Optimal Makespan in a Minute Timespan! A Scalable Multi-Robot Goal Assignment Algorithm for Minimizing Mission Time

Aakash and Indranil Saha

Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India
{aakashp, isaha}@cse.iitk.ac.in

Abstract

We study a variant of the multi-robot goal assignment problem where a unique goal to each robot needs to be assigned while minimizing the largest cost of movement among the robots, called makespan. A significant step in solving this problem is to find the cost associated with the robot-goal pairs, which requires solving a complex path planning problem. We present OM, a scalable optimal algorithm that solves the multi-robot goal assignment problem by computing the paths for a significantly less number of robot-goal pairs compared to the state-of-the-art algorithms, leading to a computationally superior mechanism to solve the problem. 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 (Li et al. 2021; Das, Nath, and Saha 2021), disaster response (Tian et al. 2009), precision agriculture (Gonzalez-de-Santos et al. 2017), mail and goods delivery (Grippa et al. 2019), 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 goal such that the largest cost of movement (makespan) for the robots to reach their assigned goal locations is minimized. The problem is a variant of the Anonymous Multi-Agent Path Finding (AMAPF) problem (Stern et al. 2019), where we decide the one-to-one robot-goal assignment and find the corresponding collision-free paths to optimize the makespan.

The AMAPF problem for optimizing makespan 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 robot-goal assignment and the corresponding collision-free paths concurrently. These are mainly the flow-based approaches (Yu and LaValle 2013; Ma and Koenig 2016) that lack scalability due to the extensive size of the flow network, which renders them ineffective for a problem with more than a few robots.

The papers in the second category start with finding an initial robot-goal assignment by solving the so-called *linear bottleneck assignment* (LBA) problem (Fulkerson, Glicksberg, and Gross 1953; Gross 1959), and then find the collision-free paths for this assignment. Among these works, SCRAM (MacAlpine, Price, and Stone 2015) 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. To solve the LBA problem in realistic settings where the environment has obstacles, either the costs of all robot-goal pairs are computed by finding the path for each pair (Turpin et al. 2013; Turpin, Michael, and Kumar 2013; Turpin et al. 2014), or they are computed in a lazy manner (Okumura and Défago 2022). 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 LBA problem, which prevent them from scaling up to large workspaces and large number of robots.

In this paper, our aim is to solve the goal assignment problem for a large multi-robot system with hundreds of robots within a short duration. 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 based on the independent optimal path for each robot-goal pair. An efficient goal assignment algorithm enables the robots to embark on their individual optimal paths towards their respective goals quickly and deal with the collisions with other robots and any dynamic obstacles using their local collision avoidance mechanism (Alonso-Mora et al. 2010; Snape et al. 2010; van den Berg et al. 2011; Hennes et al. 2012; Chen et al. 2017). Also, an efficient goal assignment algorithm, when plugged into the algorithms presented in (Turpin et al. 2013; Turpin, Michael, and Kumar 2013; Turpin et al. 2014; Okumura and Défago 2022), provides a more efficient solution to the AMAPF problem.

We propose a *scalable centralized* algorithm **OM** that solves the multi-robot goal assignment problem for **O**ptimal **M**akespan. It is based on the bipartite graph-based implementation of the dual method to solve the LBA problem (Burkard, Dell'Amico, and Martello 2009). However, unlike this algorithm, we do not assume that the costs for all the edges 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 implement our algorithm in Python and evaluate it through thorough experimentation on randomly generated 2D workspaces and standard 2D and 3D workspaces. 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 (Turpin et al. 2013, 2014). The second is the bottleneck assignment algorithm in TSWAP (Okumura and Défago 2022) 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.

Our contributions can be summarized as follows:

- We present a scalable algorithm OM for the multi-robot goal assignment problem. Without considering collision avoidance among the robots, the solution provided by our algorithm is optimal with respect to makespan.
- We implement our algorithm in Python and carry out extensive experimentation. Our results show that OM consistently outperforms two state-of-the-art methods in terms of computation time by a significant margin for both randomly generated 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.

