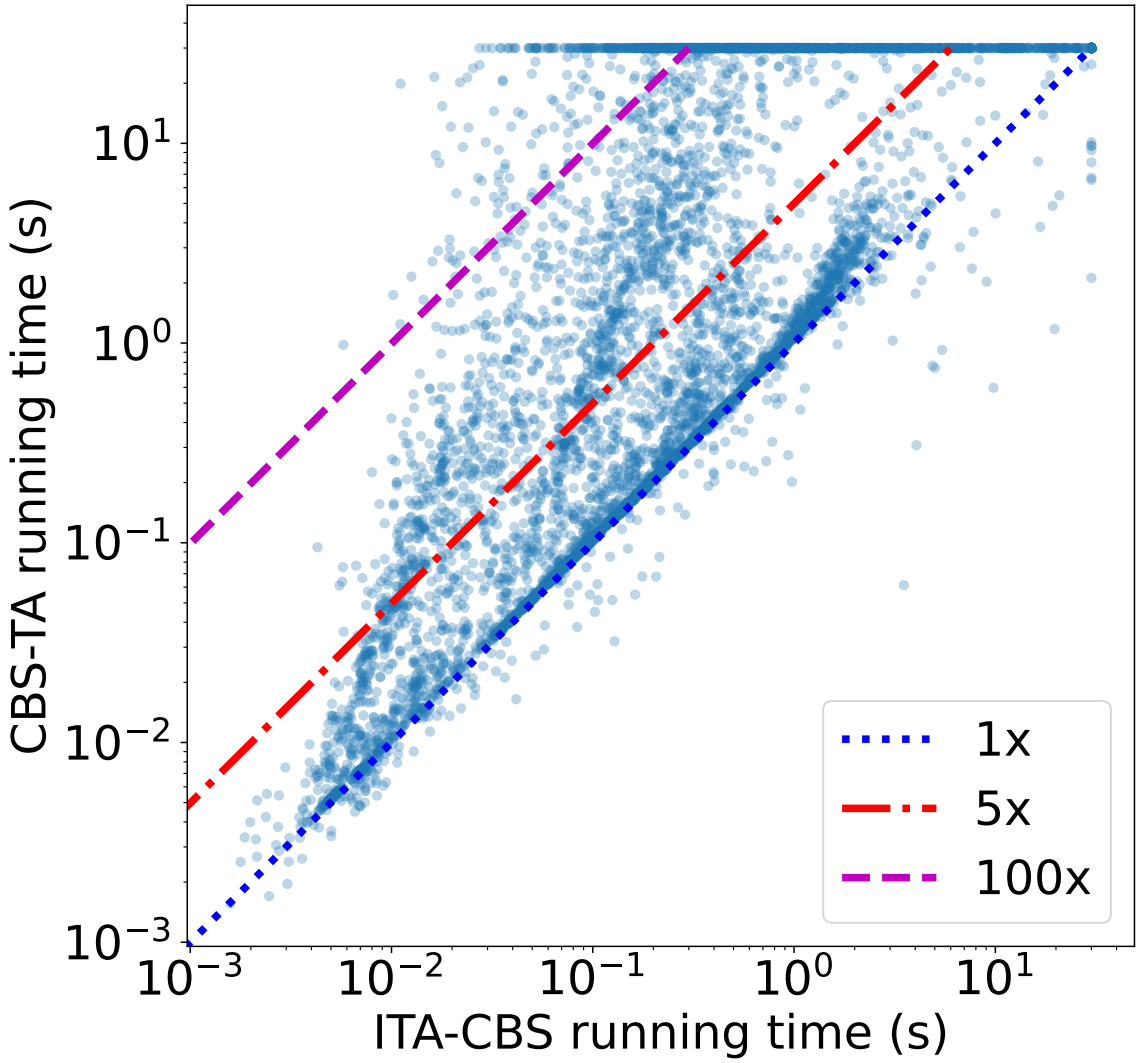


Figure 4.4: All testcases running time for ITA-CBS and CBS-TA. The X-axis represents ITA-CBS running time in seconds, and the Y-axis represents CBS-TA algorithm running time. We record their running time as 30s for timeout testcases, so there is a line in the figure top. ITA-CBS is faster in 96.1% testcases, 5 times faster in 38.7% testcases, and 100 times faster in 5.6% testcases than CBS-TA among 6,334 effective testcases.

