Collaborative Task Assignment, Sequencing and Multi-agent Path-finding

Yifan Bai, Shruti Kotpalliwar, Christoforos Kanellakis and George Nikolakopoulos

***Abstract*— In this article, we address the problem of collaborative task assignment, sequencing, and multi-agent pathfinding (TSPF), where a team of agents must visit a set of task locations without collisions while minimizing flowtime. TSPF incorporates agent-task compatibility constraints and ensures that all tasks are completed. We propose a Conflict-Based Search with Task Sequencing (CBS-TS), an optimal and complete algorithm that alternates between finding new task sequences and resolving conflicts in the paths of current sequences. CBS-TS uses a mixed-integer linear program (MILP) to optimize task sequencing and employs Conflict-Based Search (CBS) with Multi-Label A* (MLA*) for collision-free path planning within a search forest. By invoking MILP for the next-best sequence only when needed, CBS-TS efficiently limits the search space, enhancing computational efficiency while maintaining optimality.**

We compare the performance of our CBS-TS against Conflict-based Steiner Search (CBSS), a baseline method that, with minor modifications, can address the TSPF problem. Experimental results demonstrate that CBS-TS outperforms CBSS in most testing scenarios, achieving higher success rates and consistently optimal solutions, whereas CBSS achieves near-optimal solutions in some cases. The supplementary video is available at `https://youtu.be/QT8BYgvefmU`.

I. Introduction

Deploying multiple autonomous robots in domains such as warehouse pick-up and delivery [1] or search and rescue operations [2] involves addressing several challenges, among which task assignment [3] and multi-agent pathfinding (MAPF) [4] play a critical role. Task assignment determines which agents should handle which tasks, while MAPF ensures agents navigate from their start locations to task destinations without collisions. If the objective is to achieve overall optimality (e.g., minimizing flowtime), these two problems are tightly coupled, as certain task assignments can lead to fewer collisions during path planning. For instance, Figure 1a illustrates a task sequencing that initially minimizes flowtime to 5 without accounting for potential collisions. In this sequence, agent 1 is assigned tasks g_1, and agent 2 is assigned g_3 followed by g_2. However, resolving conflicts in the resulting paths requires agent 1 to wait at cell a for 2 time steps, increasing the total flowtime to 7. In contrast, Figure 1b presents an alternative sequencing result with a higher initial flowtime of 6, but it remains conflict-free, leading to a better overall solution. This demonstrates that task assignment and path planning must be jointly considered to achieve optimal execution.

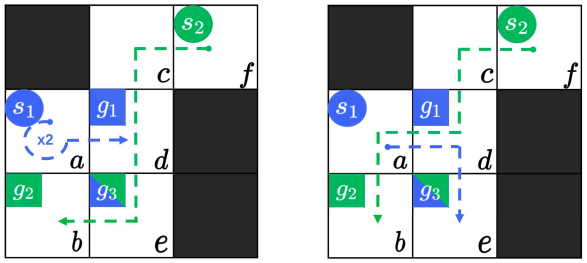


Fig. 1: An example demonstrating how different task assignments result in different conflict-free paths for two agents (starting at s_1, s_2) and three tasks (at g_1, g_2, g_3), with dashed lines representing the planned conflict-free paths. The circular segment in the blue path represents a wait action required to resolve conflicts. The black cells denote obstacles. The color of the starts and goals indicates the agent-task compatibility constraints (i.e. g_3 can be visited by either the green or blue agent).

In this article, we address the collaborative task sequencing and pathfinding (TSPF) problem and propose an optimal solution that minimizes flowtime while ensuring all tasks are completed and agent-task compatibility constraints are satisfied.

A. Related Work

Task Assignment: Chakraa et al. [3] provided a comprehensive taxonomy of multi-robot task assignment problems. Related problems include the Multiple Traveling Salesman Problem (mTSP) [5], where agents must visit a set of locations in an optimal sequence, and the Vehicle Routing Problem (VRP) [6], a variant of mTSP with additional constraints. A wide range of mTSP solvers has been developed, including exact algorithms [7], [8] for optimal solutions and heuristic methods [9], [10] for near-optimal solutions. TSPF reduces to mTSP when salesmen start from different depots (i.e., agent start positions), are not required to return to their depots, and conflicts between salesmen are ignored.

Multi-agent path finding (MAPF): A variety of methods have been proposed to solve MAPF [11]–[13]. Among them, Conflict-Based Search (CBS) [14] has gained prominence due to its optimality and completeness. However, solving MAPF optimally is NP-hard [15], motivating research into enhancements [16]–[18] that improve CBS's efficiency. Ad-

The authors are with the Robotics and AI group, in the Department of Computer Science, Electrical and Space Engineering at Luleå University of Technology, Sweden.

This work has been partially funded by the European Union's Horizon Europe Research and Innovation Programme under the Grant Agreement No.101138451 PERSEPHONE.

ditionally, several suboptimal MAPF solvers [19], [20] have been developed to achieve faster runtimes. Since our method builds on CBS, these optimizations and suboptimal variants can also be applied to improve its efficiency.

Combined task assignment and path-finding (TAPF): TAPF [21] extends MAPF by integrating task allocation, where the number of single-goal tasks matches the number of agents, and each agent is assigned exactly one task. One leading approach CBS-TA [22] guarantees optimality through a search forest but assigns only one goal per agent, leaving some goals unvisited. Extensions like [23] model tasks as sequences of fixed-ordered goals, ensuring complete goal assignment but preventing goals within a task from being assigned to different agents. Multi-Goal MAPF (MG-MAPF) [24] does not perform task assignment; instead, each agent is given a predefined set of goals to visit. While MG-MAPF allows agents to visit their assigned goals in arbitrary order and guarantees optimality, it does not permit goals to be assigned across different agents. The Multi-Agent Pickup and Delivery (MAPD) [1] problem addresses cases where agents must complete tasks that involve visiting two distinct locations, such as picking up and delivering an item. Henkel et al. [25] solve MAPD optimally via brute-force enumeration, but their paper only reports results for up to four tasks, leaving its scalability unclear. PC-TAPF [26] incorporates precedence constraints between tasks, but its low-level path planner lacks completeness guarantees, hindering robustness in complex scenarios. Recent work, CBSS [27], supports full goal coverage and allows any goal to be assigned to any agent while being theoretically optimal. However, its implementation remains computationally inefficient, even with a heuristic solver that does not guarantee optimality.