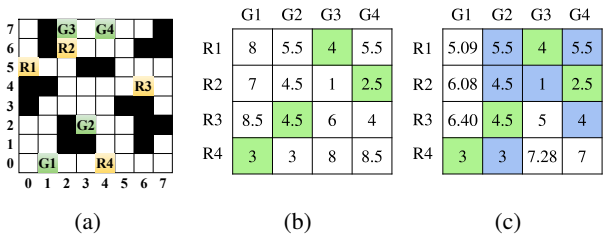
2 Problem

2.1 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 3D workspace is a 26-connected grid. 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 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.



(a) (b) (c)

Figure 1: An example problem

2.2 Problem Definition

We define the problem formally in this subsection.

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 $\texttt{cost}(i, j)$ denote the cost of movement between $s_i \in S$ and $f_j \in F$, where $i \in [R]$ and $j \in [G]$. Find a robot-goal assignment for the multi-robot application such that the largest cost of movement among all the robots (without considering the overhead for robot-robot collision avoidance) is minimized.

The cost information needs to be known to solve the above problem. 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 chokes the scalability. We aim to design an algorithm that computes these costs judiciously for only the necessary robot-goal pairs while finding an optimal solution to the goal assignment problem.

2.3 Example

Consider a multi-robot application in the 8×8 workspace shown in Figure 1(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. Figure 1(b) has a cost matrix with an actual cost for each robot-goal pair. Here, the green-colored cells depict the goal assignment that has an optimal makespan. Can we obtain this assignment without

having to compute all the actual costs? Figure 1(c) shows a cost matrix, which is an outcome of our approach. Here, only the colored cells have actual costs, and we still have the same optimal assignment as in Figure 1(b).

3 Algorithm

In this section, we present our graph theoretic algorithm to solve the multi-robot goal assignment problem. Problem 1 can be expressed in graph theoretic form as follows. Given a bipartite graph $\mathcal{G} = ([R], [G], E)$ with bipartition ($[R]$, $[G]$) and a cost function $\texttt{cost} : E \rightarrow \mathbb{R}^+$, find a maximum cardinality matching M such that the maximum cost of the edges (makespan) in M is minimized. Mathematically,

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

The readers are referred to (Leiserson et al. 1994) for the graph theoretic concepts, such as bipartite graph, matching, and vertex cover.

3.1 Algorithm Description

We present our goal assignment algorithm **OM** formally in Algorithm 1. It uses 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. For an efficient computation of optimal obstacle-free path and the corresponding A-cost, we implement *Forward Resumable* A* (FRA*) after drawing inspiration from Reverse Resumable A* (Silver 2005). It reuses the information present in the open-list (OL) and the closed-list (CL) while computing the paths from a robot's start location to several goal locations.

We now describe OM in detail. 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 (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 (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 ($acost_{Gmin}$) (Lines 7-15):

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

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* (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 of 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 (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` (Line 50). It is because, for a particular goal, some H-costs may have got replaced by corresponding A-cost while finding the minimum A-cost for each robot. `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 (Lines 55 and 59), (ii) update the cost-type in matrix T (Lines 56 and 60), and (iii) keep track of the number of path explorations in the variable $numexp$ (Line 61).

