Artificial Intelligence 347 (2025) 104388

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint

A scalable multi-robot goal assignment algorithm for minimizing mission time followed by total movement cost

Aakash [*], Indranil Saha

Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

A B S T R A C T

We study a variant of the multi-robot goal assignment problem where a unique goal for each robot needs to be assigned while minimizing the largest cost of movement among the robots, called makespan, and then minimizing the total movement cost of all the robots without exceeding the optimal makespan. A significant step in solving this problem is to find the cost associated with each robot-goal pair, which requires solving several complex path planning problems, thus, limiting the scalability. We present an algorithm that solves the multi-robot goal assignment problem by computing the paths for a significantly smaller number of robot-goal pairs compared to state-of-the-art algorithms, leading to a computationally superior mechanism to solve the problem. We perform theoretical analysis to establish the correctness and optimality of the proposed algorithm, as well as its worst-case polynomial time complexity. We extensively evaluate our algorithm for hundreds of robots on randomly generated and standard workspaces. Our experimental results demonstrate that the proposed algorithm achieves a noticeable speedup over two state-of-the-art baseline algorithms.

1. Introduction

Several multi-robot applications, such as warehouse management [1,2], disaster response [3], precision agriculture [4], mail and goods delivery [5], etc., require the robots to visit specific goal locations in the workspace to perform some designated tasks. These applications lead to the fundamental *goal assignment* problem for multi-robot systems: Given the initial locations of a set of robots and a set of goal locations, assign each robot to a unique goal such that some specific objective is satisfied with respect to the overall multi-robot system. The problem is a variant of the *Anonymous Multi-Agent Path Finding* (AMAPF) problem [6] in which we decide the one-to-one robot-goal assignment and find the corresponding collision-free paths satisfying some specific objective defined on the paths of the robots. Two objectives that are widely considered in the context of multi-robot systems are minimization of *makespan* which captures the overall mission time and minimization of *total movement cost*, henceforth referred as total cost, which captures the total energy consumption by the robots or the sum of individual robot travel times required for the completion of mission.

The AMAPF problem has been addressed in a number of recent works. We can organize the literature into two categories. The papers in the first category compute the goal assignment and the corresponding collision-free paths concurrently. Among these studies, those that focus on optimizing makespan primarily utilize the flow-based approaches [7,8] that suffer from huge size of the flow network, rendering them non-scalable, thus, ineffective for large problem instances. Conflict-based search with optimal task assignment (CBS-TA) [9] presents a task assignment approach by extending Conflict-based search (CBS) [10]. It optimizes the total cost of the multi-robot system while ensuring collision-free paths for the robots. However, it scales poorly as the number of robots grows in the environment.

The papers in the second category start with computing an initial goal assignment, and then find the collision-free paths for this assignment. Based on the chosen objective, the initial goal assignment can be determined either by solving the *linear bottleneck*

[*] Corresponding author.
E-mail addresses: aakashp@cse.iitk.ac.in (Aakash), isaha@cse.iitk.ac.in (I. Saha).

https://doi.org/10.1016/j.artint.2025.104388
Received 19 June 2024; Received in revised form 7 March 2025; Accepted 15 June 2025

Available online 20 June 2025

assignment (LBA) problem [11,12] (for optimizing makespan), or by applying the well-known Hungarian algorithm [13,14] (for optimizing total cost). Among the papers in the second category, SCRAM [15] employs a graph-theoretic algorithm that is built upon unrealistic assumptions that the environment is obstacle-free and that a robot travels from its initial location to its goal location along a straight-line path. The algorithm presented in [16] also assumes the absence of obstacles in the environment. To solve the goal assignment problem in realistic settings where the environment has obstacles, either the costs for all robot-goal pairs are obtained by computing the path for each pair [17–19], or they are derived in a lazy manner by considering the estimated robot-goal distances in ascending order and evaluating the corresponding actual distance until all the goals are assigned to the robots [20]. These decoupled approaches cannot ensure the optimality of the solution but have the potential to provide more scalability compared to the algorithms solving the goal assignment and the path planning problems concurrently. However, they still have computational limitations in solving the underlying goal assignment problem, which prevent them from scaling up to large workspaces and large number of robots.

We aim to solve the goal assignment problem for a large multi-robot system with hundreds of robots within a short duration. However, achieving this aim in a centralized manner poses a major challenge, as it requires knowing the cost of reaching each goal location for every robot in advance. This, in turn, requires solving several complex path planning problems, which limits the scalability. In our recent work, we have proposed *scalable centralized* algorithms for solving the multi-robot goal assignment problem for optimizing total cost [21] and makespan [22]. Our algorithm for optimizing total cost, called OTC, is based on the bipartite graph-based implementation of the primal-dual Hungarian algorithm [13,14] that is widely used to solve the assignment problem. On the other hand, the algorithm optimizing makespan, called OM, is based on the bipartite graph-based implementation of the dual method to solve the LBA problem [23]. However, unlike those standard algorithms, OTC and OM do not assume that the costs for all the robot-goal pairs are available a priori. Rather, we initialize those costs with an admissible heuristic cost and compute the actual cost by finding the optimal obstacle-free path for any robot-goal pair on demand, i.e., if it is indeed essential for the optimality of the goal assignment. We do not attempt to solve the AMAPF problem *optimally* as eliminating collisions statically without compromising optimality for such large multi-robot systems is computationally infeasible. We instead aim to solve the multi-robot goal assignment problem efficiently to get an optimal assignment where each robot is paired with a goal based on its independent optimal path. An efficient goal assignment algorithm enables the robots to embark on their individual optimal paths towards their respective goals quickly and deal with collisions with other robots and any dynamic obstacles using their local collision avoidance mechanism [24–28]. Also, an efficient goal assignment algorithm, when plugged into the algorithms presented in [17–20], provides a more efficient solution to the AMAPF problem.

While solving the goal assignment problem for a large multi-robot system optimizing makespan or total cost independently are significant contributions, many practical applications of multi-robot systems entail optimizing both makespan and total cost simultaneously. This indicates that we must aim for designing an algorithm that solves the goal assignment problem co-optimizing both the objectives. However, due to the existence of *Pareto-optimal structure*, both the objectives cannot be optimized simultaneously [7]. For several applications, such as disaster response, warehouse management, precision agriculture and so on, we want to complete the mission as soon as possible. As a secondary objective, we also aim for minimizing the total energy consumption due to the movement of the robots. Keeping the above requirements in mind, in this work, we propose an algorithm where we treat optimizing makespan to be the primary objective and optimizing total cost to be the secondary objective. Our algorithm first employs the method presented in [22] to identify all the assignments having the optimal makespan. Subsequently, we adapt the solution from [21] to obtain an assignment with the optimal total cost among the assignments having the optimal makespan.

We implement our algorithm in Python and evaluate it through thorough experimentation on randomly generated 2D workspaces and standard 2D and 3D workspaces. We compare the total cost of assignment solutions obtained from OM [22], our proposed algorithm, and OTC for several standard workspaces available in the literature [29,6]. The comparison results demonstrate that our approach to solve the aforesaid multi-objective goal assignment problem provides solutions having optimal mission time without compromising on the optimality of the total cost of the trajectories of the robots significantly. For the comparison of computation time, we consider two algorithms as the baseline. The first is the goal assignment algorithm where the costs for all robot-goal pairs are computed, as is done in [17–19]. The second is the bottleneck assignment algorithm in TSWAP (Algorithm 3 in [20]) that uses lazy evaluation of actual costs. Our experimental results demonstrate that the proposed algorithm achieves a noticeable speedup over the two state-of-the-art baseline algorithms. We also go a step ahead and replace the bottleneck assignment algorithm in TSWAP with our algorithm for the initial goal assignment, which makes their AMAPF solution significantly faster.

Contributions. Our contributions in this paper can be summarized as follows:

- We present a highly scalable algorithm for the goal assignment problem for multi-robot systems. Without considering collision avoidance among the robots, the solution provided by our algorithm is optimal with respect to makespan and optimal with respect to total cost among all the solutions having optimal makespan.
- We provide a detailed theoretical analysis of the proposed algorithm. Specifically, we prove that the proposed algorithm is guaranteed to generate optimal solution with respect to makespan followed by total cost. Furthermore, we provide a detailed analysis of the worst-case time complexity of the algorithm.
- We implement our algorithm in Python and carry out exhaustive experimentation. Our empirical findings confirm that our algorithm generates assignment solutions with near-optimal total cost values, which is significantly less than the total cost of the optimal-makespan assignment. This underscores the importance of addressing the aforesaid multi-objective goal assignment problem rather than focusing solely on optimizing makespan. Our results also show that our algorithm consistently outperforms

two state-of-the-art methods in terms of computation time by a significant margin for both randomly generated workspaces and benchmark workspaces.

- We plug our algorithm into the state-of-the-art decoupled algorithm TSWAP for the AMAPF problem, leading to an order-of-magnitude improvement in its computation time for a large number of robots and thus providing a highly scalable solution for multi-robot goal assignment with collision-free paths.

Connection with the Conference Papers. The algorithmic solution presented in this paper employs techniques from two of our previously published conference papers [21] and [22]. We use the optimal-makespan goal assignment algorithm presented in [22] directly, whereas the optimal total cost goal assignment algorithm presented in [21] requires significant adaptation. In both cases, we provide a more elaborate description of the algorithms. We provide a detailed theoretical analysis of our algorithm, which was missing in both the conference papers. Furthermore, all experiments evaluating the efficacy of the algorithm proposed in this paper have been conducted independently.

Paper Organization. The remainder of the paper is structured as follows: Section 2 delves into a comprehensive review of the current state-of-the-art in goal assignment and path finding for multi-robot systems. Next, in Section 3, we define the problem formally and provide an illustrative example. Section 4 presents our algorithm, whereas its theoretical analysis for correctness and time complexity is presented in Section 5. We present a thorough evaluation of our algorithm in Section 6. Finally, we conclude the paper in Section 7, with an outline of possible future research directions.

2. Related work

The fundamental goal assignment problem for multi-robot systems has gained considerable attention from the research community in the past decade, highlighting its inherent importance. Table 1 provides a concise summary of the related literature. Throughout the table, ‘NA’ denotes ‘Not applicable’. In general, the assignment problem optimizing the total cost (TC) is known as *linear sum assignment* problem [30]. It can be solved in polynomial time by the well-known Hungarian algorithm [13,14]. On the other hand, an assignment solution optimizing makespan (MS) can be obtained by solving the popular *linear bottleneck assignment* problem [11,12]. The second column of Table 1 distinguishes between centralized (Cen.) and decentralized (Decen.) approaches. A major challenge in solving the goal assignment problem in a centralized manner is that the cost of reaching each goal location by each robot needs to be known a priori. This requires solving several complex path planning problems, which limits the scalability.

To circumvent the computational challenge of the centralized approach, several researchers have developed decentralized solutions to the multi-robot goal assignment problem [31–35]. In the decentralized setting, each robot is aware of the cost of reaching a subset of goals. With their available information, they participate in a consensus algorithm that involves communication with other robots. These algorithms are computationally efficient but may involve a large volume of message passing leading to significant communication overhead, which may render the practical implementation of such algorithms infeasible. Thus, a major outstanding question is whether it is possible to design a centralized solution to the multi-robot goal assignment problem that is computationally efficient and thus can be deployed successfully for hundreds of robots and goals.

The multi-robot goal assignment problem studied in this paper is a restricted version of the *Anonymous Multi-Agent Path Finding* (AMAPF) problem [6] in which collision-free paths for the robots are also computed together with the goal assignment. The fifth column of Table 1 indicates the two approaches that have been devised in the literature to solve the AMAPF problem. The first is the decoupled or sequential approach (Seq.) in which an initial goal assignment is computed, and then collision-free paths are found for this assignment [16–20]. These decoupled approaches cannot ensure the optimality of the solution but have the potential to be scalable. The second is the integrated approach (Int.) in which the goal assignment and the corresponding collision-free paths are computed concurrently in an iterative manner [15,7–9]. Although these integrated approaches ensure optimal solutions, they often compromise on scalability.

The research works presented in [15] and [16] make the assumption of an obstacle-free operational environment, which is not reflective of real-world conditions. TSWAP [20] operates by computing an initial goal assignment and then performing suboptimal collision-free path planning by applying the idea of stop-and-go and target swapping to resolve conflicts between the paths of two robots. For the computation of goal assignment, it evaluates the shortest obstacle-free distance between robot-goal pairs by considering the estimated distances in ascending order, and checks for a feasible assignment after the distance evaluation for each pair. It repeats this process until all the goals are assigned to the robots. In TSWAP, the authors present two distinct optimal goal assignment algorithms (without considering robot-robot collisions): one that optimizes makespan alone, and another that optimizes makespan followed by total cost within the set of assignments having the optimal makespan. The need to compute a feasible assignment after each distance evaluation is a major bottleneck, which our approach effectively overcomes.

It is worth noting that all the studies listed in Table 1 solve the assignment problem. Among these, the approaches specifically targeting the AMAPF problem (super rows 3 and 4) tackle the assignment computation in a naive way and rather, focus on the collision avoidance aspect. However, the assignment computation in itself is significantly computationally-intensive which impedes the scalability of the overall approach to tackle the AMAPF problem. Our paper aims to enhance the scalability of assignment computation within this context.

Table 1
Summary of Related Work.

Reference Coordinates	Cen. / Decen.	Obstacles	Objective	Seq. / Int.	Ensure Col. Avoid.	Optimal Paths
(Fulkerson et. al., The Rand Corporation, 1953, [11])	Cen.	NA	MS	NA	No	NA
(Kuhn, Naval Research Logistics, 1955, [13])	Cen.	NA	TC	NA	No	NA
(Munkres, SIAM Journal, 1957, [14])	Cen.	NA	TC	NA	No	NA
(Gross, The Rand Corporation, 1959, [12])	Cen.	NA	MS	NA	No	NA
(Choi et. al., IEEE Trans. Robotics, 2009, [31])	Decen.	NA	TC	NA	No	NA
(Giordani et. al., Trends in Applied Intelligent Systems, 2010, [32])	Decen.	NA	TC	NA	No	No
(Liu and Shell, RSS, 2012, [34])	Decen.	NA	TC	NA	No	No
(Giordani et. al., Computers and Industrial Engineering, 2013, [33])	Decen.	NA	TC	NA	No	No
(Chopra et. al., IEEE Trans. Robotics, 2017, [35])	Decen.	NA	TC	NA	No	No
(Turpin et. al., Algorithmic Foundations of Robotics X, 2013, [16])	Cen., Decen.	No	TC, Velocity^2	Seq.	Yes	Yes (Cen.), No (Decen.)
(Turpin et. al., RSS, 2013, [17])	Cen.	Yes	MS	Seq.	Yes	No
(Turpin et. al., ICRA, 2013, [18])	Cen.	Yes	MS	Seq.	Yes	No
(Turpin et. al., Auton. Robots, 2014, [19])	Cen.	Yes	MS	Seq.	Yes	No
(Okumura and Défago, Artificial Intelligence, 2023, [20])	Cen.	Yes	MS, MS $\rightarrow$ TC	Seq.	Yes	No
(Yu and LaValle, Algorithmic Foundations of Robotics X, 2013, [7])	Cen.	Yes	MS, TC	Int.	Yes	Yes
(MacAlpine et. al., AAAI, 2015, [15])	Cen.	No	MS	NA	Yes	Yes
(Ma and Koenig, AAMAS, 2016, [8])	Cen.	Yes	MS	Int.	Yes	Yes
(Hönig et. al., AAMAS, 2018, [9])	Cen.	Yes	TC	Int.	Yes	Yes

The columns represent the following information: **Reference Coordinates**: Provides author information, the venue of publication, and the year of publication for each research work; **Cen. / Decen.**: Indicates whether the approach is centralized (Cen.) or decentralized (Decen.); **Obstacles**: Points out whether obstacles are considered in the corresponding work; **Objective**: Highlights the objective function used for optimization in the study (MS (Makespan), TC (Total Cost), Velocity^2, MS $\rightarrow$ TC (MS followed by TC)); **Seq. / Int.**: Differentiates between sequential (Seq.) and integrated (Int.) approaches; **Ensure Col. Avoid.**: States whether the work ensures avoidance of robot-robot collisions; **Optimal Paths**: Indicates if the overall paths are optimal for the multi-robot system. *Note*: Throughout the table, 'NA' means 'Not Applicable'.

3. Problem

3.1. Preliminaries

Notations. Let $\mathbb{N}$ represent the set of natural numbers and $\mathbb{R}$ represent the set of real numbers. For a natural number $X \in \mathbb{N}$, let $[X]$ denote the set $\{1, 2, 3, \ldots, X\}$.