While existing methods tackle various aspects of task assignment and pathfinding, they face critical limitations, such as incomplete goal coverage [22], [28], rigid goal assignment within predefined sets [23], [24], suboptimal [29], [30] or incomplete [26] solutions, and computational inefficiencies [25], [27]. These limitations highlight the need for a scalable and optimal approach for TSPF, which guarantees all goals are visited while allowing goals to be assigned freely, subject to agent-task constraints.

B. Contribution

In this paper, we propose CBS-TS, an efficient, optimal, and complete approach to the TSPF problem. CBS-TS ensures that all goal locations are visited while allowing any goal to be assigned to any agent, provided the agent-task constraints permit it.

CBS-TS employs a two-level hierarchical approach. At the high level, the algorithm constructs a search forest by iteratively alternating between two processes: (1) identifying the next-best task sequence (root node of a new search tree) and (2) resolving conflicts to generate collision-free paths for the given task sequence (exploring the solution in a search tree). At the low level, CBS-TS leverages Multi-label A* (MLA*) [31] to compute collision-free paths for agents to visit their assigned tasks in the specified sequence.

A key contribution of this work is the introduction of a Mixed-Integer Linear Programming (MILP) formulation to efficiently determine the next-best task sequence. We provide theoretical proof demonstrating that CBS-TS is both optimal and complete.

Through extensive experiments, we compare CBS-TS with the CBSS algorithm, demonstrating its better performance in terms of effectiveness and efficiency. Additionally, we validate the practicality of our approach through physical experiments, further underscoring its applicability in real-world scenarios.

II. Problem Description

Consider an undirected graph $G = (V, E)$, where V represents a set of locations and $E \subseteq V \times V$ represents connections between these locations. We are given a set of N heterogeneous agents $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$, each starting from an initial location in $\mathcal{S} = \{s_1, s_2, \ldots, s_N\} \subseteq V$, where s_i denotes the initial location of agent a_i. Additionally, there are M goal locations, $\mathcal{G} = \{g_1, g_2, \ldots, g_M\} \subseteq V$, each of which must be visited by an agent. In this work, tasks refer to the requirement for agents to visit specific goal locations in the environment.

To model the agent-task compatibility constraints, we define a binary compatibility matrix Δ of size $N \times M$, where each entry $\delta_{ij} \in \{0, 1\}$ indicates whether agent a_i is permitted to reach goal g_j. Specifically, $\delta_{ij} = 1$ if agent a_i is allowed to reach goal g_j, and $\delta_{ij} = 0$ otherwise.

All agents operate under a global clock. At each discrete time step, an agent can either wait at its current vertex or move to an adjacent vertex. The objective is to determine a task sequencing and corresponding paths $\pi_i = \left(v_i^0, v_i^1, \ldots, v_i^{T_i}\right)$ for each agent a_i, such that the following conditions are satisfied: noitemsep

1) **Initial Condition:** Each agent a_i starts at its designated initial location, i.e. $v_i^0 = s_i, s_i \in \mathcal{S}$;
2) **Terminal Condition:** If agent a_i assigned tasks, all goal locations in its assigned task sequence must satisfy the agent-task compatibility constraints, and a_i stops at the last goal location in that task sequence and remains there, i.e. $v_i^{T_i} = g_j, g_j \in \mathcal{G}, \delta_{ij} = 1, v_i^t = g_j, \forall t > T_i$
   If not assigned any tasks, agent a_i stays at its starting location, i.e. $v_i^t = s_i, \forall t$;
3) **Goal Coverage:** Each goal location $g_j \in \mathcal{G}$ must be visited exactly once by an agent a_i, provided $\delta_{ij} = 1$, i.e. $\forall g_j \in \mathcal{G}, \quad \exists i, t : v_i^t = g_j \wedge \delta_{ij} = 1$;
4) **Conflict-Free Paths:** The paths of all agents must be conflict-free, ensuring:
   $v_i^t \neq v_j^t, \forall t, \forall i \neq j$ (no vertex conflict)
   $\left(v_i^t, v_i^{t+1}\right) \neq \left(v_j^{t+1}, v_j^t\right), \forall t, \forall i \neq j$ (no edge conflict);
5) **Objective:** The flowtime $\sum_{i=1}^{N} T_i$ is minimized.

Remark: Our method can be easily adapted to minimize the makespan $\max_{1 \leq i \leq N} T_i$ by modifying the objective function

in the MILP formulation. However, in this paper, we focus solely on minimizing flowtime to ensure a fair comparison with the baseline method CBSS under the same metric.

III. Methodology

In this section, we propose CBS-TS (Conflict-Based Search with Task Sequencing), a two level method inspired from CBS-TA [22] but designed to solve TSPF problems. At the high level, CBS-TS extends CBS framework by incorporating optimal task sequencing through a Mixed-Integer Linear Programming (MILP) model, ensuring all goal locations are visited. At the low level, CBS-TS utilizes MLA* to plan collision-free paths for agents to visit their specific task sequences.

A. High Level of CBS-TS

CBS-TS builds upon CBS [14], which uses a binary tree to resolve conflicts and find collision-free paths for MAPF. CBS-TS incorporates task sequencing through a search forest, where each tree corresponds to the CBS conflict resolution of a distinct task sequencing. Each node in the forest consists of: (1) A boolean flag *root* that indicates whether the node is a root node, (2) The task *sequencing* associated with the tree, (3) A set of *constraints*, where vertex constraint $\langle a_i, u, t\rangle$ prevents agent a_i from occupying location u at time t and edge constraint $\langle a_i, u, v, t\rangle$ prohibits agent a_i from moving from u to v at time t, (4) A set of *paths* for all agents that satisfy the constraints, and (5) *Cost*, measured as the total flowtime of all paths.