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

$$\begin{aligned}
E_{th} = \{(i,j) \mid \texttt{cost-type}(i,j) = \text{`}a\text{'} \text{ and }\\
\texttt{cost}(i,j) \leq makespan\}.
\end{aligned}$$

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 $\mathcal{G}_{th}$ (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 returns the optimal makespan as output and terminates. 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 $\langle R_c, G_c \rangle$ (Line 19), which can be computed for a bipartite graph in polynomial time by utilizing M (Bondy and Murty 1976). The sets R_c and G_c refer to the covered robots and goals, respectively. The

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

Global: $C, T, P, numexp, OL, CL$

1 **procedure** `get_optimal_makespan` *(WS, S, F)*
2 $R \leftarrow |S|,\ \ G \leftarrow |F|,\ \ M \leftarrow \emptyset$
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** $makespan$

25 **procedure** `update_makespan` *(WS, S, F, Δ, $makespan$)*
26 **while** *True* **do**
27 $\langle \Delta_{min}, index \rangle \leftarrow$ `find_min_delta`(Δ)
28 $i' \leftarrow \Delta_{min}.i,\ \ j' \leftarrow \Delta_{min}.j$
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 $C(i')(j') \leftarrow c,\ P(i')(j') \leftarrow p,\ T(i')(j') \leftarrow$ 'a'
36 $numexp \leftarrow numexp + 1$
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 $R \leftarrow |S|,\ \ G \leftarrow |F|$
41 **if** $flag = 1$ **then**
42 $pivot\_entity \leftarrow R,\ \ scan\_entity \leftarrow G$
43 **else**
44 $pivot\_entity \leftarrow G,\ \ scan\_entity \leftarrow R$
45 **for** $i \leftarrow 1$ *to* $pivot\_entity$ **do**
46 **if** $flag \neq 2$ **then**
47 $OL(i) \leftarrow [\,],\ \ CL(i) \leftarrow [\,]$
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 $C(i)(hindex) \leftarrow c,\ \ P(i)(hindex) \leftarrow p$
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 $C(hindex)(i) \leftarrow c,\ \ P(hindex)(i) \leftarrow p$
60 $T(hindex)(i) \leftarrow$ 'a'
61 $numexp \leftarrow numexp + 1$
62 $\langle hcost_{min}, hindex \rangle \leftarrow$ `find_min_cost`$(i, scan\_entity,$ 'h'$)$
63 $\langle acost_{min}(i), - \rangle \leftarrow$ `find_min_cost`$(i, scan\_entity,$ 'a'$)$
64 **return** $acost_{min}$

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

procedure `collect_uncovered_costs` collects the costs of the edges between all the uncovered robot-goal pairs (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 (Lines 27-32). Note that this procedure invokes `find_min_delta` procedure (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 (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 $\mathcal{G}_{th}$ (Line 23). The process loops until M becomes a total matching.

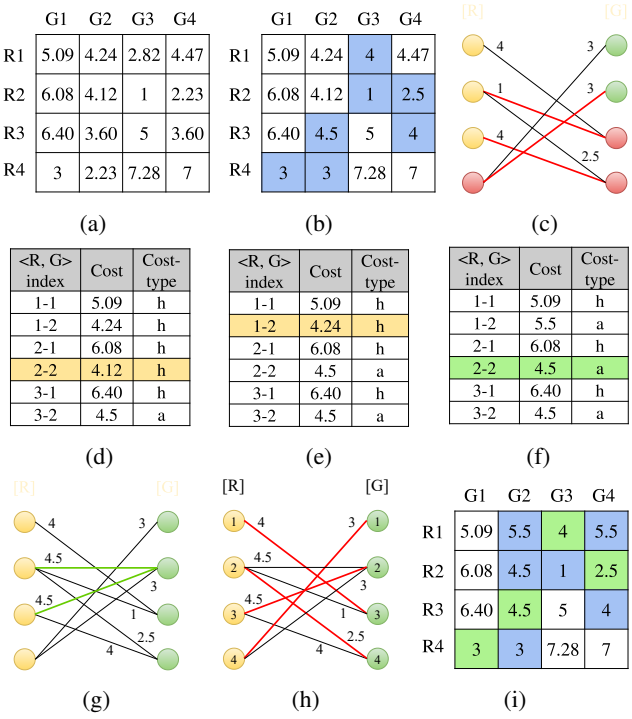


Figure 2: Illustration of OM on the problem introduced in Figure 1(a)

3.2 Example

Figure 2 illustrates the execution of OM on the multi-robot goal assignment problem introduced in Figure 1(a). Figure 2(a) shows the cost matrix that has an H-cost for each robot-goal pair. Figure 2(b) shows the transformed cost matrix after the exploration of the minimum A-cost for each robot and goal (since $R = G$). The blue cells have A-costs, while the white cells have H-costs. For a particular robot / goal, its minimum H-cost is replaced by the 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.