Workspace. A workspace WS is a rectangular (2D) or cuboidal (3D) space which is divided by grid lines into square-shaped or cube-shaped cells, respectively. Each cell can be addressed using its coordinates. In general, a workspace consists of a set O of cells that are occupied by obstacles. Mathematically, $WS = \langle dimension, O \rangle$, where $dimension$ is a tuple of the number of cells along the coordinate axes.

Motion Primitives. In a 2D workspace, we assume that a robot can move in 8 directions (North, South, East, West, North-East, North-West, South-East, and South-West) from its current location while respecting the workspace boundaries. It can move diagonally only if the cells on the sideways are obstacle-free. The cost of a diagonal movement is 1.5 units, while the same for a non-diagonal movement is 1 unit. Similarly, we consider that a robot can move in 26 directions in a 3D workspace. We take the cost of a diagonal movement which causes displacement in all the three axes as 2 units, the same of a diagonal movement on a plane parallel to coordinate axes as 1.5 units, while the same for a non-diagonal movement as 1 unit. Here, the cost of each motion primitive represents both the delay and the energy consumed in executing it.

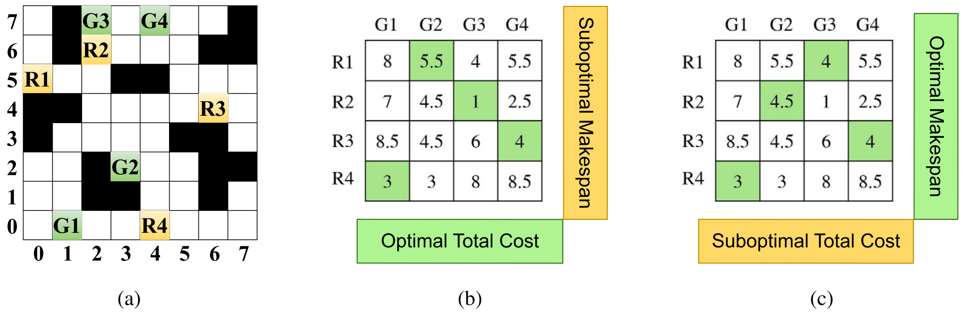


Fig. 1. Assignment with Pareto-optimality between makespan and total cost. (a) A problem instance comprising of a 8×8 grid-based workspace with 4 robots (R1, R2, R3, and R4) and 4 goals (G1, G2, G3, and G4). The black-colored cells denote obstacles whereas the white cells indicate free area through which the robots can navigate. (b) Corresponding cost matrix showing optimal-total cost but suboptimal-makespan assignment using the green cells. (c) Corresponding cost matrix showing optimal-makespan but suboptimal-total cost assignment using the green cells. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Graph Theory Concepts [36]. This paper employs the following concepts from graph theory:
A *bipartite graph* is a graph $\mathcal{G} = (V, E)$ whose vertex set V is partitioned into two disjoint sets V_1 and V_2 such that each edge in the edge set E has one of its endpoints in V_1 and the other in V_2.
A *matching* M in a graph $\mathcal{G} = (V, E)$ is a subset of the edge set E such that no two edges in M have a common vertex. A vertex is said to be *saturated* by M if it is an endpoint of an edge in M. A matching having the largest possible number of edges is called *Maximal Cardinality Matching*.
A *vertex cover* is a subset of vertices of a graph $\mathcal{G} = (V, E)$ such that it consists of at least one endpoint of each edge of $\mathcal{G}$. A vertex cover with the minimum number of vertices is called *Minimum Vertex Cover*.

Connection between Goal Assignment and Matching. We assume that any two robots cannot be assigned to the same goal, as well as any two goals cannot be delegated to a common robot. This assumption makes the computation of goal assignment analogous to finding a matching in a bipartite graph, whose bipartition corresponds to the robots and the goals, and whose edge set represents the paths between the corresponding robot-goal pairs.

3.2. Problem definition

In a typical multi-robot application, robots must complete a set of tasks within a designated workspace. These tasks are associated with specific locations within the workspace, referred to as *goal locations*. The robots must navigate to their respective goal locations to complete their assigned tasks. This brings up what is called the fundamental *goal assignment* problem for multi-robot systems. In this problem, the initial locations of a set of robots and a set of goal locations are given, and the aim is to assign each robot to a unique goal such that some specific objective is satisfied with respect to the overall multi-robot system.

The two prominent objectives considered while solving the multi-robot goal assignment problem are minimization of: (a) the largest movement cost among all the robots, also called *makespan* (relevant in time-critical missions), and (b) the total movement cost incurred due to all the robots (relevant in efficiency contexts). It has been shown in [7] that there exists a *Pareto-optimal structure* for both these objectives when considered in the goal assignment problem. It essentially means that there exist problem instances for which both the objectives cannot be optimized simultaneously (see Fig. 1).

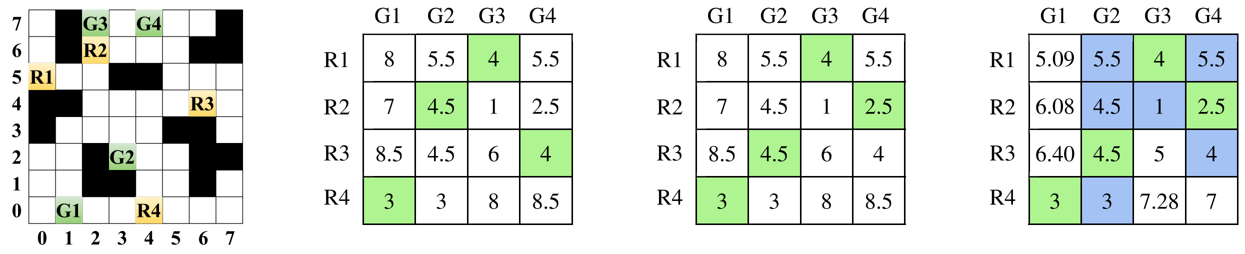
The set of all Pareto-optimal solutions is called *Pareto front*. Since generating the entire Pareto front is usually computationally hard, we consider a lexicographic version of this problem where the primary objective of a multi-robot system is to complete the mission as soon as possible (i.e., in minimal makespan). However, there could be multiple solutions with varying total costs, yet achieving the optimal makespan. This prompts consideration of a secondary objective: to obtain an optimal total cost solution from the set of solutions having the optimal makespan. We proceed to define the problems formally.

Problem 1 *(Goal Assignment with Optimal Makespan). Consider a multi-robot application in a grid-based workspace WS, where the set S of start locations of the robots and the set F of goal locations are given as inputs. Let $R = |S|$ and $G = |F|$ denote the number of robots and goals, respectively.*

Each robot can be assigned to at most one goal, and each goal can be served by at most one robot. Let $\mathtt{cost}(i, j)$ denote the cost of movement between $s_i \in S$ and $f_j \in F$, where $i \in [R]$ and $j \in [G]$. In the bipartite graph $\mathcal{G} = ([R], [G], E)$ with bipartition $([R], [G])$, edge set $E = \{(i, j) \mid i \in [R],\ j \in [G]\}$, and a cost function $\mathtt{cost} :\ E \rightarrow \mathbb{R}^+$, find a maximum cardinality matching M such that the maximum cost among the edge costs (makespan) in M is minimized (without considering the overhead for collision avoidance).
Mathematically,

$$\underset{M}{\mathtt{minimize}}\ \textit{makespan}(= \mathtt{maximum}_{(i,j)\in M}(\mathtt{cost}(i, j))).$$

Let us extend Problem 1 to define the following problem.



(b)

	G1	G2	G3	G4
R1	8	5.5	4	5.5
R2	7	4.5	1	2.5
R3	8.5	4.5	6	4
R4	3	3	8	8.5

(c)

	G1	G2	G3	G4
R1	8	5.5	4	5.5
R2	7	4.5	1	2.5
R3	8.5	4.5	6	4
R4	3	3	8	8.5

(d)

	G1	G2	G3	G4
R1	5.09	5.5	4	5.5
R2	6.08	4.5	1	2.5
R3	6.40	4.5	5	4
R4	3	3	7.28	7

Fig. 2. An example problem. (a) A problem instance comprising of a 8×8 grid-based workspace with 4 robots (R1, R2, R3, and R4) and 4 goals (G1, G2, G3, and G4). The black-colored cells denote obstacles whereas the white cells indicate free area through which the robots can navigate. (b) Cost matrix with actual movement cost for all robot-goal pairs. Here, the green cells depict an assignment that has optimal makespan but suboptimal total cost. (c) Cost matrix with actual movement cost for all robot-goal pairs. Here, the green cells depict an assignment that has an optimal total cost among the assignments having the optimal makespan. (d) Cost matrix in which only the colored cells have actual movement cost whereas the white cells have heuristic cost (Euclidean distance), while we have the same assignment as in (c).

Problem 2 *(Goal Assignment with Optimal Makespan followed by Optimal Total Cost). Given the optimal makespan $makespan_{opt}$ in a bipartite graph $\mathcal{G} = ([R], [G], E)$ with bipartition $([R], [G])$, edge set $E = \{(i, j) \mid i \in [R],\ j \in [G]\}$, and a cost function $\mathtt{cost} : E \to \mathbb{R}^+$, find a maximum cardinality matching M such that the cost of each edge in M does not exceed $makespan_{opt}$ and the total cost of edges in M is minimized (without considering the overhead for collision avoidance).*
Mathematically,

$$\underset{M}{\mathtt{minimize}} \sum_{(i,j)\in M,\ \mathtt{cost}(i,j)\le makespan_{opt}} \mathtt{cost}(i, j).$$

The cost information needs to be known to solve both the problems. In order to find the *actual cost* of movement between two cells in a workspace, the shortest obstacle-free path between them has to be computed. Finding such cost for all the robot-goal pairs requires solving several complex path planning problems, which limits the scalability. We aim to design an algorithm that computes these costs judiciously for only the necessary robot-goal pairs while finding optimal solutions to the goal assignment problems.

3.3. Example

Consider a multi-robot application in the 8×8 workspace shown in Fig. 2(a). It has four robots (R1, R2, R3, and R4) and four goals (G1, G2, G3, and G4). The black-colored cells denote the obstacles whereas the white cells indicate free area through which the robots can navigate. Figs. 2(b) and 2(c) show the matrices having an actual cost for each robot-goal pair. A feasible goal assignment is depicted by the green color in each of these matrices. The assignment in Fig. 2(b) has an optimal makespan but sub-optimal total cost. Fig. 2(c) shows an assignment that has an optimal total cost among the assignments having the optimal makespan. Now, can we have the same optimal goal assignment as in Fig. 2(c) without having to compute all the actual costs? Fig. 2(d) depicts the transformed matrix from Fig. 2(c), which is an outcome of our approach. Here, only the colored cells have the actual costs, and we obtain the same optimal assignment as in Fig. 2(c).

4. Algorithm

In this section, we present the details of our algorithmic solutions to solve the multi-robot goal assignment problems (Problems 1 and 2). Algorithm 1 offers the solution to Problem 1 by computing a goal assignment that minimizes makespan. Furthermore, Algorithm 2 outlines the solution to Problem 2 by computing a goal assignment that optimizes the total cost among the set of assignments having the optimal makespan. Both the algorithms use Euclidean distance between a robot's start location and a goal location as the *admissible* heuristic cost, or *H-cost* for short. Hereafter, we shall refer to the optimal actual cost (after taking the obstacles into account) between the same locations as *A-cost* for short. Note that between any two locations, an H-cost never exceeds its corresponding A-cost. For efficient computation of an optimal obstacle-free path and the corresponding A-cost, we adapt *Reverse Resumable* A* *(*RRA**)* [37] to implement *Forward Resumable* A* (FRA*). The RRA* algorithm is a variant of the well-known A* path planning algorithm [38], designed to significantly reduce computation time when computing paths from several robots' initial locations to a specific goal location. Like A*, it maintains two data structures: an open-list (OL), which stores elements corresponding to unexplored workspace cells prioritized by the sum of their current cost (from goal location to current location) and heuristic cost (from current location to robot's initial location), and a closed list (CL), which tracks elements corresponding to the already explored workspace cells. For computing the first path to a goal location, RRA* operates identically to A*, finding a path from a robot's initial location to that goal location in the reverse direction. However, for subsequent path computations to the same goal, for each such path, RRA* updates the priority of elements in OL by revising their heuristic cost based on the new robot's initial location. It then resumes the path search from the element in OL that has the highest priority, instead of starting the search from scratch. This reuse of the prior search effort (information stored in OL) significantly improves efficiency. The terminating condition for search is the same for both RRA* and A*, i.e., the search continues until a path to the goal location is found or OL becomes empty, whichever occurs first.

In our algorithm (Algorithm 1 and Algorithm 2), we need to compute the paths to multiple goal locations for a specific robot in order to obtain the least cost path for the robot. Thus, our adaptation, FRA*, reverses the direction of search: instead of computing paths from a goal location to multiple initial locations (as in RRA*), it computes paths from a robot's initial location to multiple goal locations, making it a forward search variant.

4.1. *Algorithm description*

Goal Assignment Optimizing Makespan (Algorithm 1). We present our goal assignment algorithm **OM** formally in Algorithm 1 that computes an **O**ptimal **M**akespan for multi-robot systems. The first procedure `get_optimal_makespan` captures the *main* module that takes the workspace WS, and the sets S and F as inputs. To begin with, it computes the H-cost for each robot-goal pair and stores them in the 2D matrix C (Algo 1: Line 5). We keep a record of the cost-type attribute (i.e., whether a cost is heuristic or actual) in the 2D matrix T, and it is initialized with 'h' (symbolizing 'heuristic') for all the robot-goal pairs. The paths are initialized in the 2D matrix P (Algo 1: Line 6).

Depending on the relationship between R and G (i.e., $R = G$, $R < G$, or $R > G$), OM determines an initial estimate of makespan ($makespan_{init}$) by using the minimum A-cost information of each robot ($acost_{Rmin}$) and/or each goal ($acost_{Gmin}$) (Algo 1: Lines 7-15):

$$\begin{aligned} R = G &: makespan_{init} \leftarrow \mathtt{max}(\max_{i\in[R]} acost_{Rmin}(i),\ \max_{j\in[G]} acost_{Gmin}(j)); \\ R < G &: makespan_{init} \leftarrow \max_{i\in[R]}(acost_{Rmin}(i)); \\ R > G &: makespan_{init} \leftarrow \max_{j\in[G]}(acost_{Gmin}(j)). \end{aligned} \tag{1}$$

The initial makespan serves as a lower bound of the optimal makespan. When R and G are equal, each robot and goal must get assigned, and therefore, the optimal makespan must be at least the maximum of the minimum A-costs of each robot and goal. So, for this case, the initial makespan is computed using the minimum A-cost of both the robots and the goals. However, if R and G are unequal, then it is computed using the minimum A-cost information of only the minority entity, because considering the majority entity may give a misleading value of the initial makespan (which may be even greater than the optimal value), as some of the elements in the majority entity would not get an assignment.

The procedure `explore_min_acost` discovers the minimum A-cost for each robot or each goal (referred to as *pivot_entity*) without computing all its A-costs *naively* (Algo 1: Lines 7-14). It uses a flag variable *flag* to indicate whether the minimum A-cost has to be computed for the robots (*flag* = 1) or the goals (after its invocation for the robots (*flag* = 2) or independently (*flag* = 3)). To compute the minimum A-cost for each robot, it proceeds in the following way: At the outset, a particular robot i has an H-cost for each of the goals. The procedure searches for its minimum H-cost $hcost_{min}$ and replaces it with the corresponding A-cost. This step repeats until robot i's current minimum H-cost exceeds its current minimum A-cost (Algo 1: Line 52). This ensures that further replacement of H-costs is unnecessary, as they already serve as under-approximations of A-costs, and any replacements would only yield equal or higher A-costs. Note that when *flag* is 2, the minimum A-cost for each goal is not initialized directly with infinity, but by using an auxiliary procedure `find_min_cost` (Algo 1: Line 50). It is because, for a particular goal, some H-costs may have got replaced by corresponding A-costs while finding the minimum A-cost for each robot. The procedure `find_min_cost` fetches the minimum cost of a particular cost-type (heuristic or actual). Whenever an H-cost is replaced by its corresponding A-cost, we also (i) save the path information (Algo 1: Lines 55 and 59) and (ii) update the cost-type in matrix T (Algo 1: Lines 56 and 60).