The pseudocode for CBS-TS is provided in Algorithm 1. The algorithm starts by generating the root node of the first tree. This node contains the best task sequencing, solved by MILP (Section III-B), while ignoring conflicts among agents. Initial paths for each agent are then computed using MLA* based on the assigned task sequencing, without any vertex/edge constraints. This root node is added to the OPEN list [Lines 1–9]. During execution, the algorithm iteratively selects the lowest cost node P from the OPEN list [**Line** 11]. If paths in P are conflict-free, the algorithm returns the solution [**Lines** 12-13]. Otherwise, the earliest conflict (vertex or edge) is identified and resolved by generating two child nodes, each inheriting the parent's constraints and adding a new constraint derived from the conflict. The path of the constrained agent is replanned using MLA* [**Lines** 25-33].

To explore alternative task sequencings, CBS-TS creates a new root node R' with the next best task sequencing before expanding the child nodes of the current root node [**Lines** 14-23]. The next-best sequencing is determined by the MILP solver while excluding previously found sequencings in $\mathcal{P}_{\text{set}}$ [**Line** 16]. By conducting the best-first search across the entire search forest, it is ensured that the globally optimal flowtime solution is found among all possible task sequencing options.

Algorithm 1 High Level of CBS-TS

Input: N, M, cost matrix C, Δ
Output: optimal path for each agent
1: $\mathcal{P}_{\text{set}} \leftarrow \emptyset$ // previously found solution set
2: R.root $\leftarrow$ *true*
3: R.sequencing $\leftarrow$ MILP$(N, M, C, \Delta, \mathcal{P}_{\text{set}})$
4: Insert R.sequencing to $\mathcal{P}_{\text{set}}$
5: R.constraints $\leftarrow \emptyset$
6: **for** $i \leftarrow 1$ to N **do**
7: R.paths$[a_i] \leftarrow$ MLA$^*(a_i, R.\text{sequencing}[a_i], R.\text{constraints})$
8: R.cost $\leftarrow$ flowtime$(R.\text{paths})$
9: Insert R to OPEN
10: **while** OPEN $\neq \emptyset$ **do**
11: $P \leftarrow$ best node from OPEN // lowest solution cost
12: **if** P.paths do not have collisions **then**
13: **Return** P.paths
14: **if** P.root is *true* **then**
15: R'.root $\leftarrow$ *true* // new root node (next best sequencing)
16: R'.sequencing $\leftarrow$ MILP$(N, M, C, \Delta, \mathcal{P}_{\text{set}})$
17: Insert R'.sequencing to $\mathcal{P}_{\text{set}}$
18: R'.constraints $\leftarrow \emptyset$
19: **for** $i \leftarrow 1$ to N **do**
20: R'.paths$[a_i] \leftarrow$ MLA$^*(a_i, R'.\text{sequencing}[a_i], R'.\text{constraints})$
21: R'.cost $\leftarrow$ flowtime$(R'.\text{paths})$
22: R'.collisions $\leftarrow$ findCollisions(R')
23: Insert R' to OPEN
24: $\langle a_i, a_j, u, t\rangle$ or $\langle a_i, a_j, v, u, t\rangle \leftarrow$ Earliest conflict in P
25: **for each** agent a_k **in** $\{a_i, a_j\}$ **do**
26: Q.root $\leftarrow$ *false* // Q is a child node of P
27: Q.sequencing $\leftarrow$ P.sequencing
28: Q.constraints $\leftarrow$ P.constraints $\cup \{\langle a_k, u, t\rangle / \langle a_k, v, u, t\rangle\}$
29: Q.paths$[a_k] \leftarrow$ MLA$^*(a_k, Q.\text{sequencing}[a_k], Q.\text{constraints})$
30: **if** Q.paths$[a_k]$ **is None then**
31: **continue**
32: Q.cost $\leftarrow$ flowtime$(Q.\text{paths})$
33: Insert Q to OPEN
34: **Return** No Solution

B. Sequential Next Best Task Sequencing (MILP Formulation)

Let $V' = \mathcal{S} \cup \mathcal{G}$ represent the union of initial and goal locations for all agents. Let x_{ijk} be the binary decision variable, which equals 1 if agent k traverses from location i to j, and 0 otherwise. Since the task sequencing phase ignores potential conflicts, c_{ij} in Cost matrix C represents the travel cost between locations i and j, which is the shortest distance obtained from A* planner. We define an auxiliary variable y_{ik} to handle sub-tour elimination constraints.

The MILP formulation is as follows:

$$\text{minimize} \sum_{k\in\mathcal{A}} \sum_{i\in V'} \sum_{j\in V'} c_{ij} x_{ijk} \tag{1}$$

Subject to

$$\sum_{i\in V'} \sum_{k\in\mathcal{A}} x_{ijk} = 1 \quad \forall j \in \mathcal{G} \tag{2}$$

$$\sum_{j\in V'} \sum_{k\in\mathcal{A}} x_{ijk} = 1 \quad \forall i \in \mathcal{S} \tag{3}$$

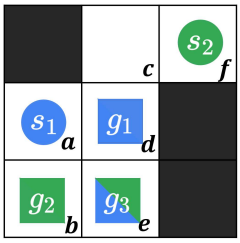
$$\sum_{i\in V'} x_{ijk} - \sum_{i\in V'} x_{jik} = 0 \quad \forall k \in \mathcal{A}, j \in \mathcal{G} \tag{4}$$

$$y_{ik} - y_{jk} + \tilde{M} x_{ijk} \leqslant \tilde{M} - 1 \quad \forall (i,j) \in E, k \in \mathcal{A} \tag{5}$$