The initial makespan is 4 from the first formula in (1). Using it, a threshold subgraph is configured (Figure 2(c)). The vertices on the left denote the robots, whereas the ones 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 vertex cover consists of R4, G3 and G4 (shown as red vertices in the Figure 2(c)).

In Figures 2(d)-(f), we illustrate the search process for the minimum uncovered *A-cost*. Initially, 4.12 is retrieved as the minimum uncovered cost. But, since it is not an A-cost, it is replaced by its corresponding A-cost 4.5. 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. When we search for minimum uncovered cost one more time, we get the value 4.5, which is an A-cost. As the minimum uncovered A-cost 4.5 exceeds the current makespan 4, the makespan is revised to 4.5. In Figure 2(g), the threshold subgraph gets an update such that new edges (shown in green) that have an A-cost not exceeding the current makespan are added. In Figure 2(h), M is recomputed on the updated 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. Figure 2(i) shows the A-costs explored so far in colored cells, and the assignment is shown in green.

3.3 Theoretical Guarantees

OM provides the following guarantees.

Theorem 1 (Correctness)**.** *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.*

Theorem 2 (Time complexity)**.** *Considering Ψ denotes the number of free cells in the workspace, the worst-case time complexity of OM is: (a) $R = G$: $\mathcal{O}(\max(R^2\Psi, R^4))$, (b) $R < G$: $\mathcal{O}(\max(RG\Psi, R^3G))$, (c) $R > G$: $\mathcal{O}(\max(RG\Psi, RG^3))$.*

4 Evaluation

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

4.1 Experimental Setup

Baselines. For the evaluation of OM, we consider two algorithms as the baseline. The first algorithm, which we refer to as Base-1, computes the optimal obstacle-free path *for each* robot-goal pair by using Dijkstra's shortest path algorithm (Dijkstra 1959) and then uses the dual method to solve the LBA problem. The second algorithm, which we refer to as Base-2, is the *Bottleneck Assignment* algorithm in (Okumura and Défago 2022) that uses lazy evaluation of A-costs to compute the optimal makespan. The original implementation of Base-2 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[1].

Benchmarks and Evaluation Metrics. We evaluate OM on randomly generated 2D workspaces and benchmark 2D and 3D workspaces (Sturtevant 2012; Stern et al. 2019). 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 the OL nodes

[1]The source code of implementation is available at https://github.com/iitkcpslab/H-Scalable-MRGA-Makespan.git