The next step (Algo 1: Line 16) is to formulate a threshold bipartite subgraph $\mathcal{G}_{th} = ([R], [G], E_{th})$ having bipartition ($[R]$, $[G]$) and whose edge set E_{th} is given by:

$$E_{th} = \{(i, j) \mid \mathtt{cost\text{-}type}(i, j) = \text{'}a\text{'} \text{ and } \mathtt{cost}(i, j) \leq makespan\}.$$

Symbolically, the robots and the goals form the bipartition of the vertex set of $\mathcal{G}_{th}$. A maximum cardinality matching M is obtained in $\mathcal{G}_{th}$ (Algo 1: Line 17). A vertex is said to be *saturated* by a matching if it is an endpoint of a matched edge. We define a matching as *total matching* if it completely saturates the set of robots $[R]$ (when $R \leq G$) or the set of goals $[G]$ (when $G \leq R$). Thus, if M is a total matching in $\mathcal{G}_{th}$, the procedure provides the following as output before terminating: (i) *makespan*: the optimal makespan, (ii) $acost_{min}$: minimum A-cost information for each robot and/or each goal. However, if the current M is not a total matching, additional steps are required to update M until it becomes one. The additional steps begin by finding the minimum vertex cover $\langle R_c, G_c \rangle$ (Algo 1: Line 19), which can be computed for a bipartite graph in polynomial time by utilizing M [39]. The sets R_c and G_c refer to the covered robots and goals, respectively. The procedure `collect_uncovered_costs` collects the cost of the edges between all the uncovered robot-goal pairs (Algo 1: Line 20). The collection data structure Δ stores tuples of the following information, which can be accessed using the dot operator: (i) robot ID $i \in [R]$, (ii) goal ID $j \in [G]$, (iii) cost $C(i)(j)$, and (iv) cost-type $T(i)(j)$.

The procedure `update_makespan` finds the minimum uncovered *A-cost* and checks whether it is greater than the current makespan estimate. If yes, then the makespan is revised to be the minimum uncovered A-cost (Algo 1: Lines 27-32). Note that this procedure invokes `find_min_delta` procedure (Algo 1: Line 27), which searches for the tuple having minimum uncovered cost and picks the one containing an A-cost over the one containing an H-cost in case there is a tie. Next, the threshold subgraph $\mathcal{G}_{th}$ is updated by adding the edges between the uncovered robot-goal pairs, provided that the corresponding A-costs do not exceed the current makespan (Algo 1: Line 22). We consider only the uncovered robot-goal pairs for the addition of new edges so as to optimize the number of A-cost computations. We attempt to maximize the current M on the updated $\mathcal{G}_{th}$ (Algo 1: Line 23). The process loops until M becomes a total matching.

Algorithm 1: Goal Assignment Optimizing Makespan for Multi-Robot Systems (**OM**).

```
Global: $C, T, P, OL, CL$
1  procedure get_optimal_makespan $(WS,\ S,\ F)$
2      $R \leftarrow |S|,\quad G \leftarrow |F|,\quad M \leftarrow \emptyset$
3      for $i \leftarrow 1$ to $R$ do
4          for $j \leftarrow 1$ to $G$ do
5              $C(i)(j) \leftarrow$ get_Euclidean_distance$(S(i), F(j))$
6              $T(i)(j) \leftarrow$ ‘$h$’,  $P(i)(j) \leftarrow [\,]$
7      if $R = G$ then
8          $acost_{Rmin} \leftarrow$ explore_min_acost$(WS, S, F, 1)$
9          $acost_{Gmin} \leftarrow$ explore_min_acost$(WS, S, F, 2)$
10         $acost_{min} \leftarrow$ concatenate$(acost_{Rmin}, acost_{Gmin})$
11     else if $R < G$ then
12         $acost_{min} \leftarrow$ explore_min_acost$(WS, S, F, 1)$
13     else
14         $acost_{min} \leftarrow$ explore_min_acost$(WS, S, F, 3)$
15     $makespan \leftarrow$ max$(acost_{min})$
16     $\mathcal{G}_{th} \leftarrow$ get_threshold_subgraph $(WS, S, F,\ makespan)$
17     $M \leftarrow$ maximize_match$(\mathcal{G}_{th}, M)$
18     while $M$ is not a total matching do
19         $\langle R_c, G_c\rangle \leftarrow$ get_min_vertex_cover$(\mathcal{G}_{th}, M)$
20         $\Delta \leftarrow$ collect_uncovered_costs$(R, G, R_c, G_c)$
21         $makespan \leftarrow$ update_makespan$(WS, S, F, \Delta,$
                                                  $makespan)$
22         $\mathcal{G}_{th} \leftarrow$ update_threshold_subgraph
                    $(\mathcal{G}_{th}, R_c, G_c,\ WS, S, F, makespan)$
23         $M \leftarrow$ maximize_match$(\mathcal{G}_{th}, M)$
24     return $\langle makespan, acost_{min}\rangle$

25 procedure update_makespan $(WS,\ S,\ F,\ \Delta,\ makespan)$
26     while True do
27         $\langle \Delta_{min}, index\rangle \leftarrow$ find_min_delta$(\Delta)$
28         $i' \leftarrow \Delta_{min}.i,\quad j' \leftarrow \Delta_{min}.j$
29         if $\Delta_{min}.t =$ ‘$a$’ then
30             if $makespan < \Delta_{min}.c$ then
31                 $makespan \leftarrow \Delta_{min}.c$
32             return $makespan$
33         else
34             $\langle c, p, OL(i'), CL(i')\rangle \leftarrow$ FRAStar
                 $(WS, S(i'), F(j'),\ OL(i'),\ CL(i'))$
35             $C(i')(j') \leftarrow c,\quad P(i')(j') \leftarrow p,$
36             $T(i')(j') \leftarrow$ ‘$a$’
37             Update: $\Delta(index) \leftarrow \langle i', j', C(i')(j'),$ ‘$a$’$\rangle$
38     return $makespan$

39 procedure explore_min_acost $(WS, S, F, flag)$
40     $R \leftarrow |S|,\quad G \leftarrow |F|$
41     if $flag = 1$ then
42         $pivot\_entity \leftarrow R,\quad scan\_entity \leftarrow G$
43     else
44         $pivot\_entity \leftarrow G,\quad scan\_entity \leftarrow R$
45     for $i \leftarrow 1$ to $pivot\_entity$ do
46         if $flag \neq 2$ then
47             $OL(i) \leftarrow [\,],\quad CL(i) \leftarrow [\,]$
48             $acost_{min}(i) \leftarrow \infty$
49         else
50             $\langle acost_{min}(i), -\rangle \leftarrow$ find_min_cost
                                   $(i, scan\_entity,$ ‘$a$’$)$
51         $\langle hcost_{min}, hindex\rangle \leftarrow$ find_min_cost
                                   $(i, scan\_entity,$ ‘$h$’$)$
52         while $(hindex \neq -1)$ &
                 $(hcost_{min} \leq acost_{min}(i))$ do
53             if $flag = 1$ then
54                 $\langle c, p, OL(i), CL(i)\rangle \leftarrow$ FRAStar $(WS, S(i),$
                    $F(hindex), OL(i), CL(i))$
55                 $C(i)(hindex) \leftarrow c,\quad P(i)(hindex) \leftarrow p$
56                 $T(i)(hindex) \leftarrow$ ‘$a$’
57             else
58                 $\langle c, p, OL(hindex), CL(hindex)\rangle \leftarrow$
                    FRAStar$(WS, S(hindex), F(i),$
                    $OL(hindex), CL(hindex))$
59                 $C(hindex)(i) \leftarrow c,\quad P(hindex)(i) \leftarrow p$
60                 $T(hindex)(i) \leftarrow$ ‘$a$’
61             $\langle hcost_{min}, hindex\rangle \leftarrow$ find_min_cost
                                   $(i, scan\_entity,$ ‘$h$’$)$
62             $\langle acost_{min}(i), -\rangle \leftarrow$ find_min_cost
                                   $(i, scan\_entity,$ ‘$a$’$)$
63     return $acost_{min}$

64 procedure collect_uncovered_costs $(R,\ G,\ R_c,\ G_c)$
65     for each uncovered robot $i \in [R] \setminus R_c$ do
66         for each uncovered goal $j \in [G] \setminus G_c$ do
67             $\Delta.add(\langle i, j, C(i)(j), T(i)(j)\rangle)$
68     return $\Delta$
```

Goal Assignment Optimizing Makespan followed by Total Cost (Algorithm 2). The algorithm introduced in [21] solves the multi-robot goal assignment problem, optimizing total cost incurred due to all the robots (without considering robot-robot collisions). Like OM, it computes the optimal obstacle-free path for any robot-goal pair on demand, i.e., if it is indeed essential for the optimality of the goal assignment. With necessary modifications, we utilize the algorithm presented in [21] to extend OM, resulting in our goal assignment algorithm **OM+OTC** (**O**ptimal **M**akespan followed by **O**ptimal **T**otal **C**ost), which we formally present in Algorithm 2.

OM+OTC computes goal assignment that has an optimal total cost among the set of assignments having the optimal makespan. At the outset, it invokes OM to compute optimal makespan for a problem instance (Algo 2: Line 2). OM+OTC intrinsically ensures that the resulting assignment does not violate the optimal makespan. To achieve this, first it gets rid of those cost values (either heuristic or actual) which exceed the optimal makespan (Algo 2: Lines 4-8). Second, if an H-cost for a robot-goal pair (that does not

Algorithm 2: Goal Assignment Optimizing Makespan followed by Total Cost (**OM+OTC**).

```
   Global: $C, T, P, OL, CL$
1  procedure solve_goal_assignment $(WS,\ S,\ F)$
2      $\langle makespan, acost_{min}\rangle \leftarrow$ get_optimal_makespan $(WS, S, F)$
3      $R \leftarrow |S|,\quad G \leftarrow |F|,\quad M \leftarrow \emptyset$
4      for $i \leftarrow 1$ to $R$ do
5          for $j \leftarrow 1$ to $G$ do
6              if $C(i)(j) > makespan$ then
7                  $C(i)(j) \leftarrow \infty$
8                  $T(i)(j) \leftarrow$ ‘$a$’
9      if $R \le G$ then
10         $\forall\, i \in [R]$: $Z_R(i) \leftarrow acost_{min}(i)$
11         $\forall\, j \in [G]$: $Z_G(j) \leftarrow 0$
12     else
13         $\forall\, i \in [R]$: $Z_R(i) \leftarrow 0$
14         $\forall\, j \in [G]$: $Z_G(j) \leftarrow acost_{min}(j)$
15     $\mathcal{G}_{eq} \leftarrow$ get_equality_subgraph$(Z_R, Z_G)$
16     $M \leftarrow$ maximize_match$(\mathcal{G}_{eq}, M)$
17     while $M$ is not a total matching do
18         $\langle R_c, G_c\rangle \leftarrow$ get_min_vertex_cover$(\mathcal{G}_{eq}, M)$
19         $\Delta \leftarrow$ collect_slacks$(R, G, R_c, G_c, Z_R, Z_G)$
20         $\delta_{min} \leftarrow$ get_min_slack$(WS, S, F, Z_R, Z_G, \Delta, makespan)$
21         uncover_actual_cost$(WS, S, F, R_c, G_c, Z_R, Z_G, \delta_{min}, makespan)$
22         if $R \le G$ then
23             $\forall\, i \in R_c$          : $Z_R(i) \leftarrow Z_R(i) - \delta_{min}$
24             $\forall\, j \in [G] \setminus G_c$: $Z_G(j) \leftarrow Z_G(j) + \delta_{min}$
25         else
26             $\forall\, i \in [R] \setminus R_c$: $Z_R(i) \leftarrow Z_R(i) + \delta_{min}$
27             $\forall\, j \in G_c$          : $Z_G(j) \leftarrow Z_G(j) - \delta_{min}$
28         $\mathcal{G}_{eq} \leftarrow$ get_equality_subgraph$(Z_R, Z_G)$
29         $M \leftarrow$ maximize_match$(\mathcal{G}_{eq}, M)$
30     return $M$

31 procedure collect_slacks $(R,\ G,\ R_c,\ G_c,\ Z_R,\ Z_G)$
32     for each uncovered robot $i \in [R] \setminus R_c$ do
33         for each uncovered goal $j \in [G] \setminus G_c$ do
34             $\delta \leftarrow C(i)(j) - (Z_R(i) + Z_G(j))$
35             $\Delta.add(\langle i, j, \delta, T(i)(j)\rangle)$
36     return $\Delta$

37 procedure get_min_slack $(WS,\ S,\ F,\ Z_R,\ Z_G,\ \Delta,\ makespan)$
38     while True do
39         $\langle \Delta_{min}, index\rangle \leftarrow$ find_min_delta$(\Delta)$
40         if $\Delta_{min}.t =$ ‘$a$’ then
41             return $\Delta_{min}.\delta$
42         else
43             $\langle c, p, OL(\Delta_{min}.i), CL(\Delta_{min}.i)\rangle \leftarrow$ FRAStar$(WS, S(\Delta_{min}.i), F(\Delta_{min}.j]), OL(\Delta_{min}.i), CL(\Delta_{min}.i))$
44             $C(\Delta_{min}.i)(\Delta_{min}.j) \leftarrow c$
45             $P(\Delta_{min}.i)(\Delta_{min}.j) \leftarrow p$
46             $T(\Delta_{min}.i)(\Delta_{min}.j) \leftarrow$ ‘$a$’
47             if $C(\Delta_{min}.i)(\Delta_{min}.j) \le makespan$ then
48                 $\delta_{new} \leftarrow C(\Delta_{min}.i)(\Delta_{min}.j) - (Z_R(i)+Z_G(j))$
49                 Update: $\Delta(index) \leftarrow \langle \Delta_{min}.i, \Delta_{min}.j, \delta_{new}, \text{‘}a\text{’}\rangle$
50             else
51                 $C(\Delta_{min}.i)(\Delta_{min}.j) \leftarrow \infty$
52                 Delete: $\Delta(index)$

53 procedure uncover_actual_cost $(WS,\ S,\ F,\ R_c,\ G_c,\ Z_R,\ Z_G,\ \delta_{min},\ makespan)$
54     $R \leftarrow |S|,\quad G \leftarrow |F|$
55     for each uncovered robot $i \in [R] \setminus R_c$ do
56         for each uncovered goal $j \in [G] \setminus G_c$ do
57             $\delta \leftarrow C(i)(j) - (Z_R(i) + Z_G(j))$
58             if $T(i)(j) =$ ‘$h$’ and $\delta = \delta_{min}$ then
59                 $\langle c, p, OL(i), CL(i)\rangle \leftarrow$ FRAStar $(WS, S(i), F(j), OL(i), CL(i))$
60                 $C(i)(j) \leftarrow c$,
61                 $P(i)(j) \leftarrow p$
62                 $T(i)(j) \leftarrow$ ‘$a$’
63                 if $C(i)(j) > makespan$ then
64                     $C(i)(j) \leftarrow \infty$
```

exceed optimal makespan), when translated to its corresponding A-cost, exceeds the optimal makespan, it is discarded from further consideration in the assignment computation (Algo 2: Lines 50-52, 63-64).

Let us further review the tailored algorithm steps from [21] that are integrated into OM+OTC, which aid in finding a goal assignment that has an optimal total cost among the set of assignments having the optimal makespan. The notion of *feasible vertex labeling*, used in [21], can be defined as: Given a bipartite graph $\mathcal{G} = ([R], [G], E)$ with bipartition $([R], [G])$, edge set $E = \{(i, j) \mid i \in [R], j \in [G]\}$, a cost function $\texttt{cost} : E \rightarrow \mathbb{R}^+$, and vertex labeling functions $Z_R : [R] \rightarrow \mathbb{R}$ and $Z_G : [G] \rightarrow \mathbb{R}$, a feasible vertex labeling refers to the labeling of vertices of $\mathcal{G}$ such that:

$$\texttt{cost}(i, j) \ge Z_R(i) + Z_G(j) \qquad \forall (i, j) \in E.$$

The minimum A-cost information ($acost_{min}$) supplied by OM is used to generate feasible labels for the robots and the goals (Algo 2: Lines 9-14). Using the labels, we proceed to formulate an *equality subgraph* $\mathcal{G}_{eq} = ([R], [G], E_{Z_R,Z_G})$, which is a subgraph of the bipartite graph $\mathcal{G} = ([R], [G], E)$ (representing the complete graph for the multi-robot system), such that its edge set E_{Z_R,Z_G} is given by:

$$E_{Z_R,Z_G} = \{(i,j) \mid Z_R(i) + Z_G(j) = \texttt{cost}(i,j)\}.$$

As Algorithm 2 computes the A-costs between the robot-goal pairs in a demand-driven way, it does not have the initial complete bipartite graph consisting of edges with A-costs for all the robot-goal pairs, from which it can derive an equality subgraph. Instead, the algorithm leaps to directly construct the equality subgraph $\mathcal{G}_{eq}$ (Algo 2: Line 15). Symbolically, the robots and the goals form the bipartition of the vertex set of $\mathcal{G}_{eq}$ (see Fig. 3(m) for an example). We compute maximum cardinality matching M in $\mathcal{G}_{eq}$ (Algo 2: Line 16). If M happens to be a total matching, OM+OTC terminates before reporting the robot-goal assignment M as output. However, if the current M is not a total matching, additional steps are taken to update M until it becomes one. The additional steps begin by finding the minimum vertex cover $\langle R_c, G_c\rangle$ corresponding to M (Algo 2: Line 18). Like before, the sets R_c and G_c refer to the covered robots and goals, respectively. The procedure `collect_slacks` gathers the *slack* δ for edges between each pair of uncovered robot and uncovered goal (Algo 2: Line 19). The slack δ for an edge (i,j) is computed as:

$$\delta = \texttt{cost}(i,j) - (Z_R(i) + Z_G(j)).$$

The collection data structure Δ stores tuples of the following information, which can be accessed using the dot operator: (i) robot ID $i \in [R]$, (ii) goal ID $j \in [G]$, (iii) δ between i and j, and (iv) cost-type $T(i)(j)$ of the cost that was used to calculate δ.