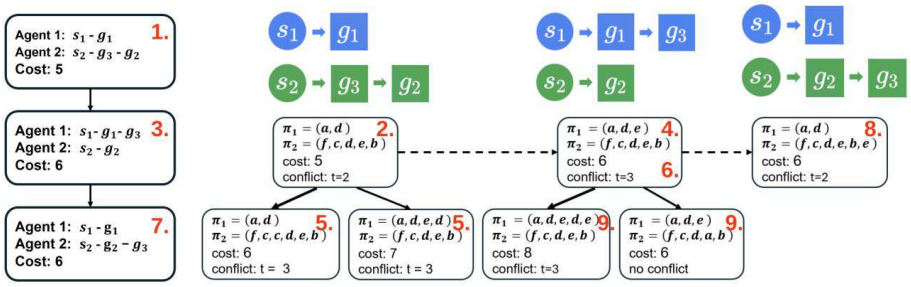
$$x_{ijk} \in \{0,1\} \quad \forall i \in V', j \in V', k \in \mathcal{A} \tag{6}$$

$$x_{ijk} \leq \delta_{kj} \quad \forall i \in V', j \in G, k \in \mathcal{A} \tag{7}$$

Constraints (2) and (3) ensure that each task location is visited exactly once by an agent, and each agent starts from



(a) Start and task locations on a 3x3 grid map with agent-task compatibility constraints.



(b) Best-first search on a conflict forest with the next-best task sequencing, where red numbers indicate the steps.

Fig. 2: An example of conducting the best-first search on the conflict forest to find the optimal solution in CBS-TS.

its designated initial location. Constraints (4) and (5) are the flow conservation and sub-tour elimination constraints, respectively, while the constraint (7) ensures that agent i can only visit goal j if they are compatible.

In the default MILP formulation, agents would return to their start locations. To prevent this, we modify the formulation by treating start locations as dummy goals with zero travel costs. This ensures that optimal task sequencing is not influenced by return trips. During low-level pathfinding, the final segment of the route is discarded to enforce the non-return condition.

To obtain the next best task sequencing for the construction of the conflict forest, we impose additional constraints on the MILP to exclude previously found solutions. Two approaches are proposed. The first approach is a ***direct constraint*** (8), which exludes all the previous solutions simultaneously. Since task sequencing requires exactly one visit to all the goals and dummy goals (starts), the sum of corresponding decision variables x_{ijk} equal to the total number of start locations and goal locations. The direct constraint ensures that the new solution differs from all previous solutions in at least one element—either by reassigning a task to a different agent or altering the sequencing of assigned tasks:

$$\sum_{k\in\mathcal{A}}\sum_{(i,j)\in P_k} x_{ijk} \le |\mathcal{A}| + |\mathcal{G}| - 1, \quad \forall \mathcal{P} \in \mathcal{P}_{\text{set}} \tag{8}$$

where $\mathcal{P}_{\text{set}}$ is the set of previously found sequencing. And P_k denote the set of visiting order (i,j) of agent k in a given sequencing $\mathcal{P}$,

The second approach is ***iterative constraints*** that excludes the previously found solutions one at a time. In each iteration, a single solution $\mathcal{P} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_N\}$, where $\mathcal{R}_i$ is the visiting sequence for agent i, is excluded. We define M as a non-empty subset of the solution space $\mathcal{Q}$. The iterative constraints are constructed as follows:

$$\begin{aligned} M_1 &= \{(\overline{\mathcal{R}_1})\} \\ M_2 &= \{\mathcal{R}_1; (\overline{\mathcal{R}_2})\} \\ &\cdots \\ M_n &= \{\mathcal{R}_1, \cdots, \mathcal{R}_{n-1}; (\overline{\mathcal{R}_n})\} \quad \text{for } n = 1, \cdots, N. \end{aligned} \tag{9}$$

where $\overline{\mathcal{R}_n}$ and $\mathcal{R}_n$ represent enforcing and prohibiting the task sequence of agent n in solution $\mathcal{P}$. These subsets M_n are mutually disjoint, and their union covers the entire solution space excluding $\mathcal{P}$ [32]. By solving the MILP within these subsets, we ensure a full exploration of the solution space while excluding $\mathcal{P}$. The mathematical formulations for banning and enforcing a specific route are as follows:

1) **Ban a specific route** $\overline{R_k} = \{v_{i_1}, v_{i_2}, \ldots, v_{i_n}\}$**:** Limits the agent to traverse at most $n-2$ edges in this sequence.

$$\sum_{j=1}^{n-1} x_{i_j i_{j+1} k} \le n - 2 \tag{10}$$

2) **Enforce a specific route** $R_k = \{v_{i_1}, v_{i_2}, \ldots, v_{i_n}\}$**:** Ensure that agent k follows the exact sequence without deviation.

$$x_{i_j i_{j+1} k} = 1 \quad \forall j \in \{1, 2, \ldots, n-1\} \tag{11}$$

The MILP can be solved with any off-the-shelf solver; in our experiments, we use Gurobi [33]. In Algorithm 1, MILP$(N, M, C, \Delta, \mathcal{P}_{\text{set}})$ represents the Gurobi MILP solver, which takes the number of agents N, the number of tasks M, the compatibility matrix Δ, the cost matrix C and the previously excluded sequencings $\mathcal{P}_{\text{set}}$ as input.

C. Low level of CBS-TS: Multi-label A*

The Multi-Label A* (MLA*) [31] algorithm extends the classical A^* search method to optimally handle scenarios where an agent must visit a sequence of predefined task locations. Each state is represented as a tuple containing the agent's current location, time, and task index. The task index, which tracks the current task, only updates when the agent reaches a task location, advancing to the next task in the sequence. This process continues until the agent reaches the final task.

MLA* uses an advanced heuristic that estimates the cost to reach the next task location, along with the cumulative cost of completing all remaining tasks. This heuristic, combined with the index-based state representation, ensures that MLA* finds not only a feasible path but also the optimal one, minimizing total travel time or cost.