4.3 Program Profile

For this section, all time and CT node number related data are from the previous 2 scenarios' test data. For this test, we only use 5,134 testcases in which both algorithms successfully find optimal solutions within the given runtime limit and take the average of these data.

4.3.1 Running Time of Various Parts

Now we show the average running time for various parts of each algorithm program in Figure 4.5. We divide the program running time into 4 parts: time of target assignment, time of low-level path search, and time of collision detaction and other time. The average time for CBS-TA and ITA-CBS are $\{1.2s, 0.51s, 0.22s, 0.058s\}$ and $\{0.006s, 0.36s, 0.032s, 0.027s\}$. This result shows that our dynamic Hungarian algorithm largely reduced the time taken by target assignment. Because ITA-CBS and CBS-TA may have different numbers of CT nodes which may result in an unfair comparison of target assignment, we also show their target assignment average runtime in Figure 4.6. The figure shows that ITA-CBS is an order of magnitude faster than CBS-TA. And for time of collision detaction, since this action will be invoked for each CT node, the result matches the CT node numbers in Figure 4.6.

4.3.2 The number of Constraint Tree (CT) nodes and CTs

Figure 4.6 also shows the numbers of CT nodes and Constraint Trees(CTs) for each test case. CBS-TA runs target assignment only when it needs a new CT, and ITA-CBS runs it in every CT node update. So we compare the number of ITA-CBS CT nodes with CBS-TA's numbers of CTs and CT nodes. The result shows that even comparing the number of ITA-CBS CT nodes with CBS-TA CTs, ITA-CBS has fewer target assignment than CBS-TA, which can imply constraints in low-level search can reduce target assignment search space. We also found the ratio of the root node number and total CT node number may be very high for CBS-TA. For all 5,134 testcases, the ratio will be 37.7% with 2226 mean TA times compare with ITA-CBS 862 times. This result explains CBS-TA has a very large TA partition in