The procedure `get_min_slack` determines the minimum slack δ_{min} (from Δ) that is calculated based on the A-cost (Algo 2: Line 20). The equality subgraph should not have any edge with H-cost (since matching should be computed on A-cost edges only). Therefore, only when the update of labels (Algo 2: Lines 22-27) happens with a δ_{min} computed on A-cost, a new edge with A-cost is certain to appear in the subsequently revised equality subgraph (Algo 2: Line 28). So, if δ_{min} was computed on A-cost, it is used to update the labels. However, if δ_{min} was computed on H-cost, then the procedure computes the corresponding A-cost for the concerned robot-goal pair and updates the slack using this A-cost. Furthermore, it updates Δ with the revised slack before repeating the process to search for δ_{min} (Algo 2: Lines 42-52). Note that this procedure invokes another procedure `find_min_delta` (Algo 2: Line 39), which searches for the minimum δ and picks the one that is computed on A-cost over the one computed on H-cost in case there is a tie.

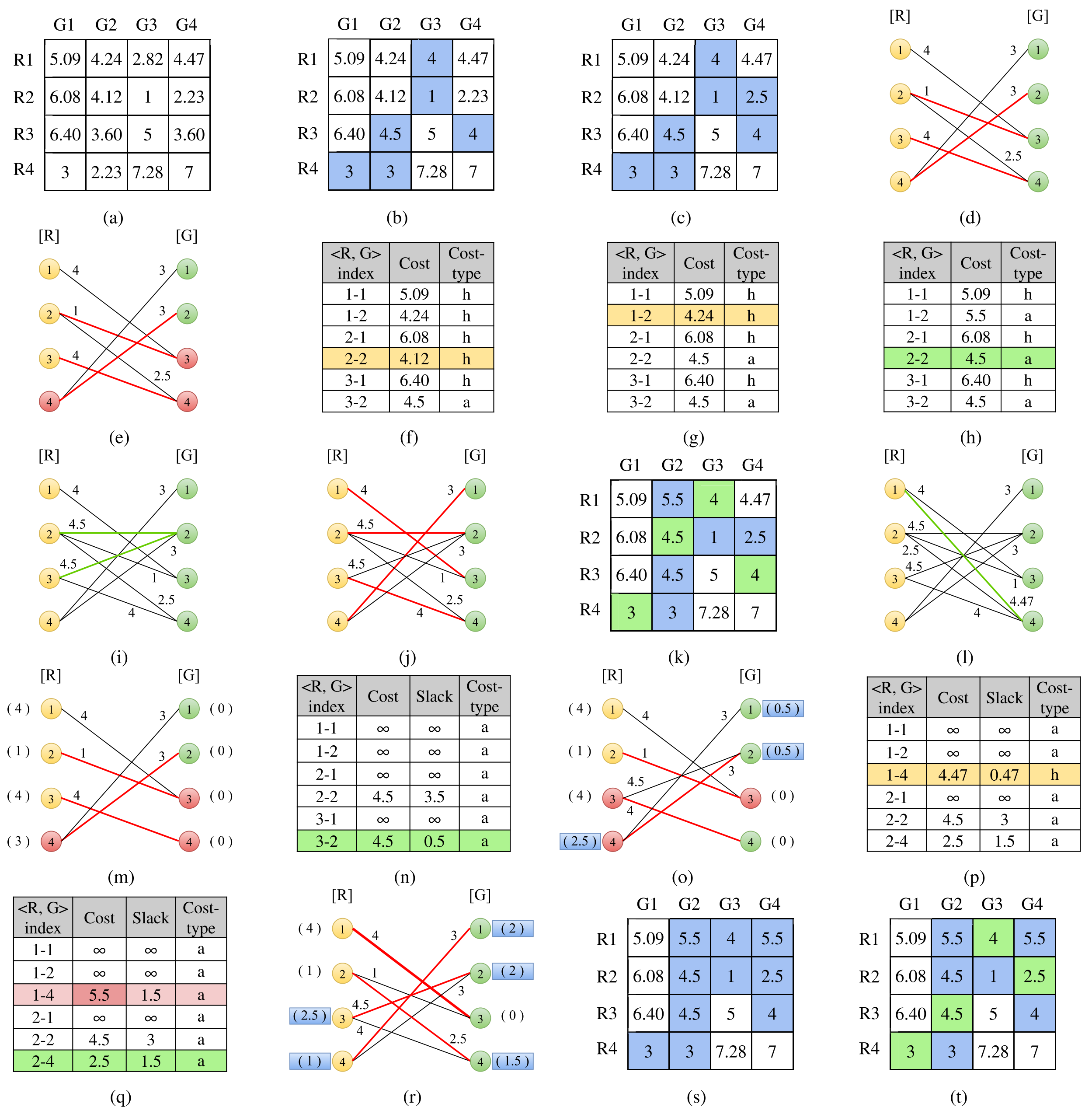
The procedure `uncover_actual_cost` replaces the H-costs by corresponding A-costs for those (uncovered robot, uncovered goal) pairs whose δ are computed on H-costs and are equal to δ_{min} (Algo 2: Lines 21, 54-64). This ensures that the explored A-costs with δ values still equal to δ_{min} can participate in the subsequent revision of $\mathcal{G}_{eq}$. The labels (Z_R and Z_G) undergo modification as follows. For $R \le G$ case, the labels of covered robots are decreased by δ_{min} (Algo 2: Line 23) whereas the labels of uncovered goals are increased by δ_{min} (Algo 2: Line 24). For $R > G$ case, the labels of uncovered robots are increased by δ_{min} (Algo 2: Line 26) whereas the labels of covered goals are decreased by δ_{min} (Algo 2: Line 27). This label-update rule ensures that a new edge in $\mathcal{G}_{eq}$ would appear only between an uncovered robot and an uncovered goal. The revision of $\mathcal{G}_{eq}$ occurs using the updated labels (Algo 2: Line 28) and an attempt is made to maximize the current M in $\mathcal{G}_{eq}$ (Algo 2: Line 29). OM+OTC checks whether M has evolved into a total matching (Algo 2: Line 17); if not, the process loops until a total matching is achieved.

4.2. Example

Fig. 3 illustrates the execution of OM+OTC on the multi-robot goal assignment problem introduced in Fig. 2(a). The execution begins with the application of OM on the problem instance. Fig. 3(a) shows the cost matrix that has an H-cost for each robot-goal pair. We employ the Euclidean distance between a robot's initial location and a goal location as the *admissible* heuristic cost. Fig. 3(b) displays the transformed cost matrix following the exploration of minimum A-cost for each robot, while Fig. 3(c) depicts the same matrix after the exploration of minimum A-cost for each goal (since $R = G$). The blue cells contain A-costs, while the white cells have H-costs.

For a particular robot / goal, its minimum H-cost is replaced by corresponding A-cost iteratively until its current minimum H-cost exceeds its current minimum A-cost. This may lead to the presence of more than one blue cell for a robot / goal in the transformed cost matrix. Let us analyze the presence of two blue cells for robot R3. Initially, $hcost_{min}$ for R3 is 3.60 (for G2). With an aim to discover $acost_{min}$ for R3, we use the FRA* search algorithm to explore the A-cost corresponding to $hcost_{min}$, which yields 4.5 (see R3-G2 cell in Fig. 3(c)). The new $hcost_{min}$ is 3.60 (for G4), which is less than the current $acost_{min}$ (4.5). So, there is a possibility that the A-cost corresponding to this new $hcost_{min}$ can be less than the current $acost_{min}$. Therefore, the A-cost for the R3-G4 pair is explored, resulting in a value of 4. Now, the new $hcost_{min}$ is 5 (for G3), which is greater than the current $acost_{min}$ (4). Consequently, there is no need to explore A-cost corresponding to the new $hcost_{min}$ as that would result in a cost greater than or equal to 5 (thus, leaving the current $acost_{min}$ unchanged to 4). This results in two blue cells for R3. There are two blue cells (i.e., two A-costs) for robot R2 also in the transformed cost matrix. The one for R2-G3 is the result of exploring the minimum A-cost for R2. The A-cost for R2-G4 is the result of exploring the minimum A-cost for G4. Now, focusing at the case of robot R4, its initial $hcost_{min}$ is 2.23 (for G2). The corresponding A-cost is 3, which turns out to be equal to the new $hcost_{min}$ of 3 for G1. The A-cost for the R4-G1 pair is explored, resulting in a value of 3. Thus, there are two blue cells for R4 too.

The initial estimation of makespan is 4 from the first formula in (1). Using the initial makespan, a threshold subgraph is configured (Fig. 3(d)). The yellow vertices on the left denote the robots, whereas the green vertices on the right denote the goals. The red edges belong to the maximum cardinality matching M. We see that the robot R1 is unmatched. As the current matching is not a total matching, we proceed according to lines 18 and 19 of OM to find the minimum vertex cover in the threshold subgraph. The minimum



(a)

	G1	G2	G3	G4
R1	5.09	4.24	2.82	4.47
R2	6.08	4.12	1	2.23
R3	6.40	3.60	5	3.60
R4	3	2.23	7.28	7

(b)

	G1	G2	G3	G4
R1	5.09	4.24	4	4.47
R2	6.08	4.12	1	2.23
R3	6.40	4.5	5	4
R4	3	3	7.28	7

(c)

	G1	G2	G3	G4
R1	5.09	4.24	4	4.47
R2	6.08	4.12	1	2.5
R3	6.40	4.5	5	4
R4	3	3	7.28	7

(f)

<R, G> index	Cost	Cost-type
1-1	5.09	h
1-2	4.24	h
2-1	6.08	h
2-2	4.12	h
3-1	6.40	h
3-2	4.5	a

(g)

<R, G> index	Cost	Cost-type
1-1	5.09	h
1-2	4.24	h
2-1	6.08	h
2-2	4.5	a
3-1	6.40	h
3-2	4.5	a

(h)

<R, G> index	Cost	Cost-type
1-1	5.09	h
1-2	5.5	a
2-1	6.08	h
2-2	4.5	a
3-1	6.40	h
3-2	4.5	a

(k)

	G1	G2	G3	G4
R1	5.09	5.5	4	4.47
R2	6.08	4.5	1	2.5
R3	6.40	4.5	5	4
R4	3	3	7.28	7

(n)

<R, G> index	Cost	Slack	Cost-type
1-1	∞	∞	a
1-2	∞	∞	a
2-1	∞	∞	a
2-2	4.5	3.5	a
3-1	∞	∞	a
3-2	4.5	0.5	a

(p)

<R, G> index	Cost	Slack	Cost-type
1-1	∞	∞	a
1-2	∞	∞	a
1-4	4.47	0.47	h
2-1	∞	∞	a
2-2	4.5	3	a
2-4	2.5	1.5	a

(q)

<R, G> index	Cost	Slack	Cost-type
1-1	∞	∞	a
1-2	∞	∞	a
1-4	5.5	1.5	a
2-1	∞	∞	a
2-2	4.5	3	a
2-4	2.5	1.5	a

(s)

	G1	G2	G3	G4
R1	5.09	5.5	4	5.5
R2	6.08	4.5	1	2.5
R3	6.40	4.5	5	4
R4	3	3	7.28	7

(t)

	G1	G2	G3	G4
R1	5.09	5.5	4	5.5
R2	6.08	4.5	1	2.5
R3	6.40	4.5	5	4
R4	3	3	7.28	7

Fig. 3. Application of OM+OTC on the problem introduced in Fig. 2(a). (a) Matrix with heuristic cost (Euclidean distance) for each robot-goal pair. (b) The transformed cost matrix following the exploration of minimum actual cost for each robot. The blue cells depict actual costs. (c) Matrix after the exploration of minimum actual cost for each goal (since $R = G$). (d) Threshold subgraph (yellow vertices on the left denote the robots, whereas the green vertices on the right denote the goals). The red edges in the threshold subgraph belong to maximum cardinality matching M. (e) The vertices in red color belong to minimum vertex cover. (f-h) Search for minimum uncovered actual cost. (i) Update threshold subgraph with edges whose actual cost do not exceed makespan estimate. (j) Recompute the maximum cardinality matching in the updated threshold subgraph, which results in all the robots getting matched. (k) Matrix illustrating the actual costs computed so far, shown in colored cells. Green cells indicate the goal assignment with an optimal makespan. (l) Residual threshold subgraph from OM augmented with edges whose cost do not exceed the optimal makespan (here, the green edge). (m) Equality subgraph with labels for each vertex depicted in parentheses. The red edges belong to maximum matching and the red vertices belong to minimum vertex cover. (n) Compute minimum slack based on actual cost. (o) Label update (shown in blue color) using the minimum slack. The maximum matching undergoes revision. As R1 is not yet matched, recompute minimum vertex cover. (p-q) Compute minimum slack based on actual cost. (r) Label update (shown in blue color) using the minimum slack. The maximum matching undergoes revision. All robots are matched. (s) Cost matrix in which blue cells depict actual costs computed so far. (t) Cost matrix in which green cells indicate the goal assignment that has an optimal total cost among the assignments having optimal makespan.

vertex cover consists of R4, G3, and G4 (red vertices in Fig. 3(e)). Here, R1, R2, and R3 are the uncovered robots, while G1 and G2 are the uncovered goals.

In Figs. 3(f)-(h), we illustrate the search for the minimum uncovered *A-cost*. Initially, 4.12 is retrieved as the minimum uncovered cost (Fig. 3(f)). But, since it is not an A-cost, it is replaced by its corresponding A-cost 4.5 (Fig. 3(g)). On searching the minimum uncovered cost again, we get 4.24, which is again an H-cost and is therefore replaced by its corresponding A-cost 5.5 (Figs. 3(g), 3(h)). When we search for minimum uncovered cost one more time, we get the value 4.5, which is an A-cost (Fig. 3(h)). As the minimum uncovered A-cost 4.5 exceeds the current estimation of makespan 4, the estimation of the makespan value is revised to 4.5. In Fig. 3(i), the threshold subgraph gets an update such that new edges (shown in green) having an A-cost not exceeding the current estimation of makespan are added. In Fig. 3(j), the maximum cardinality matching M is recomputed on the revised threshold subgraph, and here we see that all the robots are matched. Therefore, the process of finding the optimal makespan terminates with an output of 4.5. Fig. 3(k) shows the A-costs explored so far in colored cells, and the goal assignment is depicted using the green color.

Having found the optimal makespan using OM, OM+OTC proceeds to compute an assignment that has an optimal total cost among the assignments having the optimal makespan. In order to facilitate this, OM+OTC considers only those costs (whether H-costs or A-costs) between the robot-goal pairs that do not exceed the optimal makespan. This results in an augmented version of the residual threshold subgraph from OM (Fig. 3(l)). The green-colored edge is an H-cost edge with a cost of 4.47, added to the residual threshold subgraph in this process. Following the steps of the algorithm in [21], OM+OTC constructs an equality subgraph and computes a maximum cardinality matching M in it (Fig. 3(m)). The red-colored edges are in M. As we observe that robot R1 is not matched, OM+OTC finds the minimum vertex cover corresponding to M in the equality subgraph. The minimum vertex cover consists of R4, G3, and G4 (shown as red vertices in the Fig. 3(m)). Thus, R1, R2, and R3 are the uncovered robots, while G1 and G2 are the uncovered goals. The minimum slack δ_{min} among the (uncovered robot, uncovered goal) pairs is 0.5 (Fig. 3(n)). Using δ_{min}, the labels of vertices of equality subgraph undergo update by following the rule in lines 23 and 24 of Algorithm 2 (Fig. 3(o)). The update of labels enforces revision in the equality subgraph, which in turn triggers a revision in the maximum cardinality matching (Fig. 3(o)). However, we observe that R1 still remains unmatched (Fig. 3(o)). Thus, OM+OTC computes the minimum vertex cover (red vertices shown in Fig. 3(o)).