TABLE I: Performance of four algorithms (CBSS, CBS-TS$_\mathrm{i}$, CBS-TS$_\mathrm{d}$, ECBS-TS) on 8×8 maps with agent-task constraints

N	M	CBSS				CBS-TS$_i$				CBS-TS$_d$				ECBS-TS			
		SR(%)	**Cost**	$\boldsymbol{t}\,(s)$	**Op**(s)	**SR**(%)	**Cost**	$\boldsymbol{t}\,(s)$	**Op**(s)	**SR**(%)	**Cost**	$\boldsymbol{t}\,(s)$	**Op**(s)	**SR**(%)	**Cost**	$\boldsymbol{t}\,(s)$	**Op**$(10^{-5}s)$
5	10	100	27.58	2.24	2.16	93.44	25.93	1.16	0.95	90.16	25.93	0.70	0.60	93.44	26.46	0.39	2.19
	20	96.61	40.62	13.99	13.81	91.53	39.53	4.02	3.29	91.53	39.53	6.16	5.47	94.92	40.51	0.87	2.19
	30	51.92	49.67	8.56	8.46	68.63	49.67	9.67	7.00	68.63	49.67	9.65	7.17	90.20	50.04	3.58	2.55
10	10	98.31	20.69	11.27	11.13	91.53	19.98	2.16	2.00	89.83	19.98	2.63	2.49	100	20.63	1.09	2.84
	20	28.00	30.14	8.39	8.31	72.00	30.14	10.14	9.56	72.00	30.14	9.93	9.36	94.00	30.64	2.78	2.91
	30	25.00	42.92	19.88	19.73	55.77	42.92	21.39	19.53	53.85	42.92	13.54	11.91	92.31	43.08	4.02	3.22
20	10	33.93	14.05	6.20	6.07	62.50	14.05	6.52	6.25	62.50	14.05	2.50	2.27	92.86	14.37	4.38	4.70
	20	24.07	24.66	24.59	24.38	35.19	24.66	9.13	8.35	37.04	24.66	3.80	3.05	77.78	24.92	7.41	4.80
	30	5.88	34.33	48.30	47.99	27.45	34.33	18.95	16.94	31.37	34.33	8.52	6.72	84.31	36.33	9.08	5.17

D. Example

In Figure 2, we provide a step-by-step explanation of how CBS-TS operates using the same example presented in Figure 1, with the red numbers indicating the sequence of steps. The algorithm first computes the best task sequencing (i.e. minimum-cost sequence) and corresponding paths while initially ignoring conflicts, forming the root node, which is added to the OPEN list (Steps 1-2). The node with the lowest cost is always selected for expansion. If the selected node is a root node, a new root node is first created by computing the next-best task sequencing before resolving conflicts (Steps 3-4). Then, the first conflict in the selected node is identified and resolved (Step 5). The process repeats: the lowest-cost node is selected (Step 6), and if it is another root node, a new root is generated before conflict resolution (Steps 7-8). Finally, conflict-free paths are obtained by resolving conflicts in the selected node (Step 9).

E. Properties of CBS-TS

Theorem 1 - CBS-TS is complete.

Proof. The completeness of CBS-TS relies on the following key points. First, CBS [14] is known to be complete for MAPF. MLA* [31] does not lose any feasible paths when navigating through a set of ordered goals. Thus, for a given task sequence, CBS-TS is guaranteed to find a conflict-free solution if one exists. Secondly, each time a root node is expanded, the next-best sequencing is computed until all possible sequencing have been explored. Since CBS-TS exhaustively explores both task sequencing (via MILP) and path planning (via CBS), it is guaranteed to find a solution if one exists.

Theorem 2 - CBS-TS guarantees an optimal solution that minimizes the flowtime, provided such a solution exists.

Proof. The optimality of CBS-TS stems from its systematic exploration of task sequences and conflict resolution. Each search tree in CBS-TS corresponds to a fixed task sequencing. The cost of the root node of a tree serves as a lower bound for that tree because the MLA* computes paths without any constraints, and any conflict resolution by imposing constraints can only maintain or increase the cost.

If the root node has conflict, CBS-TS does not immediately branch into child nodes to resolve them. Instead, it explores the next-best task sequencing, which becomes the root node of a new tree, thereby constructing a search forest. CBS-TS performs a best-first search over this search forest, the new sequencing will only be explored if it has a lower cost than current minimum cost. This guarantees that the cost of the found optimal solution is minimal among all explored task sequences and is no greater than the lower bounds of all unexplored root nodes, which represent the unexplored task sequences.

IV. Experiments

The proposed algorithms were implemented in C++ and tested on a 1.9GHz AMD Ryzen 7 Pro 5850U laptop with 16GB RAM, running Ubuntu 20.04 LTS.

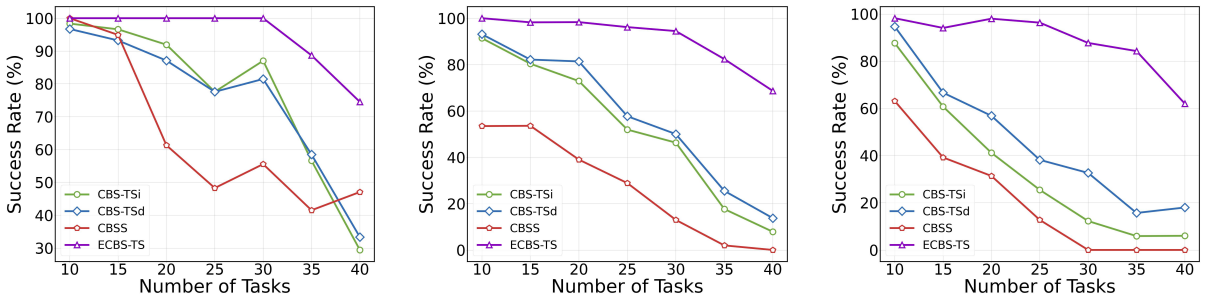
A. Simulation

We implement two variants of CBS-TS: CBS-TS$_\mathrm{d}$, which uses ***direct constraint***, and CBS-TS$_\mathrm{i}$, which uses ***iterative constraints***. Additionally, we introduce ECBS-TS, a bounded-suboptimal variant of CBS-TS$_\mathrm{d}$. ECBS-TS applies the same principles to CBS-TS$_\mathrm{d}$ as ECBS [20] does to CBS, incorporating focal search at both the high and low levels. Specifically, ECBS-TS introduces a suboptimality factor ω to balance solution quality and computation time. By maintaining a FOCAL list, which contains nodes whose f-values are within ω times the minimum f-value in the OPEN list, ECBS-TS enables faster identification of suboptimal solutions. The solution is guaranteed to be at most ω times worse than the optimal solution. In our implementation, we set $\omega = 1.2$.