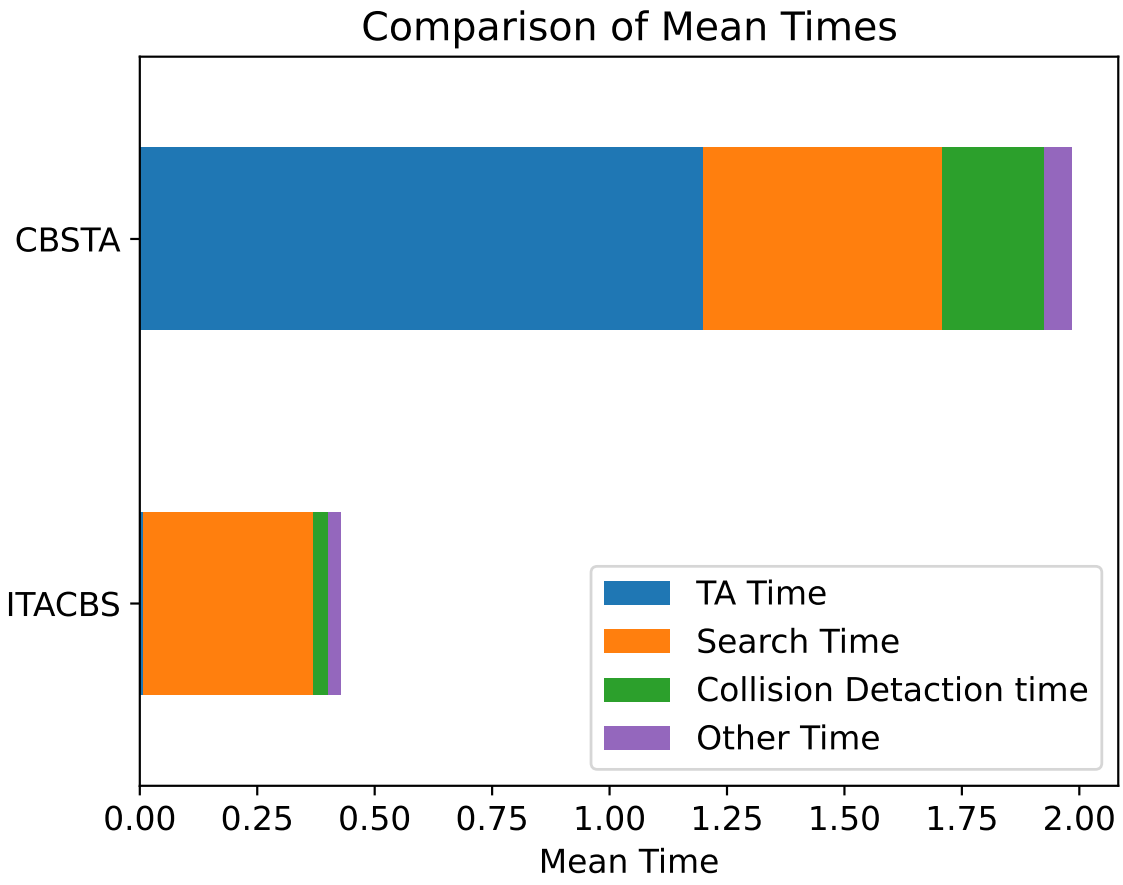


Figure 4.5: TA Time (time of target assignment), Search Time (time of low-level shortest path search), Collision detection Time and Other Time. The average time for CBS-TA and ITA-CBS are $\{1.2s, 0.51s, 0.22s, 0.058s\}$ and $\{0.006s, 0.36s, 0.032s, 0.027s\}$.