The minimum slack δ_{min} among the (uncovered robot, uncovered goal) pairs is 0.47, however, it is computed on the H-cost 4.47 for the robot-goal pair R1-G4 (Fig. 3(p)). The corresponding A-cost (5.5) for the robot-goal pair R1-G4 exceeds the optimal makespan (4.5), thus, leading to its exclusion from consideration for the assignment (Fig. 3(q)). On finding the minimum slack again yields 1.5 for R2-G4, which is computed on A-cost (Fig. 3(q)). The vertex labels, equality subgraph, and the maximum cardinality matching undergoes update once again and we find that all the robots are matched this time (Fig. 3(r)). Fig. 3(s) shows the A-costs computed up to this point in blue color. The green cells in Fig. 3(t) depict the goal assignment solution that has an optimal total cost of 14, while the individual costs in the assignment do not exceed the optimal makespan (4.5).

5. Theoretical properties

Let us examine the theoretical properties of OM and OM+OTC.

5.1. Correctness

First, we analyze the correctness of OM. We begin by stating the following auxiliary lemma (see [40] or [35] for details).

Lemma 1. *Given a bipartite graph $\mathcal{G} = ([R], [G], E)$ with bipartition ($[R]$, $[G]$), edge set $E = \{(i, j) \mid i \in [R],\ j \in [G]\}$, cost of each edge in E, and a corresponding maximum cardinality matching M, each iteration of the body of* while *loop (Algo 1: Lines 18-23 in the procedure* `get_optimal_makespan` *of OM) results in the following: the matching size $|M|$ increases, or there is no change in $|M|$ but there is an increase in $|R_c|$ (or $|G_c|$) and corresponding decrease in $|G_c|$ (or $|R_c|$), such that $|R_c| + |G_c| = |M|$.*

Note that it depends on the implementation of finding M and the corresponding minimum vertex cover $\langle R_c, G_c \rangle$ that which out of $|R_c|$ and $|G_c|$ increases. Henceforth, we would consider that $|R_c|$ increases.

Following is the theorem on the correctness of OM.

Theorem 1 *(Correctness (OM)). OM provides an optimal solution to the multi-robot goal assignment problem (Problem 1) such that either the robots or the goals, whichever is less in number, get assigned completely, and the resultant assignment has an optimal makespan.*

Proof. We prove the theorem in the following two parts:

a) OM terminates: After computing the maximum cardinality matching M for the first time (Algo 1: Line 17), OM terminates if M happens to be a total matching. Now, consider that M is not a total matching, and consequently, the instructions in the *while* loop (Algo 1: Lines 18-23) have to be executed. From Lemma 1, with each iteration of the *while* loop (Algo 1: Lines 18-23), either $|M|$ increases or $|R_c|$ increases. $|R_c|$ can increase up to R, implying that $|M|$ has to increase in the other iterations. The said loop breaks when M becomes a total matching, i.e., when $|M| = \min(R, G)$. Therefore, OM is guaranteed to terminate.

b) OM provides an optimal solution to the multi-robot goal assignment problem (Problem 1) such that either the robots or the goals, whichever is less in number, get assigned completely, and the resultant assignment has an optimal makespan: For the *while* loop

(Algo 1: Lines 18-23) in OM, let us consider the following loop invariant: *"The current estimate of makespan for the current M is optimal."* We proceed with the loop invariant proof.

Initialization: Towards the start of processing, OM computes an initial estimate of makespan using the formula in (1). When R and G are equal, each robot and goal must get assigned, and therefore, the optimal makespan must be at least the maximum of the minimum A-costs of each robot and goal. When $R < G$, each robot must get assigned. Consequently, in this case, the optimal makespan must be at least the maximum of the minimum A-costs of each robot. When $R > G$, each goal must get assigned. This implies that the optimal makespan in this case must be at least the maximum of the minimum A-costs of each goal. Thus, the initial estimate of makespan serves as a lower bound of the optimal makespan.

By definition, a threshold subgraph $\mathcal{G}_{th}$ consists of only those edges whose A-costs do not exceed the current estimate of makespan. When OM computes the maximum cardinality matching M on $\mathcal{G}_{th}$ for the first time (Algo 1: Line 17), it may or may not be a total matching. As the initial estimate of makespan is a lower bound of the optimal makespan, it implies that the initial estimate is optimal for the current M. Thus, the loop invariant is `true` prior to the first iteration of the concerned *while* loop. Note that the first iteration of the *while* loop occurs only if M is not a total matching (i.e., if M leaves at least one robot unmatched).

Maintenance: Suppose the loop invariant was `true` before the current iteration of the loop. In the current iteration, there would be at least one uncovered robot-goal pair. Note that, initially, there would not exist any edge between an uncovered robot-goal pair in $\mathcal{G}_{th}$. So, to add edges between such pairs, OM determines the minimum uncovered A-cost. There could arise two cases. First, the minimum uncovered A-cost does not exceed the current estimate of makespan. In this case, the makespan is not updated, and only the new edges are added in $\mathcal{G}_{th}$ between uncovered robot-goal pairs. Thus, the optimality of the current makespan remains intact after the current iteration. In the second case, the minimum uncovered A-cost exceeds the current makespan. This means that the current makespan is not sufficient to obtain a total matching. So, OM revises the makespan value to the minimum uncovered A-cost and allows the addition of new edges in $\mathcal{G}_{th}$ according to the new makespan value. Since the revision of makespan takes place using the *minimum* uncovered A-cost, the updated makespan is optimal for the resulting M of the current iteration. Therefore, in either case, the loop invariant holds `true` before the next iteration of the loop.

Termination: Initially, OM determines an initial estimate of makespan, which is a lower bound of the optimal makespan. Any subsequent revision in the estimate of makespan makes use of the minimum uncovered A-cost. The revision takes place only when the current M is not a total matching and the minimum uncovered A-cost exceeds the current estimate. The loop terminates when M becomes a total matching. This implies that either the set of robots, the set of goals, or both get assigned completely, and the current estimate of makespan is optimal for the total matching. □

Next, we analyze the correctness of OM+OTC. We begin by stating a well-known theorem below.

Theorem *(Kuhn-Munkres [14]). If the vertex labeling provided by Z_R and Z_G is feasible, and the maximum cardinality matching M is feasible (i.e., M is a total matching in equality subgraph with respect to Z_R and Z_G), then M is a minimum cost matching.*

We also state an auxiliary lemma (see [40] or [35] for more details).

Lemma 2. *Given a bipartite graph $\mathcal{G} = ([R],[G],E)$ with bipartition $([R], [G])$, edge set $E = \{(i,j) \mid i \in [R], j \in [G]\}$, cost of each edge in E, the feasible vertex labeling functions Z_R and Z_G, and a corresponding maximum cardinality matching M, each iteration of the body of* while *loop (Algo 2: Lines 17-29 in procedure* `solve_goal_assignment` *of Algorithm 2) results in the following: a) the labels get updated and they remain feasible, and b) the matching size $|M|$ increases, or there is no change in $|M|$ but there is an increase in $|R_c|$ (or $|G_c|$) and corresponding decrease in $|G_c|$ (or $|R_c|$), such that $|R_c| + |G_c| = |M|$.*

Note that it depends on the implementation of finding M and the corresponding minimum vertex cover $\langle R_c, G_c \rangle$ that which out of $|R_c|$ and $|G_c|$ increases. Henceforth, we would consider that $|R_c|$ increases.

Following is the theorem on the correctness of OM+OTC. We prove the theorem with the help of above results.

Theorem 2 *(Correctness (OM+OTC)). OM+OTC provides a solution to the multi-robot goal assignment problem (Problem 2) such that either the robots or the goals, whichever is less in number, get assigned completely, and the resultant assignment has an optimal makespan as well as an optimal total cost within the set of solutions having the optimal makespan.*

Proof. OM+OTC first invokes OM to compute optimal makespan for the given problem instance. According to Theorem 1, OM provides a guarantee on the optimality of the output makespan. We prove the correctness of OM+OTC in the following two parts:

a) OM+OTC terminates: From Lemma 2, with each iteration of the *while* loop (Algo 2: Lines 17-29), either $|M|$ increases or $|R_c|$ increases. $|R_c|$ can increase up to R, implying that $|M|$ has to increase in other iterations. The said loop breaks when M becomes a total matching, i.e., when $|M| = \min(R, G)$. Therefore, OM+OTC is guaranteed to terminate.

b) OM+OTC provides a solution to the multi-robot goal assignment problem (Problem 2) such that either the robots or the goals, whichever is less in number, get assigned completely, and the resultant assignment has an optimal makespan as well as an optimal total cost within the set of solutions having the optimal makespan: For the *while* loop (Algo 2: Lines 17-29), let us consider the following loop invariant:

$$\texttt{cost}(M) \leq \sum_{i\in[R]} Z_R(i) + \sum_{j\in[G]} Z_G(j)$$
$$\text{where } \texttt{cost}(M) = \sum_{(i,j)\in M} \texttt{cost}(i,j). \tag{2}$$

The invariant states that the sum of the costs of edges in the maximum cardinality matching M never exceeds the sum of the labels of all the robot and goal vertices. We proceed with the loop invariant proof for which we use $R \leq G$ case for the ease of exposition. The proof for $G < R$ case is similar.

Initialization: M is computed in equality subgraph $\mathcal{G}_{eq}$. Recall that for each edge (i, j) in $\mathcal{G}_{eq}$ ($i \in [R]$ and $j \in [G]$),

$$\texttt{cost}(i,j) = Z_R(i) + Z_G(j). \tag{3}$$

Prior to the first iteration of the *while* loop, M may or may not be a total matching. If M is not a total matching, then there is at least one robot corresponding to which there is no edge in M. This implies that the unmatched robot does not contribute to the cost of matching $\texttt{cost}(M)$, however, its label is still counted while computing the cost of labeling. So,

$$\texttt{cost}(M) < \sum_{i\in[R]} Z_R(i) + \sum_{j\in[G]} Z_G(j). \tag{4}$$

On the other hand, if M is a total matching, then there is an edge for each robot in M. In that case,

$$\texttt{cost}(M) = \sum_{i\in[R]} Z_R(i) + \sum_{j\in[G]} Z_G(j). \tag{5}$$

Thus, on combining both the cases, it is concluded that the loop invariant is `true` prior to the first iteration of the concerned *while* loop. Note that the first iteration of the *while* loop takes place only if M is not a total matching (i.e., if M leaves at least one robot unmatched).

Maintenance: Suppose the loop invariant was `true` before the current iteration of the loop. In the current iteration, we can say from Lemma 2 that there should be an increase in either $|M|$ or $|R_c|$. If an increase in $|M|$ makes M a total matching, then (5) holds. Whereas in the cases where a) the increase in $|M|$ is still not sufficient to make M a *total matching* or b) $|M|$ does not increase leaving at least one unassigned robot, (4) holds as discussed previously. Therefore, on combining all the cases, the loop invariant holds `true` before the next iteration of the loop.

Termination: The *while* loop terminates when M becomes feasible, i.e., a total matching. The initialization of labels, along with their update rules, ensures that they always remain feasible. Therefore, from the Kuhn-Munkres theorem, M is a minimum-cost matching. Moreover, costs computed for the robot-goal pairs are themselves optimal since OM+OTC employs the FRA* search algorithm for this purpose (which uses the Euclidean distance as a consistent heuristic). If the translation of an H-cost results in an A-cost that exceeds the optimal makespan, it is discarded from further consideration in the assignment computation. Thus, the solution to the multi-robot goal assignment problem (Problem 2) obtained using OM+OTC has an optimal makespan as well as optimal total cost within the set of solutions having the optimal makespan. □

5.2. Time complexity

Theorem 3 *(Time complexity). Considering* Ψ *denotes the number of free cells in the workspace, the worst-case time complexity of OM+OTC is:*

- $R = G$: $\mathcal{O}(\max(R^2\Psi, R^4))$
- $R < G$: $\mathcal{O}(\max(RG\Psi, R^3G))$
- $R > G$: $\mathcal{O}(\max(RG\Psi, RG^3))$

Proof. Let λ represent the number of motion primitives available to a robot within a grid-based workspace. In other words, a robot can move in λ directions from its current cell while adhering to the workspace boundaries. The maximum number of edges in the graph representation of such workspace is bounded by $\mathcal{O}(\lambda \times \Psi)$. Considering λ to be a constant, the number of edges in the aforementioned graph is bounded by $\mathcal{O}(\Psi)$. Note that both R and G are bounded by $\mathcal{O}(\Psi)$ since the number of robots or goals cannot exceed the number of free cells in the workspace.

Now, let us analyze the time complexities of various components within OM. The FRA* search algorithm reuses the elements stored in the open-list (OL) and the closed-list (CL) while determining optimal obstacle-free paths for a robot from its initial location to several goal locations. In the worst-case scenario, the FRA* search algorithm explores $\mathcal{O}(\Psi)$ workspace cells when computing paths for a robot between its initial location and various goal locations. Therefore, for R robots, the total number of workspace cells explored is $\mathcal{O}(R\Psi)$. When the goal to which a path needs to be computed changes for a particular robot, FRA* updates the H-cost (corresponding to the new goal) of each element present in the OL. For a single robot, this process of updating the H-costs for G goals has the time complexity of $\mathcal{O}(G\Psi)$, which becomes $\mathcal{O}(RG\Psi)$ for R robots in the multi-robot system. Therefore, the total computations by the FRA* search algorithm have a worst-case time complexity of $\mathcal{O}(RG\Psi + R\Psi)$, which is equivalent to $\mathcal{O}(RG\Psi)$.

Computing H-costs in the form of Euclidean distance for all robot-goal pairs takes $\mathcal{O}(RG)$. The worst-case time complexity for procedure `explore_min_acost` in OM (excluding the computations by FRA*) is as following:

When $flag = 1$: $\mathcal{O}(R(G + G(2G))) = \mathcal{O}(RG + RG^2) = \mathcal{O}(RG^2)$.
When $flag = 2$: $\mathcal{O}(G(R + R + R(2R))) = \mathcal{O}(2RG + 2R^2G) = \mathcal{O}(R^2G)$.
When $flag = 3$: $\mathcal{O}(G(R + R(2R))) = \mathcal{O}(RG + 2R^2G) = \mathcal{O}(R^2G)$.

The major computation in the procedure `get_threshold_subgraph` is performed by the FRA* search algorithm. Its time complexity is, therefore, bounded by the time complexity of FRA* ($\mathcal{O}(RG\Psi)$). Since the '*while*' loop (Algo 1: Lines 18-23) terminates when M becomes a total matching, from Lemma 1, the loop's execution count is bounded by $\mathcal{O}((\min(R,G))^2)$. In the worst-case scenario, the body of the 'while' loop executes for the maximum possible number of times. This implies that, in such a case, the `maximize_match` procedure makes minimal progress in each iteration — either increasing $|M|$ by one, or if $(R \leq G)$, increasing $|R_c|$, or if $(G < R)$, increasing $|G_c|$. Consequently, other procedures in the loop such as `get_min_vertex_cover`, `update_makespan`, and `update_threshold_subgraph` also exhibit minimal progress. Since a bipartite graph with R robots and G goals contains at most $R \times G$ edges, therefore, on excluding the computations by FRA*, each of the procedures `get_min_vertex_cover`, `collect_uncovered_costs`, `update_makespan`, `update_threshold_subgraph`, and `maximize_match` has a worst-case time complexity of $\mathcal{O}(RG)$ for a single iteration of the 'while' loop. Thus, excluding the computations by FRA*, all procedures within the 'while' loop (Algo 1: Lines 18-23) have a worst-case time complexity of $\mathcal{O}((\min(R,G))^2 \times (RG))$.

Effectively, OM comprises of the following four components:

1. Compute H-costs.
2. Compute A-costs using FRA*.
3. Find initial estimate of makespan (using `explore_min_acost`).
4. Compute optimal makespan (using '*while*' loop (Algo 1: Lines 18-23)).