We compare these methods against CBSS [27], which addresses a similar problem but requires a subset of tasks—equal to the number of agents—to serve as destinations where agents must terminate. To adapt CBSS for TSPF problem and ensure a fair comparison with CBS-TS, we modify CBSS as follows: (1) We use the same task set as CBS-TS, ensuring that all tasks are visited; (2) We introduce dummy destinations and set the cost from any location to these destinations to 0, effectively removing termination constraints. This allows agents to move freely without being assigned fixed destinations, making CBSS equivalent to CBS-TS in this setting. It should be noted that while CBSS claims optimality, its public implementation uses a heuristic TSP solver that does not guarantee an optimal solution[1].

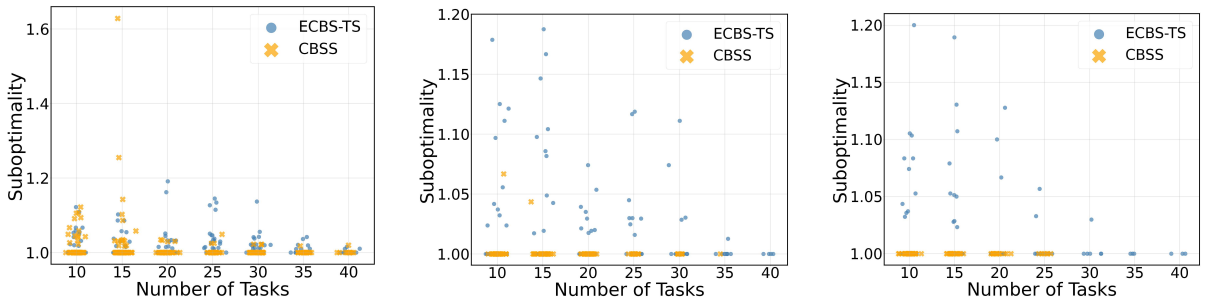
We conduct experiments on randomly generated 8×8 four-connected grids with 20% obstacles. The scenarios are

[1]https://github.com/rap-lab-org/public_pymcpf



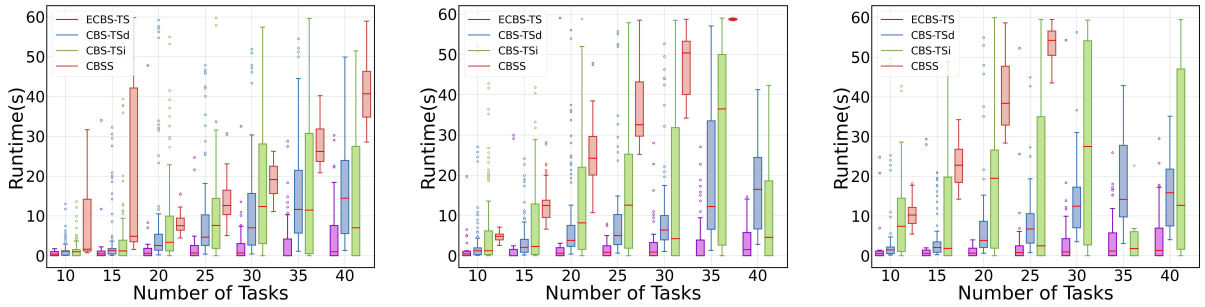
(a) 10 agents (b) 20 agents (c) 30 agents

Fig. 3: Success rate with $60s$ time limit.



(a) 10 agents (b) 20 agents (c) 30 agents

Fig. 4: Suboptimality ratio for CBSS and ECBS-TS. Only successful cases are included.



(a) 10 agents (b) 20 agents (c) 30 agents

Fig. 5: Boxplot of runtime. Only successful cases are included.

defined by varying numbers of agents $N \in \{5, 10, 20\}$ and tasks $M \in \{10, 20, 30\}$. The start locations of agents and task locations are unique and obstacle-free. Agents and tasks are assigned types to incorporate agent-task compatibility constraints: two types of agents (Agent α and Agent β) and three types of tasks (Task A, Task B, and Task C). Agent α can handle Task A and Task C; while agent β could execute Task B and Task C. For each setting, we run 60 instances with a time limit of 60 seconds for all algorithms.

Table I presents the performance of CBSS, CBS-TS$_\text{i}$, CBS-TS$_\text{d}$, and ECBS-TS on 8x8 maps. The success rate (**SR**), represents the percentage of instances, out of a total of 60, where a solution is found within the time limit. $\boldsymbol{t}$ denotes the total computational time, and **Op** refers to the optimization time for task sequencing, with MILP for CBS-TS$_\text{i}$, CBS-TS$_\text{d}$ and ECBS-TS, while mTSP for CBSS. **Cost** represents the average flowtime. For fairness, Cost, Op, and t are compared only for instances where all algorithms succeed.