WS Size	OD	R	NumExp		NumNodeExp (%)		Runtime (s)			Speedup	
			OM	Base 2	OM	Base 2	OM	Base 1	Base 2	Base 1	Base 2
100^2	20	400	$6148_{\pm 1133}$	$9361_{\pm 1294}$	$1.87_{\pm 0.62}$	$3.09_{\pm 0.83}$	$4.71_{\pm 1.38}$	$44.08_{\pm 1.94}$	$225.30_{\pm 97.00}$	9.36	47.83
200^2	20	400	$5659_{\pm 943}$	$8603_{\pm 1396}$	$1.31_{\pm 0.29}$	$2.32_{\pm 0.53}$	$8.49_{\pm 2.29}$	$171.03_{\pm 3.13}$	$235.05_{\pm 114.36}$	20.14	27.69
300^2	20	400	$5298_{\pm 886}$	$8408_{\pm 1381}$	$1.14_{\pm 0.27}$	$2.16_{\pm 0.50}$	$13.01_{\pm 3.51}$	$421.40_{\pm 5.58}$	$210.93_{\pm 108.96}$	32.39	16.21
400^2	20	400	$5033_{\pm 729}$	$8242_{\pm 1245}$	$1.05_{\pm 0.21}$	$2.11_{\pm 0.46}$	$19.00_{\pm 5.68}$	$720.22_{\pm 12.00}$	$228.43_{\pm 114.59}$	37.91	12.02
200^2	10	400	$4679_{\pm 929}$	$7410_{\pm 1180}$	$0.88_{\pm 0.26}$	$1.71_{\pm 0.41}$	$8.15_{\pm 2.18}$	$207.46_{\pm 3.10}$	$217.90_{\pm 119.24}$	25.46	26.74
200^2	15	400	$4941_{\pm 587}$	$8384_{\pm 1056}$	$1.01_{\pm 0.17}$	$2.12_{\pm 0.38}$	$9.17_{\pm 2.30}$	$190.07_{\pm 3.51}$	$264.10_{\pm 127.40}$	20.73	28.80
200^2	20	400	$5691_{\pm 913}$	$9491_{\pm 1988}$	$1.34_{\pm 0.31}$	$2.67_{\pm 0.78}$	$10.37_{\pm 3.23}$	$168.10_{\pm 2.22}$	$272.75_{\pm 179.18}$	16.21	26.30
200^2	25	400	$7136_{\pm 1111}$	$10848_{\pm 1790}$	$2.53_{\pm 0.99}$	$3.88_{\pm 1.69}$	$10.81_{\pm 4.00}$	$154.57_{\pm 4.30}$	$232.48_{\pm 103.96}$	14.30	21.51
300^2	20	100	$925_{\pm 129}$	$1451_{\pm 259}$	$2.73_{\pm 0.54}$	$5.31_{\pm 1.54}$	$3.13_{\pm 0.60}$	$93.00_{\pm 0.76}$	$7.01_{\pm 2.27}$	29.71	2.24
300^2	20	200	$2474_{\pm 689}$	$3784_{\pm 842}$	$2.13_{\pm 0.87}$	$3.85_{\pm 1.26}$	$6.10_{\pm 2.28}$	$190.89_{\pm 2.09}$	$24.28_{\pm 13.01}$	31.29	3.98
300^2	20	300	$3843_{\pm 770}$	$6232_{\pm 1169}$	$1.48_{\pm 0.46}$	$2.90_{\pm 0.85}$	$8.63_{\pm 1.97}$	$293.85_{\pm 3.98}$	$89.19_{\pm 49.27}$	34.05	10.33
300^2	20	400	$5410_{\pm 945}$	$8807_{\pm 1343}$	$1.21_{\pm 0.30}$	$2.36_{\pm 0.52}$	$15.33_{\pm 3.71}$	$396.12_{\pm 8.08}$	$245.86_{\pm 109.93}$	25.84	16.04

Table 1: Experimental Results on Random Workspace

expanded with respect to Base-1, and (c) **Runtime**: the execution time. We also report the **Speedup** that OM achieves over the two baseline algorithms.

We run all the experiments in a desktop machine with Intel® Core™ i7-8700 CPU @ 3.20 GHz processor, 32 GB RAM, and Ubuntu 20.04 OS. We run each experiment for 20 times and report the mean and standard deviation for the evaluation metrics.