Therefore, the overall time complexity of OM for the following cases is:

When $R = G$: $\mathcal{O}([RG] + [RG\Psi] + [RG^2 + R^2G] + [(\min(R,G))^2 \times (RG)]) = \mathcal{O}(\max(R^2\Psi, R^4))$.
When $R < G$: $\mathcal{O}([RG] + [RG\Psi] + [RG^2] + [(\min(R,G))^2 \times (RG)]) = \mathcal{O}(\max(RG\Psi, R^3G))$.
When $R > G$: $\mathcal{O}([RG] + [RG\Psi] + [R^2G] + [(\min(R,G))^2 \times (RG)]) = \mathcal{O}(\max(RG\Psi, RG^3))$.

Let us examine the time complexity of the remaining part of OM+OTC.

As discussed earlier, the FRA* search algorithm has a worst-case time complexity of $\mathcal{O}(RG\Psi)$. The time complexity of the thresholding operation (Algo 2: Lines 4-8) is bounded by $\mathcal{O}(RG)$. The initial feasible labeling (Algo 2: Lines 9-14) has a worst-case time complexity of $\mathcal{O}(\max(R,G))$. Excluding the computations by FRA*, the time complexities of the procedures `get_equality_subgraph` and `maximize_match` are bounded by $\mathcal{O}(RG)$. Since the '*while*' loop (Algo 2: Lines 17-29) terminates when M becomes a total matching, the loop's execution count is bounded by $\mathcal{O}((\min(R,G))^2)$. Thus, excluding the computations by FRA*, all the procedures within the '*while*' loop (Algo 2: Lines 17-29) have the worst-case time complexity of $\mathcal{O}((\min(R,G))^2 \times (RG))$. We note that OM, being the initial step in OM+OTC, has a worst-case time complexity that dominates the overall worst-case time complexity of OM+OTC. Therefore, the worst-case time complexity of OM+OTC is the same as that of OM. □

6. Evaluation

In this section, we present the results obtained from our detailed experimental evaluation of OM+OTC.

6.1. Experimental setup

Baselines. For the evaluation of OM+OTC (which also captures OM), we consider two algorithms as the baseline. The first algorithm, which we refer to as Base-1, computes optimal obstacle-free paths *for all* robot-goal pairs by executing Dijkstra's shortest path algorithm [41] once for each robot and then uses the dual method to solve the LBA problem. It further uses the Hungarian method to optimize the total cost of the assignment such that the individual costs do not exceed the optimal makespan. The second algorithm, which we refer to as Base-2, is the *Bottleneck Assignment* algorithm in [20] (Algorithm 3 in [20]) that considers H-costs in increasing order for the lazy evaluation of A-costs so as to compute an assignment that has an optimal makespan as well as optimal total cost within the set of assignments having the optimal makespan. The original implementation of Base-2 by the authors uses the breadth-first search with 4 motion primitives on an unweighted graph. However, as we consider 8 and 26 motion primitives in the 2D and 3D environments, respectively, on a weighted graph with costs proportional to their Euclidean distance, we implement Base-2 with the more efficient FRA* search algorithm. We implement the baseline algorithms (Base-1 and Base-2) and the proposed algorithm in Python. The source code of implementation is available at https://github.com/iitkcpslab/H-Scalable-MRGA-Makespan-TC.git.

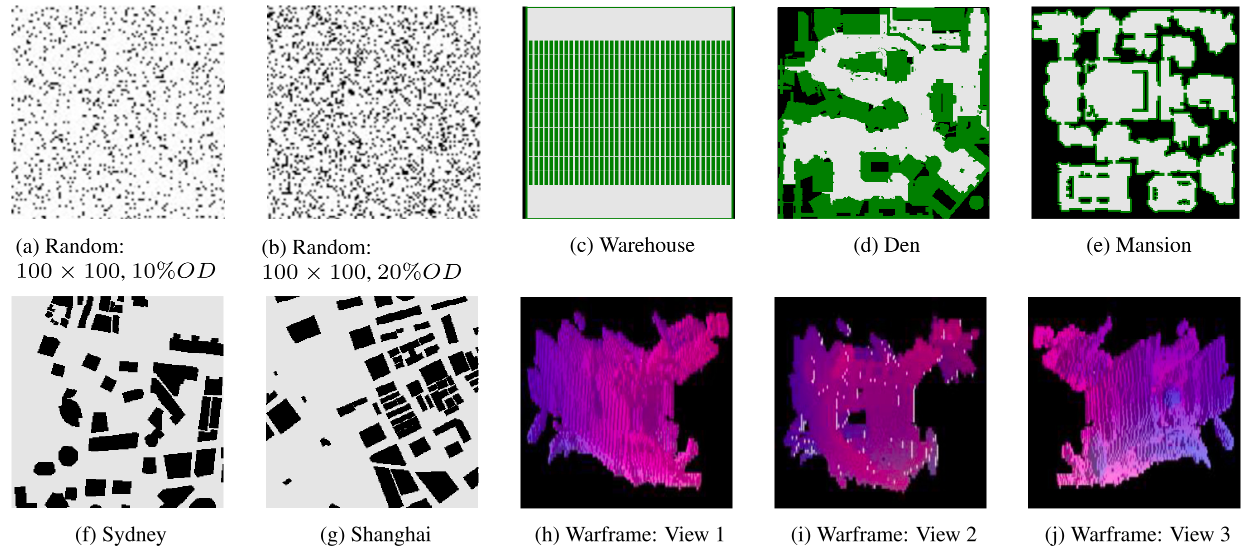


Fig. 4. Sample random maps (a)–(b) and benchmark maps (c)–(j).

Benchmarks. We evaluate our algorithm on randomly generated 2D workspaces and benchmark 2D and 3D workspaces [29,6] (see Fig. 4). The benchmark 2D workspaces comprise of structured indoor environments, such as warehouse and mansion, as well as unstructured outdoor environments, like den and cities. The benchmark 3D workspace resembles an environment in Warframe game [42], featuring debris from a destroyed spaceship.

We run all the experiments in a desktop machine with Intel® Core^{TM} i7-8700 CPU @ 3.20 GHz processor, 32 GB RAM, and Ubuntu 20.04 OS. We run each experiment for 20 times, where for each run, we randomly generate the set of start locations for robots and the set of goal locations. We report the mean and standard deviation for the evaluation metrics.

6.2. *Experimental results*

Here, we present various experimental results that illustrate the performance of our proposed algorithm OM+OTC.

6.2.1. *Analyzing total cost: a comparative study of OM, OM+OTC, and OTC*

OM offers goal assignment solutions that optimize mission time or makespan for multi-robot systems. However, it does not control the quality of solutions with respect to total cost incurred due to all robots. This means that efficient completion of a mission timewise might not necessarily equate to a cost-effective solution.

To address this concern, our proposed algorithm, OM+OTC, helps to determine an assignment that optimizes the total cost among a set of assignments having an optimal makespan. To assess the benefits of this optimization, we perform a comparative analysis of the total cost between the assignment solutions obtained using OM, OM+OTC, and OTC on the benchmark workspaces. In the results illustrated in Fig. 5, we observe that not only the total cost of the assignment generated by OM+OTC is consistently much smaller than the total cost of the solution generated by OM, but also, it is consistently close to that of the solution generated by OTC. Thus, we deduce that OM+OTC plays a vital role in diminishing the total costs borne by multi-robot systems to near-optimal values. And this advantage escalates with the growing number of robots in any multi-robot application.

6.2.2. *Comparison of computation time with baselines*

We use three evaluation metrics: (a) **NumExp**: the number of robot-goal pairs for which the A-cost is computed, (b) **NumNodeExp (%)**: the percentage of OL nodes expanded with respect to Base-1, and (c) **Runtime**: the execution time. We also report the **Speedup** that our algorithm achieves over the two baseline algorithms.

Randomly Generated Workspaces. In Table 2, we report the results of experiments on randomly generated 2D workspaces having an equal number of robots and goals. In the first two blocks of the table, we increase the workspace size while keeping R constant at 200 and 400, respectively. Owing to the demand-driven computation of A-costs by OM+OTC, its NumExp is significantly less when compared with Base-1 (which computes $R \times G$ A-costs) and even when compared with Base-2 due to the algorithmic differences. The speedup with respect to Base-1 increases with an increase in the workspace size. Computing the paths for all the robot-goal pairs leads to a significant increase in Base-1's runtime. Though there is also an increase in the runtime of OM+OTC, it is not proportional to that of Base-1. This is because there is a consistent decrease in NumExp of OM+OTC due to the fact that the number of free cells per robot increases in the workspace, which tends to reduce the gap between H-costs and corresponding A-costs, leading to lesser number of A-costs computations. Thus, the impact of the increase in the computation time of longer paths (in the increased workspace size) is compensated by the decrease in NumExp. The speedup with respect to Base-2 sees a decline because of the following reason. Base-2

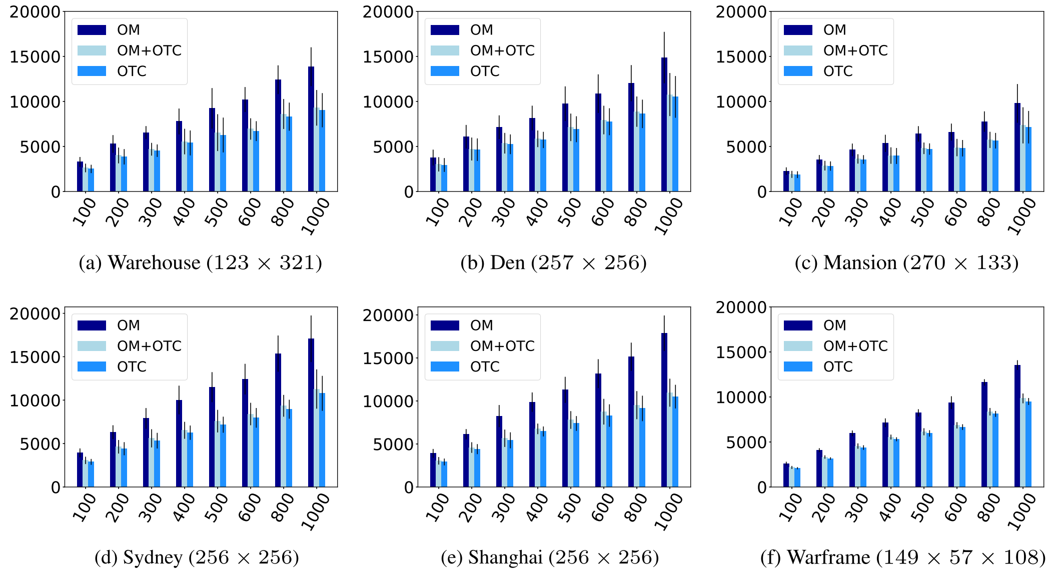


Fig. 5. Comparison of Total Cost between OM, OM+OTC, and OTC for Benchmark Workspace (*X-axis:* Number of Robots, *Y-axis:* Total Cost (in units)).

has a non-trivial overhead of computing the maximum cardinality matching in the bipartite graph after every single addition of an edge. With an increase in the workspace size, NumExp for Base-2 decreases (due to the same reason as for OM+OTC), which implies a reduction in the number of times the maximum cardinality matching is computed. Therefore, the runtime of Base-2 does not increase in proportion to that of OM+OTC.

Obstacle density (OD) (i.e., the percentage of cells in a workspace that are occupied by obstacles) varies in the next two blocks. The speedup with respect to Base-1 declines with an increase in OD, and this is because of the following reason. The runtime of Base-1 decreases since with more obstacles in a fixed-size workspace, Dijkstra's shortest path algorithm has fewer cells to process. However, OM+OTC's runtime sees a rise as the growth in NumExp dominates the benefit gained from the reduction in the number of cells to process. Note that with an increase in OD, the difference between an H-cost and its corresponding A-cost tends to increase, which leads to an increase in NumExp. The speedup with respect to Base-2 does not vary significantly because Base-2 has as equivalent impact as OM+OTC on increasing the OD.

The number of robots R varies in the last two blocks. We observe that with an increase in R, first there is a growth in the speedup relative to Base-1, which later experiences a decline. The reason for this behavior is the following. Initially, for the smaller values of R in the given workspace size, the time to compute goal assignment does not dominate the time to compute robot-goal paths and the corresponding A-costs. And since our approach computes quite a few paths when compared with Base-1, there is an increase in the speedup for smaller values of R. But as R continues to increase, the time to compute goal assignment in our approach starts to dominate the time to compute paths due to the inherent time complexities. While for Base-1, the increase in the time needed to solve goal assignment is meager compared to the increase in the time required to compute paths (as Base-1 computes the paths for all robot-goal pairs). This results in the decline in the speedup with respect to Base-1, as R continues to increase. We notice that the speedup with respect to Base-2 increases consistently in the last two blocks. It is because of the following. Base-2 adds edges in the bipartite graph in ascending order of A-cost, and tries to compute total matching after every single addition of an edge. This approach makes Base-2 inefficient when compared with OM+OTC that computes the A-costs on demand and also computes the total matching for a reasonable number of times by using the minimum vertex cover information. Thus, an increase in R implies an increase in the number of edges in the bipartite graph, which inflates the efficiency of OM+OTC against Base-2 (notice the increasing difference between NumExp of Base-2 and OM+OTC).

Table 3 presents the results for cases having unequal number of robots and goals in random workspaces. We observe that OM+OTC not only achieves an order of magnitude speedup with respect to Base-1 but also consistently outperforms Base-2 across all cases.

Standard Benchmark Workspaces. Through the plots in Fig. 6, we compare the performance of OM+OTC with that of the baseline algorithms for standard workspaces available in the literature [29,6]. The last plot is for a 3D workspace resembling an environment in Warframe game [42]. We take 15 min and 2 h as timeouts for the 2D and 3D workspaces, respectively. OM+OTC outperforms the baselines on both structured workspaces (e.g., warehouse and mansion) and unstructured workspaces (e.g., den and cities). Base-1 exceeds the timeout for all the problem instances considered in the 3D workspace. Note that the speedup with respect to both Base-1 and Base-2 follows the same trend of the last two super-rows (where the number of robots varies) in Table 2.

For the 2D workspaces, we observe that Base-1 outperforms Base-2 for high robot density (large number of robots in a smaller size workspace) despite the fact that Base-1 computes the paths for all robot-goal pairs whereas Base-2 computes them lazily for some of the pairs. When there are many robots and goals in the workspace, the bipartite graph used for assignment has a large number of edges. As Base-2 computes the maximum cardinality matching after every single addition of an edge, it turns out to be a significant

Table 2
Experimental Results on Random Workspace.