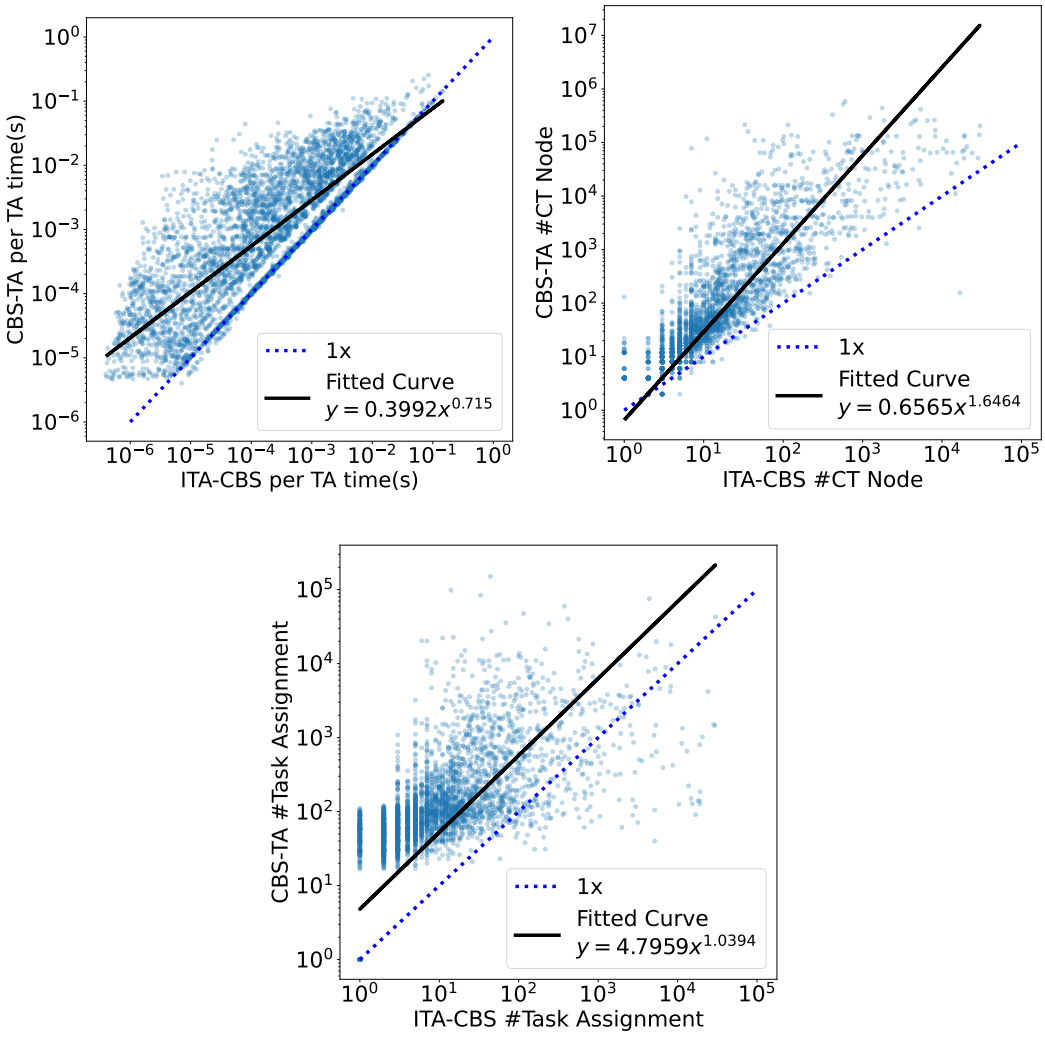


Figure 4.6: Left subfigure: X-axis is ITA-CBS per TA time and Y-axis is CBS-TA per TA time, and almost all tests ITA-CBS dynamic Hungarian algorithm outperforms CBS-TA SSP algorithm. Middle subfigure: X-axis is ITA-CBS CT node number and Y-axis is CBS CT node number. Right subfigure: X-axis is ITA-CBS TA times equal to total CT node number and Y-axis is CBS-TA root node number which is equal to its TA times. This shows CBS has more root CT nodes and requires more calls to the TA algorithm.

total runtime.

4.4 Limitation

In this section, we address certain limitations observed during our experiments. A predominant issue shared by all CBS algorithms is that the selection order of CT nodes with identical costs can significantly impact the overall performance. In all experiments referenced earlier, we employed the default comparison function from the Boost library for CT nodes with equal costs. Subsequently, we will introduce two distinct comparison functions and their respective experimental results to demonstrate the profound impact of selection order on the outcomes. Firstly, every CT node is assigned an increasing index number upon its creation. The initial comparison function prioritizes CT nodes with smaller index numbers when two nodes share the same cost, this function is termed "bfs". Conversely, the second function selects the CT node with the larger index number, and we refer to it as "dfs".

We present the running times of two ITA-CBS versions using different comparison functions across 7600 test cases in Section 4.4. Clearly, the dfs version outperforms the bfs version. When examining the CT node counts for these two versions in Section 4.4, we observed that in some test cases, the bfs version has an order of magnitude (two to three times more) CT nodes than the dfs version. This illustrates that the search order profoundly influences the number of CT nodes, leading to notable disparities in the final runtime performance.

Another limitation of ITA-CBS, when contrasted with CBS-TA, is the necessity to store a cost matrix within each CT node. A new TA solution must be recalculated in each CT node. Common sub-optimal CBS variants, like ECBS or ECBS-TA, utilize epsilon a-star (ϵ-A^* search) integrated with an auxiliary score function for bounded sub-optimal shortest path searches. This scoring mechanism primarily hinges on the conflict number between the new path and pre-existing ones. Every CT node in ITA-CBS has its own TA solution. This makes using the regular ϵ-A^* search to find the best path difficult. Because TA solutions can change, old conflict numbers might be based on out-of-date TA solutions. This is a big problem and should be considered when creating a new algorithm based on ITA-CBS.