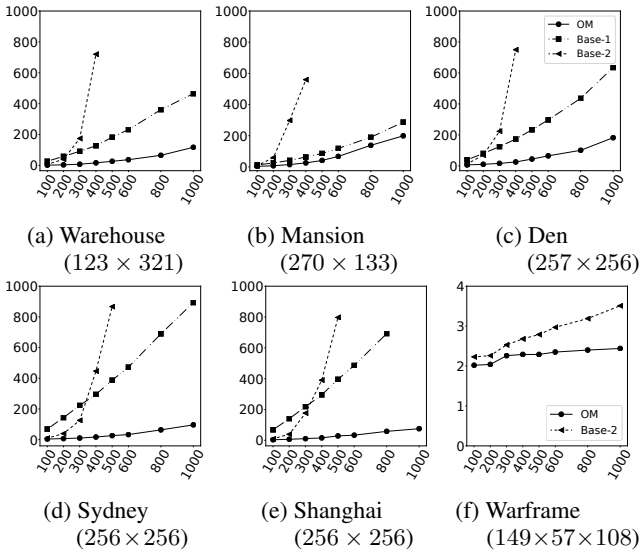
4.2 Experimental Results

Randomly Generated Workspaces. In Table 1, we report the results of experiments on randomly generated 2D workspaces having an equal number of robots and goals. In the first block of the table, we increase the workspace size while keeping R constant at 400. Owing to the demand-driven computation of A-costs by OM, 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. With an increase in the workspace size, the speedup with respect to Base-1 increases while it decreases with respect to Base-2. Obstacle density (OD) (i.e., the percentage of cells in a workspace that are occupied by obstacles) varies in the middle block. As OD increases, the speedup with respect to Base-1 declines whereas the speedup with respect to Base-2 does not vary significantly. The number of robots R varies in the last block. With an increase in R, first there is a growth in the speedup relative to Base-1, which later experiences a decline. The speedup with respect to Base-2 grows monotonically because the number of edges in the bipartite graph increases on increasing R, which inflates the efficiency of OM against Base-2 in the number of evaluations of A-cost (notice the increasing difference between NumExp of OM and Base-2).

Table 2 presents the results for cases having unequal number of robots and goals in random workspaces.

Standard Benchmark Workspaces. Through the plots in Figure 3, we compare the performance of OM with that of the baseline algorithms for standard workspaces available in the literature (Sturtevant 2012; Stern et al. 2019). The last plot is for a 3D workspace resembling an environment in Warframe game (Brewer and Sturtevant 2018). We take 15 min and 2 h as timeouts for the 2D and 3D workspaces, respectively. OM 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.

For the 2D workspaces, we observe that Base-1 outperforms Base-2 for high robot density (large number of robots in a small 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



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

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

Figure 3: Scalability plots of OM for Benchmark Workspace (*X-axis:* Number of Robots, *Y-axis:* Runtime(s) for (a-e); log Runtime(s) for (f))

WS Size	OD	R	G	NumExp		NumNodeExp (%)		Runtime (s)			Speedup	
				OM	Base 2	OM	Base 2	OM	Base 1	Base 2	Base 1	Base 2
200^2	20	150	200	$1336_{\pm 336}$	$1824_{\pm 300}$	$1.66_{\pm 0.90}$	$2.82_{\pm 1.99}$	$1.47_{\pm 0.57}$	$61.55_{\pm 1.35}$	$3.72_{\pm 1.38}$	41.87	2.53
		200	250	$1964_{\pm 520}$	$2741_{\pm 484}$	$2.25_{\pm 1.33}$	$3.53_{\pm 2.22}$	$2.55_{\pm 1.07}$	$82.03_{\pm 1.75}$	$8.01_{\pm 2.47}$	32.17	3.14
200^2	20	200	150	$1313_{\pm 245}$	$1824_{\pm 411}$	$1.26_{\pm 0.87}$	$1.94_{\pm 1.28}$	$1.40_{\pm 0.74}$	$81.83_{\pm 1.08}$	$3.51_{\pm 1.43}$	58.45	2.51
		250	200	$1838_{\pm 398}$	$2839_{\pm 663}$	$1.57_{\pm 1.09}$	$3.18_{\pm 2.98}$	$2.20_{\pm 1.15}$	$104.82_{\pm 2.01}$	$8.39_{\pm 3.89}$	47.65	3.81