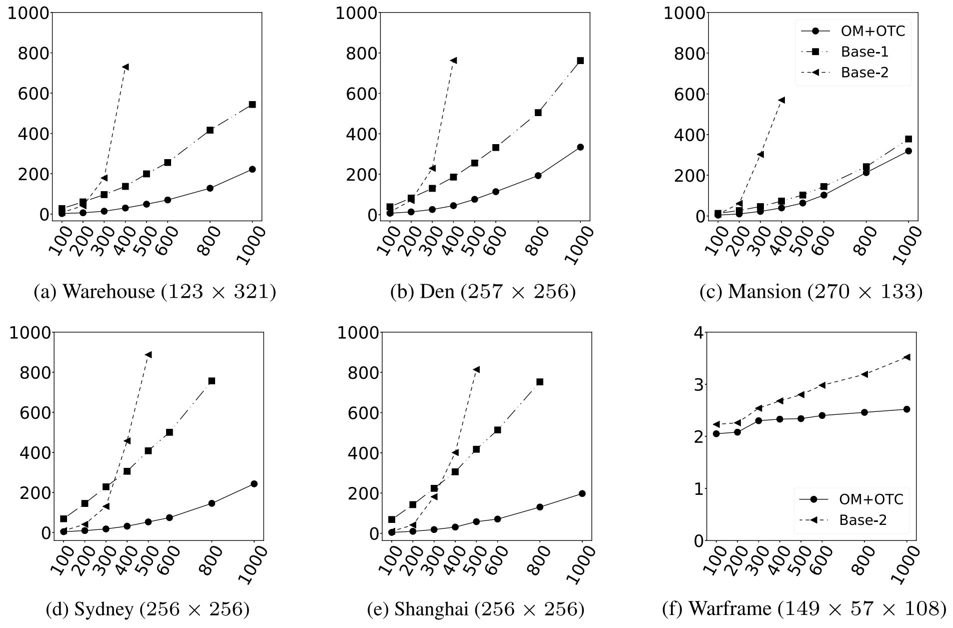
WS Size	OD	R	NumExp		NumNodeExp (%)		Runtime (s)				Speedup	
			OM+OTC	Base 2	OM+OTC	Base 2	OM	OM+OTC	Base 1	Base 2	Base 1	Base 2
100^2	20	200	$2938_{\pm 567}$	$4287_{\pm 589}$	$3.20_{\pm 0.88}$	$5.34_{\pm 1.88}$	$1.57_{\pm 0.23}$	$3.18_{\pm 0.45}$	$21.34_{\pm 0.39}$	$21.48_{\pm 6.42}$	6.71	6.75
200^2	20	200	$2599_{\pm 252}$	$4143_{\pm 604}$	$2.37_{\pm 0.34}$	$4.41_{\pm 0.93}$	$3.23_{\pm 0.60}$	$5.87_{\pm 1.05}$	$83.56_{\pm 0.92}$	$25.65_{\pm 9.73}$	14.24	4.37
300^2	20	200	$2517_{\pm 509}$	$3887_{\pm 852}$	$2.16_{\pm 0.67}$	$3.96_{\pm 1.35}$	$5.47_{\pm 1.75}$	$8.41_{\pm 2.16}$	$196.40_{\pm 2.10}$	$31.18_{\pm 15.65}$	23.35	3.71
400^2	20	200	$2422_{\pm 391}$	$3800_{\pm 901}$	$2.03_{\pm 0.54}$	$3.84_{\pm 1.35}$	$8.59_{\pm 2.44}$	$12.65_{\pm 3.31}$	$355.91_{\pm 5.36}$	$32.62_{\pm 11.29}$	28.14	2.58
100^2	20	400	$6639_{\pm 1007}$	$9401_{\pm 1285}$	$2.07_{\pm 0.60}$	$3.11_{\pm 0.83}$	$4.71_{\pm 1.38}$	$11.44_{\pm 1.89}$	$49.46_{\pm 2.23}$	$230.52_{\pm 96.99}$	4.32	20.15
200^2	20	400	$6059_{\pm 854}$	$8617_{\pm 1388}$	$1.45_{\pm 0.27}$	$2.33_{\pm 0.53}$	$8.49_{\pm 2.29}$	$20.01_{\pm 3.72}$	$180.77_{\pm 4.28}$	$243.67_{\pm 114.51}$	9.03	12.18
300^2	20	400	$5688_{\pm 809}$	$8414_{\pm 1379}$	$1.26_{\pm 0.25}$	$2.17_{\pm 0.50}$	$13.01_{\pm 3.51}$	$27.68_{\pm 6.12}$	$433.21_{\pm 7.39}$	$221.47_{\pm 109.58}$	15.65	8.00
400^2	20	400	$5456_{\pm 696}$	$8245_{\pm 1245}$	$1.19_{\pm 0.21}$	$2.11_{\pm 0.46}$	$19.00_{\pm 5.68}$	$37.57_{\pm 9.79}$	$732.56_{\pm 12.01}$	$241.50_{\pm 116.64}$	19.50	6.43
100^2	10	200	$2150_{\pm 313}$	$3479_{\pm 658}$	$1.72_{\pm 0.34}$	$3.35_{\pm 0.90}$	$1.29_{\pm 0.23}$	$2.68_{\pm 0.50}$	$25.53_{\pm 0.58}$	$18.29_{\pm 8.16}$	9.53	6.82
100^2	15	200	$2468_{\pm 348}$	$3521_{\pm 506}$	$2.14_{\pm 0.38}$	$3.49_{\pm 0.73}$	$1.31_{\pm 0.27}$	$2.71_{\pm 0.49}$	$22.98_{\pm 0.79}$	$17.80_{\pm 9.50}$	8.48	6.57
100^2	20	200	$3187_{\pm 620}$	$4203_{\pm 998}$	$3.37_{\pm 0.82}$	$4.83_{\pm 1.60}$	$1.51_{\pm 0.35}$	$2.98_{\pm 0.66}$	$21.29_{\pm 0.77}$	$23.39_{\pm 15.48}$	7.14	7.85
100^2	25	200	$3457_{\pm 748}$	$4808_{\pm 799}$	$4.59_{\pm 1.39}$	$6.79_{\pm 1.72}$	$1.66_{\pm 0.24}$	$3.14_{\pm 0.44}$	$19.23_{\pm 0.53}$	$18.57_{\pm 7.54}$	6.12	5.91
200^2	10	400	$5013_{\pm 917}$	$7433_{\pm 1183}$	$0.97_{\pm 0.26}$	$1.71_{\pm 0.41}$	$8.15_{\pm 2.18}$	$19.19_{\pm 4.30}$	$216.12_{\pm 3.63}$	$226.21_{\pm 119.36}$	11.26	11.79
200^2	15	400	$5378_{\pm 591}$	$8403_{\pm 1060}$	$1.15_{\pm 0.18}$	$2.13_{\pm 0.38}$	$9.17_{\pm 2.30}$	$21.19_{\pm 3.93}$	$199.30_{\pm 4.15}$	$273.04_{\pm 127.69}$	9.41	12.89
200^2	20	400	$6179_{\pm 928}$	$9503_{\pm 1988}$	$1.52_{\pm 0.33}$	$2.68_{\pm 0.78}$	$10.37_{\pm 3.23}$	$22.60_{\pm 5.59}$	$177.38_{\pm 3.73}$	$281.87_{\pm 179.92}$	7.85	12.47
200^2	25	400	$7623_{\pm 1064}$	$10861_{\pm 1787}$	$2.74_{\pm 0.96}$	$3.88_{\pm 1.68}$	$10.81_{\pm 4.00}$	$24.05_{\pm 6.37}$	$164.67_{\pm 5.72}$	$242.30_{\pm 105.26}$	6.85	10.07
150^2	20	50	$430_{\pm 55}$	$583_{\pm 116}$	$5.05_{\pm 1.15}$	$7.96_{\pm 2.44}$	$0.64_{\pm 0.15}$	$0.74_{\pm 0.16}$	$10.66_{\pm 0.18}$	$1.23_{\pm 0.38}$	14.41	1.66
150^2	20	100	$1011_{\pm 96}$	$1418_{\pm 235}$	$3.30_{\pm 0.55}$	$5.34_{\pm 1.47}$	$1.00_{\pm 0.14}$	$1.42_{\pm 0.20}$	$22.24_{\pm 0.35}$	$2.74_{\pm 0.73}$	15.66	1.93
150^2	20	200	$2671_{\pm 503}$	$3832_{\pm 732}$	$2.48_{\pm 0.63}$	$4.02_{\pm 1.08}$	$2.15_{\pm 0.63}$	$4.04_{\pm 0.94}$	$46.72_{\pm 1.18}$	$18.91_{\pm 9.56}$	11.56	4.68
150^2	20	300	$4226_{\pm 594}$	$6328_{\pm 1514}$	$1.84_{\pm 0.38}$	$3.12_{\pm 1.03}$	$3.70_{\pm 0.85}$	$8.37_{\pm 1.68}$	$73.33_{\pm 1.99}$	$72.81_{\pm 40.90}$	8.76	8.70
150^2	20	400	$6276_{\pm 1302}$	$9007_{\pm 1686}$	$1.61_{\pm 0.44}$	$2.56_{\pm 0.66}$	$6.72_{\pm 2.24}$	$15.24_{\pm 3.81}$	$101.52_{\pm 2.84}$	$249.66_{\pm 142.37}$	6.66	16.38
300^2	20	50	$413_{\pm 67}$	$562_{\pm 108}$	$4.77_{\pm 1.20}$	$7.63_{\pm 2.31}$	$2.29_{\pm 0.65}$	$2.59_{\pm 0.65}$	$46.23_{\pm 0.61}$	$4.17_{\pm 1.32}$	17.85	1.61
300^2	20	100	$987_{\pm 137}$	$1451_{\pm 259}$	$3.03_{\pm 0.62}$	$5.31_{\pm 1.54}$	$3.13_{\pm 0.60}$	$3.86_{\pm 0.72}$	$93.26_{\pm 0.74}$	$7.30_{\pm 2.29}$	24.16	1.89
300^2	20	200	$2626_{\pm 666}$	$3785_{\pm 842}$	$2.34_{\pm 0.86}$	$3.86_{\pm 1.26}$	$6.10_{\pm 2.28}$	$9.21_{\pm 2.70}$	$192.74_{\pm 2.10}$	$26.20_{\pm 13.37}$	20.93	2.84
300^2	20	300	$4144_{\pm 739}$	$6236_{\pm 1170}$	$1.65_{\pm 0.46}$	$2.90_{\pm 0.85}$	$8.63_{\pm 1.97}$	$16.83_{\pm 3.40}$	$299.28_{\pm 4.41}$	$94.92_{\pm 49.78}$	17.78	5.64
300^2	20	400	$5827_{\pm 882}$	$8811_{\pm 1345}$	$1.35_{\pm 0.29}$	$2.36_{\pm 0.52}$	$15.33_{\pm 3.71}$	$31.99_{\pm 5.52}$	$409.00_{\pm 9.93}$	$258.06_{\pm 110.78}$	12.79	8.07

The columns represent the following information: **WS Size**: The workspace size; **OD**: Denotes obstacle density, which refers to the percentage of workspace cells that are occupied by obstacles; **R**: The number of robots; **NumExp**: The number of robot-goal pairs for which A-cost is computed; **NumNodeExp (%)**: The percentage of OL nodes expanded with respect to Base-1; **Runtime (s)**: The execution time in seconds; **Speedup**: The speedup in runtime achieved by OM+OTC over the respective baselines.

Table 3
Results for $R \neq G$ cases in Random Workspace.

WS Size	OD	R	G	NumExp		NumNodeExp (%)		Runtime (s)				Speedup	
				OM+OTC	Base 2	OM+OTC	Base 2	OM	OM+OTC	Base 1	Base 2	Base 1	Base 2
200^2	20	100	150	$844_{\pm 207}$	$1088_{\pm 408}$	$2.12_{\pm 1.35}$	$2.99_{\pm 1.95}$	$1.09_{\pm 0.56}$	$1.34_{\pm 0.58}$	$42.85_{\pm 1.10}$	$2.08_{\pm 1.06}$	31.98	1.55
		150	200	$1383_{\pm 316}$	$1827_{\pm 301}$	$1.79_{\pm 0.99}$	$2.82_{\pm 1.99}$	$1.47_{\pm 0.57}$	$2.10_{\pm 0.64}$	$64.20_{\pm 1.78}$	$4.09_{\pm 1.40}$	30.57	1.95
		200	250	$2033_{\pm 493}$	$2744_{\pm 484}$	$2.41_{\pm 1.45}$	$3.53_{\pm 2.22}$	$2.55_{\pm 1.07}$	$3.71_{\pm 1.23}$	$86.14_{\pm 1.95}$	$8.72_{\pm 2.52}$	23.22	2.35
		200	300	$1932_{\pm 586}$	$2583_{\pm 750}$	$1.73_{\pm 1.30}$	$2.34_{\pm 1.47}$	$2.04_{\pm 1.09}$	$2.77_{\pm 1.13}$	$94.56_{\pm 3.43}$	$6.70_{\pm 2.03}$	34.14	2.42
200^2	20	150	100	$811_{\pm 258}$	$1062_{\pm 302}$	$1.39_{\pm 1.03}$	$2.02_{\pm 1.40}$	$1.00_{\pm 0.55}$	$1.16_{\pm 0.64}$	$63.26_{\pm 0.76}$	$1.92_{\pm 0.94}$	54.53	1.66
		200	150	$1347_{\pm 237}$	$1827_{\pm 411}$	$1.37_{\pm 1.08}$	$1.97_{\pm 1.33}$	$1.40_{\pm 0.74}$	$1.86_{\pm 0.93}$	$87.44_{\pm 1.71}$	$3.77_{\pm 1.52}$	47.01	2.03
		250	200	$1904_{\pm 386}$	$2845_{\pm 663}$	$1.71_{\pm 1.26}$	$3.18_{\pm 2.98}$	$2.20_{\pm 1.15}$	$3.10_{\pm 1.38}$	$113.52_{\pm 2.32}$	$8.86_{\pm 3.91}$	36.62	2.86
		300	200	$1896_{\pm 244}$	$2768_{\pm 948}$	$1.38_{\pm 1.24}$	$1.91_{\pm 1.75}$	$2.26_{\pm 1.54}$	$2.75_{\pm 1.61}$	$149.56_{\pm 3.93}$	$7.99_{\pm 5.88}$	54.39	2.91

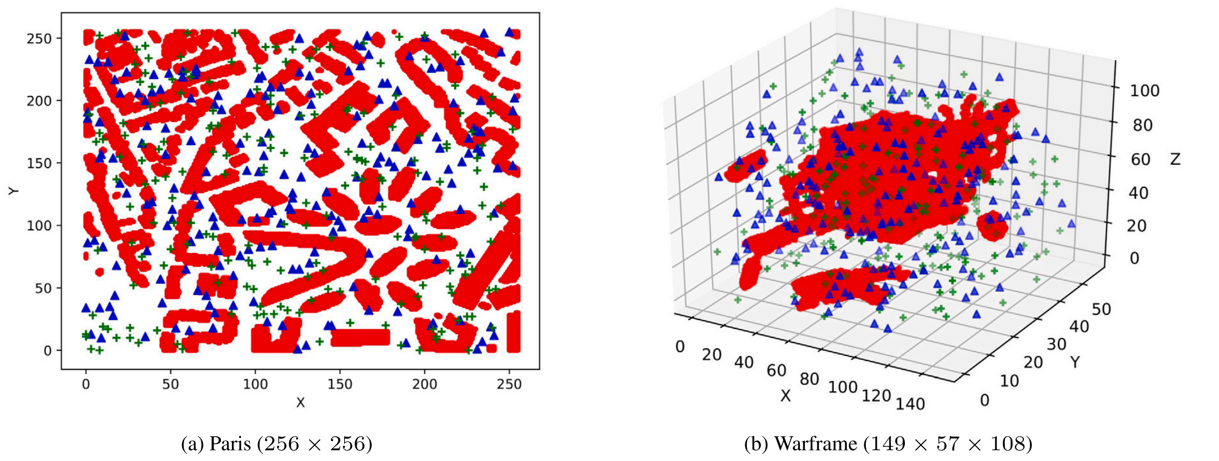
The columns represent the following information: **WS Size**: The workspace size; **OD**: Denotes obstacle density, which refers to the percentage of workspace cells that are occupied by obstacles; **R**: The number of robots; **G**: The number of goals; **NumExp**: The number of robot-goal pairs for which A-cost is computed; **NumNodeExp (%)**: The percentage of OL nodes expanded with respect to Base-1; **Runtime** (s): The execution time in seconds; **Speedup**: The speedup in runtime achieved by OM+OTC over the respective baselines.



(a) Warehouse (123 × 321) (b) Den (257 × 256) (c) Mansion (270 × 133)

(d) Sydney (256 × 256) (e) Shanghai (256 × 256) (f) Warframe (149 × 57 × 108)

Fig. 6. Scalability plots of OM+OTC for Benchmark Workspace (*X-axis:* Number of Robots, *Y-axis:* Runtime(s) for a-e; log Runtime(s) for f).



(a) Paris (256 × 256) (b) Warframe (149 × 57 × 108)

Fig. 7. Sample problem instance. Each consists of: (i) obstacles (red masses), (ii) 200 robots' initial locations (blue triangle markers), and (iii) 200 goal locations (green plus markers).

overhead which could dominate the path computation time when the robot density is high. However, for the 3D benchmark, the robot density is low and thus Base-2's performance is significantly better than Base-1, which computes the paths *for all* the robot-goal pairs, requiring exorbitant amount of time in such a large workspace.

Figs. 7(a) and 7(b) display a sample problem instance in the 2D workspace Paris and in the 3D workspace Warframe, respectively, each consisting of 200 robots' initial locations and 200 goal locations.

6.2.3. Improving TSWAP using OM+OTC

To illustrate that our goal assignment algorithm OM+OTC can help improve the state-of-the-art algorithms for the AMAPF problem, we consider the offline TSWAP [20] which solves the AMAPF problem by first finding an initial goal assignment (GA) and then using a suboptimal path planning (PP) module to plan collision-free paths for the robots. We replace its *Bottleneck Assignment* algorithm (Algorithm 3 in [20], which we refer to as Base-2) by OM+OTC. In TSWAP, the path planner uses 4 motion primitives (for 2D workspace) with equal costs and a 'stay' primitive. Instead, we use 8 motion primitives with different costs and a 'stay' primitive. We take the cost of the 'stay' motion primitive as the cost of movement of the other robot that caused the stay. We present the experimental results in Table 4. As the number of robots and goals in the workspace increases, the mean makespan decreases due to the heightened likelihood of having a robot positioned closer to each goal. The mean makespan shows no significant difference before and after PP, as mitigating robot-robot collisions rarely alter the optimal makespan. We also observe that solving goal assignment takes majority

Table 4
Improving TSWAP using OM+OTC.