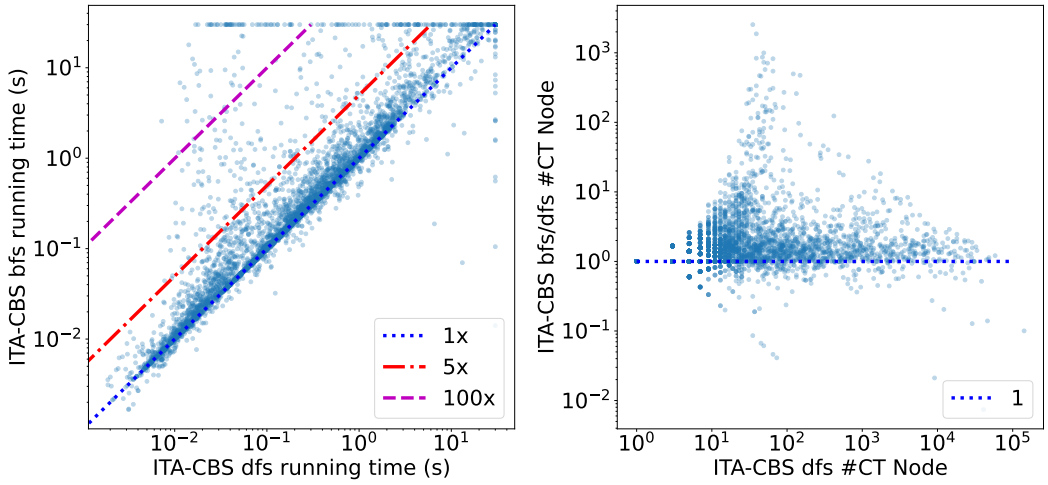


Figure 4.7: Left subfigure: The X-axis represents the running time of the ITA-CBS dfs version, while the Y-axis denotes the running time of the ITA-CBS bfs version. Each point corresponds to a single testcase. Right subfigure: The X-axis indicates the number of CT nodes in the ITA-CBS dfs version, and the Y-axis represents the ratio of CT node numbers between the bfs and dfs versions. A ratio greater than one suggests that the bfs version has more CT nodes, whereas a ratio less than one indicates that the dfs version has more CT nodes.

5 Conclusion and Future Work

5.1 Conclusion and Future Work

This work develops a new algorithm called Incremental Target Assignment CBS (ITA-CBS) to solve the TAPF problem to optimality. We show that our algorithm (1) avoids duplicate effort in conflict resolution and (2) updates target assignment incrementally, thus leading to guarantees of optimality as well as efficient computation, as attensted by our experimental results. ITA-CBS differs from the previous leading algorithm CBS-TA in the following 2 aspects. First, ITA-CBS creates only a single constraint tree during the search and is thus able to avoid duplicated conflict resolution in different trees as in CBS-TA. Second, ITA-CBS avoids solving the K-best assignment problem, and instead, ITA-CBS updates the target assignment in an incremental manner during the CBS-like search, which further reduces the computational effort. We show that ITA-CBS is guaranteed to find an optimal solution to TAPF and verify the algorithm with extensive tests. The numerical results show that our ITA-CBS is faster in 96.1% testcases, 5 times faster in 38.7% testcases, and 100 times faster in 5.6% testcases than CBS-TA in 6,334 effective testcases.

In terms of future developments, we intend to incorporate parallel techniques into our ITA-CBS algorithm. Given that ITA-CBS operates on a single binary constraint tree, where each CT node is only associated with its parent node, it presents a ripe opportunity for parallelism. Additionally, we propose to explore the development of a suboptimal version of ITA-CBS and investigate the application of various optimization methods within our algorithm. These proposed advancements hold the potential to extend the application of ITA-CBS to realistic, complex scenarios. Specifically, they could aid in the planning and navigation of robots operating in dynamic and uncertain environments, such as warehouses.

Bibliography

[1] T. S. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[2] J. Yu and S. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 27, pp. 1443–1449, 2013.

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

[4] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," *arXiv preprint arXiv:1612.05693*, 2016.

[5] W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, (Richland, SC), p. 757–765, International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[6] F. Ho, A. Goncalves, A. Salta, M. Cavazza, R. Geraldes, and H. Prendinger, "Multi-agent path finding for uav traffic management: Robotics track," 2019.