As N and M increase, the success rates of all algorithms decline. ECBS-TS, a bounded-suboptimal method, consistently achieves the highest success rates, remaining above 80% even for 20 agents and 30 tasks. In contrast, CBSS experiences the steepest decline, dropping from 100% in small instances to 5.88% in the most complex case, highlighting its struggle to scale. CBS-TS$_\text{i}$ and CBS-TS$_\text{d}$ outperform CBSS, particularly in large instances, where they maintain significantly higher success rates. Runtime t increases with N and M for all algorithms. For CBS-TS$_\text{i}$, CBS-TS$_\text{d}$ and CBSS, the majority of the runtime is consumed by optimization. In contrast, ECBS-TS spends minimal time on optimization, with most of its runtime dedicated to conflict resolution. In terms of solution quality, CBS-TS$_\text{i}$ and CBS-TS$_\text{d}$ yield the lowest, as they are optimal methods. CBSS demonstrates near-optimal behavior, with

costs slightly higher than optimal in smaller instances but converging to optimality as the problem scale increases. ECBS-TS presents the highest cost, resulting from the trade-off between optimality and efficiency.

To assess scalability, we also analyze the algorithms on a larger 16×16 map with $N \in \{10, 20, 30\}$ and $M \in \{10, 15, 20, 25, 30, 35, 40\}$. The performance trends on 16×16 maps, illustrated in Figures 3, 4, and 5, align closely with those on 8×8 maps.

Figure 3 illustrates the success rates of all the methods. ECBS-TS consistently achieves the highest success rate. CBS-TS$_\text{i}$ and CBS-TS$_\text{d}$, consistently outperform CBSS, except in some 10 agents cases. Between the two, CBS-TS$_\text{i}$ achieves a slightly higher success rate than CBS-TS$_\text{d}$ for 10 agents. However, as the agent count grows, CBS-TS$_d$ scales better, indicating that solving a more constrained MILP is more efficient than solving a less constrained MILP N times when N is large.

Figure 4 presents the suboptimality ratio, defined as the flowtime of a suboptimal method divided by the flowtime of the optimal methods (CBS-TS$_\text{i}$/CBS-TS$_\text{d}$). ECBS-TS adheres to its 1.2 suboptimality bound, while CBSS demonstrates near-optimal behavior without a formal bound. Although most CBSS instances achieve suboptimality ratios below 1.2, there is one outlier exceeding 1.6 for 10 agents and 15 tasks. Notably, the scatter plot reveals a sparser distribution of points for CBSS suboptimal ratio in larger problem instances, with some cases disappearing entirely, reflecting the declining success rates of CBSS as the problem scale grows.

Figure 5 shows the runtime distributions across all algorithms for varying numbers of agents and tasks. ECBS-TS consistently achieves the lowest median runtime, followed by CBS-TS methods, while CBSS exhibits the highest median runtime. CBS-TS$_\text{i}$ exhibits higher variability than CBS-TS$_\text{d}$, occasionally achieving lower quartiles. This variability stems from CBS-TS$_\text{i}$ partitions the solution space into N subsets when searching for the next-best sequencing. Since the solution may reside in any of these subsets, instances where the solution falls into an earlier subset are solved more quickly, while others may take longer. However, in general, CBS-TS$_\text{i}$ requires more computation time than CBS-TS$_\text{d}$, especially as the number of agents and tasks increases. In some challenging cases (e.g., 20 agents with 40 tasks or 30 agents with 35 tasks), CBS-TS$_\text{i}$ appears to have a lower runtime than CBS-TS$_\text{d}$. However, this is due to its lower success rate—fewer successful instances mean only computationally easier cases are recorded, skewing the results.

B. Physical Experiment

We conducted real-world experiments with Turtlebot3 Burger robots in an indoor testing area of approximately 4.5m × 4.5m. A grid map with 0.25m resolution of the environment is built with *octomap_server* [34]. Figure 6a illustrates the real-world setup, where four Turtlebot3 Burger are placed at 4 start locations. Two robots at $[0.56, 0.66]$ and $[0.84, -1.24]$ are classified as type α, while the other two at $[1.98, 1.30]$ and $[-0.68, 2.02]$ are classified as type β. Figure 6b presents the start locations of the robots in Rviz.

Figure 6c and 6d compare the sequencing and trajectories generated for the agents under different task-type distributions, using the same 12 task locations but varying their task types. This highlights the impact of agent-task compatibility constraints on TSPF problems.

We used CBS-TS$_\text{d}$ to determine the optimal paths for each robot to visit its assigned tasks. Based on these paths, a Model Predictive Controller (MPC) in [35] was employed to time-parameterized trajectories. The MPC assigns a time interval of 1s for each waypoint in the CBS-TS$_\text{d}$ paths. Given the grid resolution of 0.25m, this results in an expected speed of 0.25m/s. The MPC incorporates a robot collision model with a radius of $R = 0.15$m, a maximum velocity of $v^{max} = 1$m/s, and a maximum angular velocity of $\omega^{max} = 1$rad/s.

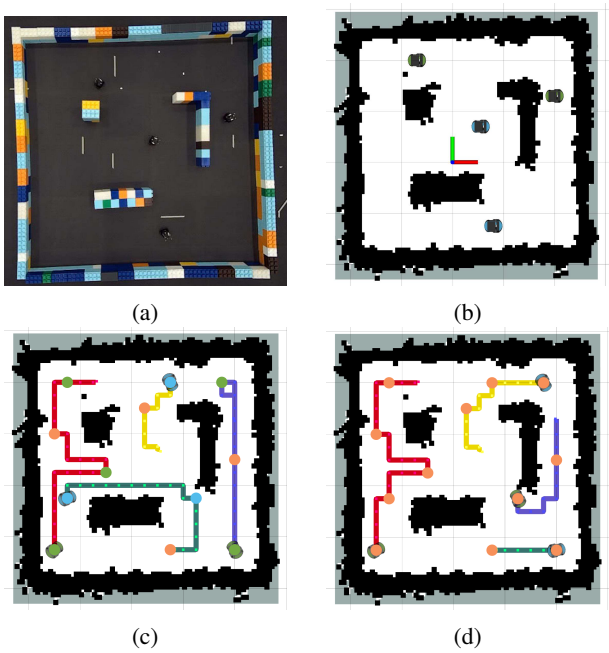


Fig. 6: Experimental setup and path visualization in Rviz. (a) Top view of the real-world environment. (b) Initial robot positions in Rviz: type α agents (blue) and type β agents (green). (c) and (d) Planned paths and generated trajectories under different task-type distributions. Paths are represented by colored cubic markers, while trajectories are shown as colored lines. Tasks are depicted as colored circles: Task A (blue) is exclusive to type α agents, Task B (green) is exclusive to type β agents, and Task C (orange) can be completed by either agent type.