Table 2: Results for $R \neq G$ cases in Random Workspace

WS Size	R	Makespan		GA Runtime (s)		PP Runtime (s)	Total Runtime (s)		Overall Speedup
		Before PP	After PP	OM	Base 2		OM	Base 2	
Warehouse 123×321	100	$64.54_{\pm 14.51}$	$64.54_{\pm 14.51}$	$1.91_{\pm 0.55}$	$7.20_{\pm 3.93}$	$0.05_{\pm 0.01}$	$1.96_{\pm 0.55}$	$7.25_{\pm 3.98}$	3.70
	200	$50.10_{\pm 13.44}$	$50.10_{\pm 13.44}$	$3.85_{\pm 0.89}$	$39.71_{\pm 18.49}$	$0.16_{\pm 0.06}$	$4.01_{\pm 0.89}$	$39.87_{\pm 18.61}$	9.94
	400	$39.32_{\pm 8.52}$	$39.70_{\pm 8.89}$	$16.82_{\pm 5.82}$	$720.37_{\pm 465.80}$	$0.41_{\pm 0.11}$	$17.23_{\pm 5.82}$	$720.78_{\pm 466.72}$	41.83
Mansion 270×133	100	$50.74_{\pm 10.25}$	$51.14_{\pm 10.31}$	$3.44_{\pm 0.48}$	$7.66_{\pm 1.63}$	$0.04_{\pm 0.01}$	$3.48_{\pm 0.48}$	$7.70_{\pm 1.63}$	2.21
	200	$41.22_{\pm 7.90}$	$42.48_{\pm 8.09}$	$7.21_{\pm 1.58}$	$59.75_{\pm 27.90}$	$0.10_{\pm 0.02}$	$7.31_{\pm 1.58}$	$59.85_{\pm 28.02}$	8.19
	400	$34.76_{\pm 5.43}$	$38.86_{\pm 6.90}$	$25.65_{\pm 6.80}$	$559.56_{\pm 216.52}$	$0.27_{\pm 0.05}$	$25.92_{\pm 6.81}$	$559.83_{\pm 216.86}$	21.60
Den 257×256	100	$74.74_{\pm 13.57}$	$75.08_{\pm 13.56}$	$6.10_{\pm 2.06}$	$12.95_{\pm 4.18}$	$0.06_{\pm 0.02}$	$6.16_{\pm 2.06}$	$13.01_{\pm 4.22}$	2.11
	200	$60.62_{\pm 13.84}$	$61.02_{\pm 14.08}$	$10.04_{\pm 4.50}$	$68.55_{\pm 49.01}$	$0.16_{\pm 0.06}$	$10.20_{\pm 4.50}$	$68.71_{\pm 49.24}$	6.74
	400	$47.56_{\pm 7.95}$	$48.80_{\pm 8.64}$	$25.19_{\pm 6.94}$	$750.02_{\pm 439.98}$	$0.48_{\pm 0.13}$	$25.67_{\pm 6.94}$	$750.50_{\pm 441.02}$	29.24
Sydney 256×256	100	$73.14_{\pm 9.31}$	$73.24_{\pm 9.28}$	$3.66_{\pm 0.83}$	$9.10_{\pm 3.16}$	$0.05_{\pm 0.01}$	$3.71_{\pm 0.82}$	$9.15_{\pm 3.17}$	2.47
	200	$50.34_{\pm 6.31}$	$50.58_{\pm 6.59}$	$6.80_{\pm 1.45}$	$39.29_{\pm 18.88}$	$0.16_{\pm 0.03}$	$6.96_{\pm 1.45}$	$39.45_{\pm 18.94}$	5.67
	400	$43.78_{\pm 6.82}$	$44.16_{\pm 6.98}$	$16.77_{\pm 3.58}$	$446.55_{\pm 336.09}$	$0.44_{\pm 0.07}$	$17.21_{\pm 3.58}$	$446.99_{\pm 336.64}$	25.97

Table 3: Comparison with TSWAP

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.