[7] J. Hagelbäck, "Hybrid pathfinding in starcraft," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 8, no. 4, pp. 319–324, 2015.

[8] 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 11272–11281, 2021.

[9] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of artificial intelligence research*, vol. 31, pp. 591–656, 2008.

[10] H. Ma, C. Tovey, G. Sharon, T. Kumar, and S. Koenig, "Multi-agent path finding with payload transfers and the package-exchange robot-routing problem," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[11] C. R. Chegireddy and H. W. Hamacher, "Algorithms for finding k-best perfect matchings," *Discrete applied mathematics*, vol. 18, no. 2, pp. 155–165, 1987.

[12] K. G. Murty, "An algorithm for ranking all the assignments in order of increasing cost," *Operations research*, vol. 16, no. 3, pp. 682–687, 1968.

[13] A. Geramifard, P. Chubak, and V. Bulitko, "Biased cost pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 2, pp. 112–114, 2006.

[14] D. Silver, "Cooperative pathfinding," in *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, vol. 1, pp. 117–122, 2005.

[15] S. Varambally, J. Li, and S. Koenig, "Which mapf model works best for automated warehousing?," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 15, pp. 190–198, 2022.

[16] A. Andreychuk, K. Yakovlev, P. Surynek, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," *Artificial Intelligence*, vol. 305, p. 103662, 2022.

[17] R. J. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[18] K.-H. C. Wang, A. Botea, *et al.*, "Fast and memory-efficient multi-agent pathfinding.," in *ICAPS*, pp. 380–387, 2008.

[19] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artificial intelligence*, vol. 219, pp. 1–24, 2015.

[20] D. Du and P. M. Pardalos, *Handbook of combinatorial optimization*, vol. 4. Springer Science & Business Media, 1998.

[21] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[22] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[23] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows," 1988.

[24] G. R. Waissi, "Network flows: Theory, algorithms, and applications," 1994.

[25] G. A. Mills-Tettey, A. Stentz, and M. B. Dias, "The dynamic hungarian algorithm for the assignment problem with changing costs," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-27*, 2007.

[26] Z. Ren, S. Rathinam, and H. Choset, "Cbss: A new approach for multiagent combinatorial path finding," *IEEE Transactions on Robotics*, pp. 1–15, 2023.

[27] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," in *Proceedings of the 2016 International Conference on Autonomous Agents Multiagent Systems*, AAMAS '16, (Richland, SC), p. 1144–1152, International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[28] C. Henkel, J. Abbenseth, and M. Toussaint, "An optimal algorithm to solve the combined task allocation and path finding problem," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4140–4146, 2019.

[29] X. Zhong, J. Li, S. Koenig, and H. Ma, "Optimal and bounded-suboptimal multi-goal task assignment and path finding," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10731–10737, 2022.

[30] V. Nguyen, P. Obermeier, T. C. Son, T. Schaub, and W. Yeoh, "Generalized target assignment and path finding using answer set programming," in *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[31] Z. Ren, S. Rathinam, and H. 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)*, pp. 11560–11565, 2021.

[32] P. Surynek, "Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 12, pp. 197–199, 2021.

[33] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5816–5823, 2021.

[34] X. Zhong, J. Li, S. Koenig, and H. Ma, "Optimal and bounded-suboptimal multi-goal task assignment and path finding," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 10731–10737, IEEE, 2022.

[35] K. Okumura and X. Défago, "Solving simultaneous target assignment and path planning efficiently with time-independent execution," *Artificial Intelligence*, p. 103946, 2023.

[36] Z. Ren, S. Rathinam, and H. Choset, "A conflict-based search framework for multiobjective multiagent path finding," *IEEE Transactions on Automation Science and Engineering*, vol. 20, no. 2, pp. 1262–1274, 2023.

[37] I. Z. Amalia, A. Saikhu, and R. Soelaiman, "A fast dynamic assignment algorithm for solving resource allocation problems," *Jurnal Online Informatika*, vol. 6, no. 1, pp. 118–127, 2021.

[38] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *Symposium on Combinatorial Search (SoCS)*, pp. 151–158, 2019.