WS Size	R	Makespan		Total Cost		GA Runtime (s)		PP Runtime (s)	Total Runtime (s)		Overall Speedup
		Before PP	After PP	Before PP	After PP	OM+OTC	Base 2		OM+OTC	Base 2	
Warehouse	100	$64.54_{\pm 14.51}$	$64.54_{\pm 14.51}$	$2611.34_{\pm 478.48}$	$2613.28_{\pm 478.87}$	$2.48_{\pm 0.70}$	$7.46_{\pm 3.97}$	$0.05_{\pm 0.01}$	$2.53_{\pm 0.70}$	$7.51_{\pm 3.98}$	2.97
123×321	200	$50.10_{\pm 13.44}$	$50.10_{\pm 13.44}$	$4000.86_{\pm 878.16}$	$4005.74_{\pm 879.39}$	$6.29_{\pm 1.34}$	$41.24_{\pm 18.63}$	$0.16_{\pm 0.06}$	$6.45_{\pm 1.34}$	$41.40_{\pm 18.61}$	6.42
	400	$39.32_{\pm 8.52}$	$39.70_{\pm 8.89}$	$5545.82_{\pm 1419.11}$	$5561.12_{\pm 1422.35}$	$29.98_{\pm 7.91}$	$730.03_{\pm 466.74}$	$0.41_{\pm 0.11}$	$30.39_{\pm 7.90}$	$730.44_{\pm 466.72}$	24.04
Den	100	$74.74_{\pm 13.57}$	$75.08_{\pm 13.56}$	$3016.16_{\pm 794.97}$	$3018.62_{\pm 795.91}$	$7.07_{\pm 2.16}$	$13.26_{\pm 4.22}$	$0.06_{\pm 0.02}$	$7.13_{\pm 2.16}$	$13.32_{\pm 4.22}$	1.87
257×256	200	$60.62_{\pm 13.84}$	$61.02_{\pm 14.08}$	$4715.76_{\pm 1273.70}$	$4723.32_{\pm 1276.38}$	$13.69_{\pm 4.89}$	$70.41_{\pm 49.23}$	$0.16_{\pm 0.06}$	$13.85_{\pm 4.90}$	$70.57_{\pm 49.24}$	5.10
	400	$47.56_{\pm 7.95}$	$48.80_{\pm 8.64}$	$5850.02_{\pm 924.64}$	$5868.26_{\pm 927.89}$	$44.60_{\pm 9.89}$	$762.92_{\pm 441.04}$	$0.48_{\pm 0.13}$	$45.08_{\pm 9.90}$	$763.40_{\pm 441.02}$	16.93
Mansion	100	$50.74_{\pm 10.25}$	$51.14_{\pm 10.31}$	$1899.18_{\pm 390.94}$	$1904.94_{\pm 392.30}$	$4.12_{\pm 0.53}$	$7.93_{\pm 1.63}$	$0.04_{\pm 0.01}$	$4.16_{\pm 0.53}$	$7.97_{\pm 1.63}$	1.92
270×133	200	$41.22_{\pm 7.90}$	$42.48_{\pm 8.09}$	$2868.80_{\pm 537.09}$	$2887.06_{\pm 541.14}$	$10.01_{\pm 1.74}$	$61.28_{\pm 28.02}$	$0.10_{\pm 0.02}$	$10.11_{\pm 1.75}$	$61.38_{\pm 28.02}$	6.07
	400	$34.76_{\pm 5.43}$	$38.86_{\pm 6.90}$	$3990.00_{\pm 915.53}$	$4052.22_{\pm 930.91}$	$39.58_{\pm 8.54}$	$569.39_{\pm 216.85}$	$0.27_{\pm 0.05}$	$39.85_{\pm 8.55}$	$569.66_{\pm 216.86}$	14.30
Sydney	100	$73.14_{\pm 9.31}$	$73.24_{\pm 9.28}$	$3065.30_{\pm 421.89}$	$3066.46_{\pm 421.87}$	$4.41_{\pm 0.82}$	$9.40_{\pm 3.17}$	$0.05_{\pm 0.01}$	$4.46_{\pm 0.82}$	$9.45_{\pm 3.17}$	2.12
256×256	200	$50.34_{\pm 6.31}$	$50.58_{\pm 6.59}$	$4632.28_{\pm 779.88}$	$4635.90_{\pm 780.34}$	$9.81_{\pm 1.67}$	$41.17_{\pm 18.94}$	$0.16_{\pm 0.03}$	$9.97_{\pm 1.68}$	$41.33_{\pm 18.94}$	4.15
	400	$43.78_{\pm 6.82}$	$44.16_{\pm 6.98}$	$6527.86_{\pm 977.03}$	$6537.90_{\pm 978.79}$	$31.94_{\pm 5.88}$	$457.41_{\pm 336.63}$	$0.44_{\pm 0.07}$	$32.38_{\pm 5.88}$	$457.85_{\pm 336.64}$	14.14
Shanghai	100	$71.30_{\pm 10.02}$	$71.35_{\pm 9.98}$	$3043.00_{\pm 438.75}$	$3044.12_{\pm 438.97}$	$4.53_{\pm 1.43}$	$10.18_{\pm 4.27}$	$0.07_{\pm 0.01}$	$4.60_{\pm 1.43}$	$10.25_{\pm 4.27}$	2.23
256×256	200	$55.48_{\pm 8.39}$	$55.88_{\pm 8.83}$	$4586.02_{\pm 618.77}$	$4589.98_{\pm 619.58}$	$9.78_{\pm 2.18}$	$41.45_{\pm 19.54}$	$0.21_{\pm 0.05}$	$9.99_{\pm 2.18}$	$41.66_{\pm 19.54}$	4.17
	400	$42.75_{\pm 7.08}$	$43.30_{\pm 7.42}$	$6755.50_{\pm 615.46}$	$6766.00_{\pm 615.01}$	$30.93_{\pm 6.72}$	$401.42_{\pm 240.36}$	$0.65_{\pm 0.06}$	$31.58_{\pm 6.72}$	$402.07_{\pm 240.36}$	12.73
Warframe	200	$28.83_{\pm 0.82}$	$28.83_{\pm 0.82}$	$3330.78_{\pm 189.88}$	$3330.78_{\pm 189.88}$	$120.16_{\pm 21.34}$	$182.93_{\pm 22.48}$	$0.11_{\pm 0.01}$	$120.27_{\pm 21.34}$	$183.04_{\pm 22.48}$	1.52
$149 \times 57 \times 108$	400	$24.63_{\pm 1.42}$	$24.80_{\pm 1.59}$	$5546.93_{\pm 271.63}$	$5547.43_{\pm 271.86}$	$212.52_{\pm 61.80}$	$483.69_{\pm 137.66}$	$0.44_{\pm 0.05}$	$212.96_{\pm 61.80}$	$484.13_{\pm 137.66}$	2.27

The columns represent the following information: **WS Size**: The workspace size; **R**: The number of robots; **Makespan (Before PP)**: Optimal makespan value before collision-free path planning; **Makespan (After PP)**: Optimal makespan value after collision-free path planning; **Total Cost (Before PP)**: Total cost of assignment solution generated by OM+OTC before collision-free path planning; **Total Cost (After PP)**: Total cost of assignment solution after collision-free path planning; **GA Runtime (s)**: Time in seconds required to compute goal assignment; **PP Runtime (s)**: Time in seconds required for collision-free path planning; **Total Runtime (s)**: Sum of times needed to compute goal assignment and collision-free paths; **Overall Speedup**: The speedup in total runtime achieved by OM+OTC over Base-2.

of the total runtime of the AMAPF solution process, and thus, plugging OM+OTC into the algorithm for AMAPF provides significant speedup.

7. Conclusion

We have presented a scalable centralized algorithm to solve the goal assignment problem optimally for multi-robot systems. The solution generated by our algorithm is guaranteed to have an optimal makespan and then an optimal total cost among all the solutions with an optimal makespan. The evaluation of our algorithm in comparison to two suitably chosen baselines in both 2D and 3D environments establishes its superiority to the state-of-the-art methods. The total cost of the assignment generated by our method is consistently much smaller than the total cost of the solution generated by the algorithm optimizing only makespan, which establishes the practical relevance of the problem studied in this paper. Though our solution has not been designed to ensure collision avoidance among the robots, we have shown that our algorithm can be easily plugged into a decoupled method like TSWAP for the AMAPF problem, leading to an order of magnitude speed up in computing the goal assignment and collision-free paths for a large number of robots.

The goal assignment problem addressed in this paper is relevant for time-critical applications where the mission needs to be completed as soon as possible. For example, the priority mail delivery system has to dispatch all the mail and goods in a time-bound manner. On the other hand, there could be less time-critical applications, such as the standard mail and goods delivery system, where the objective is to reduce the cost of the service with less concern about the mission completion time. For such applications, the other variant of the multi-objective goal assignment problem, where the total cost becomes the primary objective and the makespan becomes the secondary objective, is pertinent. One could also consider coexisting tasks among which some are time-critical and others are not. In this setting, one needs to study the goal assignment problem where we have to solve two variants of the problem — one with makespan as the primary objective and the other with the total cost as the primary objective — concurrently. In our future research, we plan to work on these variants of the multi-objective goal assignment problems.

CRediT authorship contribution statement

Aakash: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Indranil Saha:** Writing – review & editing, Validation, Supervision, Project administration, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] J. Li, A. Tinka, S. Kiesel, J.W. Durham, T.K.S. Kumar, S. Koenig, Lifelong multi-agent path finding in large-scale warehouses, in: AAAI, 2021, pp. 11272–11281.
[2] S.N. Das, S. Nath, I. Saha, Omcorp: an online mechanism for competitive robot prioritization, in: ICAPS, 2021, pp. 112–121.
[3] Y.-T. Tian, M. Yang, X.-Y. Qi, Y.-M. Yang, Multi-robot task allocation for fire-disaster response based on reinforcement learning, in: 2009 International Conference on Machine Learning and Cybernetics, vol. 4, 2009, pp. 2312–2317.
[4] P. Gonzalez-de-Santos, A. Ribeiro, C. Fernandez-Quintanilla, F. Lopez-Granados, M. Brandstoetter, S. Tomic, S. Pedrazzi, A. Peruzzi, G. Pajares, G. Kaplanis, M. Perez-Ruiz, C. Valero, J. del Cerro, M. Vieri, G. Rabatel, B. Debilde, Fleets of robots for environmentally-safe pest control in agriculture, Precis. Agric. 18 (2017) 574–614.
[5] P. Grippa, D.A. Behrens, F. Wall, C. Bettstetter, Drone delivery systems: job assignment and dimensioning, Auton. Robots 43 (2) (2019) 261–274.
[6] 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, R. Bartak, Multi-agent pathfinding: definitions, variants, and benchmarks, SoCS (2019) 151–158.
[7] J. Yu, S.M. LaValle, Multi-agent path planning and network flow, in: Algorithmic Foundations of Robotics X, Springer, 2013, pp. 157–173.
[8] H. Ma, S. Koenig, Optimal target assignment and path finding for teams of agents, in: AAMAS, 2016, pp. 1144–1152.
[9] W. Hönig, S. Kiesel, A. Tinka, J. Durham, N. Ayanian, Conflict-based search with optimal task assignment, in: AAMAS, 2018, pp. 757–765.
[10] G. Sharon, R. Stern, A. Felner, N.R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artif. Intell. 219 (2015) 40–66.
[11] D.R. Fulkerson, I.L. Glicksberg, O.A. Gross, A Production-Line Assignment Problem, The Rand Corporation, Santa Monica, California, 1953.
[12] O. Gross, The bottleneck assignment problem, Tech. Rep. P-1620, The Rand Corporation, Santa Monica, California, 1959.
[13] H.W. Kuhn, The Hungarian method for the assignment problem, Nav. Res. Logist. Q. 2 (1–2) (1955) 83–97.
[14] J. Munkres, Algorithms for the assignment and transportation problems, J. Soc. Ind. Appl. Math. 5 (1) (1957) 32–38.
[15] P. MacAlpine, E. Price, P. Stone, Scram: scalable collision-avoiding role assignment with minimal-makespan for formational positioning, in: AAAI, 2015, pp. 2096–2102.
[16] M. Turpin, N. Michael, V. Kumar, Trajectory planning and assignment in multirobot systems, in: Algorithmic Foundations of Robotics X: Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics, Springer, 2013, pp. 175–190.
[17] M. Turpin, K. Mohta, N. Michael, V. Kumar, Goal assignment and trajectory planning for large teams of aerial robots, in: RSS, 2013, pp. 401–415.
[18] M. Turpin, N. Michael, V. Kumar, Concurrent assignment and planning of trajectories for large teams of interchangeable robots, in: ICRA, 2013, pp. 842–848.
[19] M. Turpin, K. Mohta, N. Michael, V. Kumar, Goal assignment and trajectory planning for large teams of interchangeable robots, Auton. Robots 37 (4) (2014) 401–415.
[20] K. Okumura, X. Défago, Solving simultaneous target assignment and path planning efficiently with time-independent execution, Artif. Intell. 321 (2023) 103946.
[21] Aakash, I. Saha, It costs to get costs! A heuristic-based scalable goal assignment algorithm for multi-robot systems, in: ICAPS, AAAI Press, 2022, pp. 2–10.
[22] Aakash, I. Saha, Optimal makespan in a minute timespan! A scalable multi-robot goal assignment algorithm for minimizing mission time, in: AAAI, AAAI Press, 2024, pp. 10280–10287.

[23] R. Burkard, M. Dell'Amico, S. Martello, Assignment Problems, Society for Industrial and Applied Mathematics, USA, 2009.
[24] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P.A. Beardsley, R. Siegwart, Optimal reciprocal collision avoidance for multiple non-holonomic robots, in: DARS 2010, Lausanne, Switzerland, 2010, pp. 203–216.
[25] J. Snape, J. Van Den Berg, S.J. Guy, D. Manocha, Smooth and collision-free navigation for multiple robots under differential-drive constraints, in: IROS, 2010, pp. 4584–4589.
[26] J.P. van den Berg, J. Snape, S.J. Guy, D. Manocha, Reciprocal collision avoidance with acceleration-velocity obstacles, in: ICRA, 2011, pp. 3475–3482.
[27] D. Hennes, D. Claes, W. Meeussen, K. Tuyls, Multi-robot collision avoidance with localization uncertainty, in: AAMAS, 2012, pp. 147–154.
[28] Y.F. Chen, M. Liu, M. Everett, J.P. How, Decentralized non-communicating multi-agent collision avoidance with deep reinforcement learning, in: ICRA, 2017, pp. 285–292.
[29] N. Sturtevant, Benchmarks for grid-based pathfinding, IEEE Trans. Comput. Intell. AI Games 4 (2) (2012) 144–148.
[30] R.E. Burkard, E. Cela, Linear assignment problems and extensions, in: Handbook of Combinatorial Optimization, Springer, 1999, pp. 75–149.
[31] H.-L. Choi, L. Brunet, J.P. How, Consensus-based decentralized auctions for robust task allocation, IEEE Trans. Robot. 25 (4) (2009) 912–926.
[32] S. Giordani, M. Lujak, F. Martinelli, A distributed algorithm for the multi-robot task allocation problem, in: Trends in Applied Intelligent Systems, 2010, pp. 721–730.
[33] S. Giordani, M. Lujak, F. Martinelli, A distributed multi-agent production planning and scheduling framework for mobile robots, Comput. Ind. Eng. 64 (1) (2013) 19–30.
[34] L. Liu, D. Shell, A distributable and computation-flexible assignment algorithm: from local task swapping to global optimality, in: RSS, 2012, pp. 257–264.
[35] S. Chopra, G. Notarstefano, M. Rice, M. Egerstedt, A distributed version of the Hungarian method for multirobot assignment, IEEE Trans. Robot. 33 (4) (2017) 932–947.
[36] C.E. Leiserson, R.L. Rivest, T.H. Cormen, C. Stein, Introduction to Algorithms, vol. 3, MIT Press, Cambridge, MA, USA, 1994.
[37] D. Silver, Cooperative pathfinding, Aiide 1 (2005) 117–122.
[38] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107.
[39] J.A. Bondy, U.S.R. Murty, Graph Theory with Applications, vol. 290, Macmillan, London, 1976.
[40] A. Frank, On Kuhn's Hungarian method—a tribute from Hungary, Nav. Res. Logist. 52 (1) (2005) 2–5.
[41] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1) (1959) 269–271.
[42] D. Brewer, N.R. Sturtevant, Benchmarks for pathfinding in 3d voxel space, SoCS (2018).