V. Conclusion and Future Work

In this paper, we introduced CBS-TS, an optimal and complete algorithm for solving the collaborative multi-agent task assignment, sequencing and pathfinding (TSPF) problem, incorporating agent-task assignment constraints and ensuring full task coverage. Through extensive simulations, CBS-TS

not only guarantees optimality but also demonstrates better efficiency compared to the near-optimal implementation of the baseline method, CBSS. Real-world experiments validate the practical applicability of our framework, demonstrating its effectiveness in real-world scenarios.

For future work, we plan to incorporate nonlinear optimization techniques to smooth trajectories, ensuring dynamic feasibility for non-holonomic agents. Additionally, we aim to evaluate the robustness and scalability of CBS-TS in denser environments with more agents. This includes testing in constrained environments, such as narrow corridors where only one agent can pass at a time, as well as scenarios requiring agents to reposition after completing all tasks to allow others to proceed.

REFERENCES

[1] H. Ma, W. Hönig, T. S. Kumar, N. Ayanian, and S. Koenig, "Lifelong path planning with kinematic constraints for multi-agent pickup and delivery," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 7651–7658.

[2] D. S. Drew, "Multi-agent systems for search and rescue applications," *Current Robotics Reports*, vol. 2, pp. 189–200, 2021.

[3] H. Chakraa, F. Guérin, E. Leclercq, and D. Lefebvre, "Optimization techniques for multi-robot task allocation problems: Review on the state-of-the-art," *Robotics and Autonomous Systems*, p. 104492, 2023.

[4] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar, *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, no. 1, 2019, pp. 151–158.

[5] O. Cheikhrouhou and I. Khoufi, "A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy," *Computer Science Review*, vol. 40, p. 100369, 2021.

[6] G. D. Konstantakopoulos, S. P. Gayialis, and E. P. Kechagias, "Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification," *Operational research*, vol. 22, no. 3, pp. 2033–2062, 2022.

[7] K. Sundar and S. Rathinam, "An exact algorithm for a heterogeneous, multiple depot, multiple traveling salesman problem," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2015, pp. 366–371.

[8] E. Benavent and A. Martínez, "Multi-depot multiple tsp: a polyhedral study and computational results," *Annals of Operations Research*, vol. 207, pp. 7–25, 2013.

[9] K. Karabulut, H. Öztop, L. Kandiller, and M. F. Tasgetiren, "Modeling and optimization of multiple traveling salesmen problems: An evolution strategy approach," *Computers & Operations Research*, vol. 129, p. 105192, 2021.

[10] B. Soylu, "A general variable neighborhood search heuristic for multiple traveling salesmen problem," *Computers & Industrial Engineering*, vol. 90, pp. 390–401, 2015.

[11] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011, pp. 3260–3267.

[12] B. De Wilde, A. W. Ter Mors, and C. Witteveen, "Push and rotate: a complete multi-agent pathfinding algorithm," *Journal of Artificial Intelligence Research*, vol. 51, pp. 443–492, 2014.

[13] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[14] 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.

[15] J. Yu and S. M. LaValle, "Optimal multi-robot path planning on graphs: Structure and computational complexity," *arXiv preprint arXiv:1507.03289*, 2015.

[16] G. Gange, D. Harabor, and P. J. Stuckey, "Lazy cbs: Implicit conflict-based search using lazy clause generation," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, no. 1, pp. 155–162, May 2021.

[17] J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Artificial Intelligence*, vol. 301, p. 103574, 2021.

[18] E. Boyarski, A. Felner, R. Stern, G. Sharon, O. Betzalel, D. Tolpin, and E. Shimony, "Icbs: The improved conflict-based search algorithm for multi-agent pathfinding," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 6, no. 1, 2015, pp. 223–225.

[19] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 7643–7650.

[20] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Proceedings of the international symposium on combinatorial Search*, vol. 5, no. 1, 2014, pp. 19–27.

[21] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," 2016. [Online]. Available: https://arxiv.org/abs/1612.05693

[22] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2018.

[23] 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)*. IEEE, 2022, pp. 10 731–10 737.

[24] P. Surynek, "Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 14, 2021, pp. 12 409–12 417.

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

[26] 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 *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 441–447.

[27] Z. Ren, S. Rathinam, and H. Choset, "Cbss: A new approach for multiagent combinatorial path finding," *IEEE Transactions on Robotics*, vol. 39, no. 4, pp. 2669–2683, 2023.

[28] Y. Tang, Z. Ren, J. Li, and K. Sycara, "Solving multi-agent target assignment and path finding with a single constraint tree," in *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE, 2023, pp. 8–14.

[29] Y. Bai, C. Kanellakis, and G. Nikolakopoulos, "Sa-recbs: Multi-robot task assignment with integrated reactive path generation," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 7032–7037, 2023, 22nd IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896323009187

[30] Z. Ren, S. Rathinam, and H. Choset, "A bounded sub-optimal approach for multi-agent combinatorial path finding," *IEEE Transactions on Automation Science and Engineering*, 2024.

[31] F. Grenouilleau, W.-J. Van Hoeve, and J. N. Hooker, "A multi-label a* algorithm for multi-agent pathfinding," in *Proceedings of the international conference on automated planning and scheduling*, vol. 29, 2019, pp. 181–185.

[32] 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.

[33] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

[34] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, pp. 189–206, 2013.

[35] J. Li, M. Ran, and L. Xie, "Efficient trajectory planning for multiple non-holonomic mobile robots via prioritized trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 405–412, 2020.