4.3 Comparison with TSWAP

To illustrate that our goal assignment algorithm OM can help improve the state-of-the-art algorithms for the AMAPF problem, we consider the offline TSWAP (Okumura and Défago 2022) 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 2 in (Okumura and Défago 2022), which we refer to as Base-2) by OM. 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 3. 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 of the total runtime of the AMAPF solution process, and thus, plugging OM into the algorithm for AMAPF provides significant speedup.

4.4 Comparison with CBM

The Conflict-Based Min-Cost-Flow (CBM) algorithm (Ma and Koenig 2016) provides optimal goal assignment having collision-free paths. However, it is not scalable for large workspaces having large number of robots. For example, CBM takes approximately $6500\,\mathrm{s}$ to solve a problem instance comprising of a 100×100 random workspace and 100 robots (with 4 motion primitives). Table 3 shows that OM along with the collision-free path planning module of TSWAP takes less than $30\,\mathrm{s}$ on average for 400 robots (with 8 motion primitives) in the benchmark 2D workspaces, which is beyond the capability of CBM.

5 Conclusion

We have presented a scalable centralized algorithm to solve the optimal-makespan goal assignment problem for multi-robot systems. Our method outperforms the state-of-the-art methods for the same problem considerably in both 2D and 3D environments. 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. It is straightforward to extend our algorithm to find an assignment having optimal total cost among the assignments having an optimal makespan by plugging the algorithm from (Aakash and Saha 2022) after minor modifications.

Acknowledgements

We thank Hang Ma for making the source code of the CBM implementation available for our experiments.

References

Aakash; and Saha, I. 2022. It Costs to Get Costs! A Heuristic-Based Scalable Goal Assignment Algorithm for Multi-Robot Systems. In *ICAPS*, 2–10. AAAI Press.

Alonso-Mora, J.; Breitenmoser, A.; Rufli, M.; Beardsley, P. A.; and Siegwart, R. 2010. Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots. In *DARS 2010, Lausanne, Switzerland*, 203–216.

Bondy, J. A.; and Murty, U. S. R. 1976. *Graph Theory with Applications*, volume 290. Macmillan London.

Brewer, D.; and Sturtevant, N. R. 2018. Benchmarks for Pathfinding in 3D Voxel Space. *SoCS*.

Burkard, R.; Dell'Amico, M.; and Martello, S. 2009. *Assignment Problems*. USA: Society for Industrial and Applied Mathematics. ISBN 0898716632.

Chen, Y. F.; Liu, M.; Everett, M.; and How, J. P. 2017. Decentralized Non-Communicating Multi-Agent Collision Avoidance with Deep Reinforcement Learning. In *ICRA*, 285–292.

Das, S. N.; Nath, S.; and Saha, I. 2021. OMCoRP: An Online Mechanism for Competitive Robot Prioritization. In *ICAPS*, 112–121.

Dijkstra, E. W. 1959. A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1): 269–271.

Fulkerson, D. R.; Glicksberg, I. L.; and Gross, O. A. 1953. *A Production-Line Assignment Problem*. Santa Monica, California: The Rand Corporation.

Gonzalez-de-Santos, P.; Ribeiro, A.; Fernandez-Quintanilla, C.; Lopez-Granados, F.; Brandstoetter, M.; Tomic, S.; Pedrazzi, S.; Peruzzi, A.; Pajares, G.; Kaplanis, G.; Perez-Ruiz, M.; Valero, C.; del Cerro, J.; Vieri, M.; Rabatel, G.; and Debilde, B. 2017. Fleets of Robots for Environmentally-Safe Pest Control in Agriculture. *Precision Agriculture*, 18: 574–614.

Grippa, P.; Behrens, D. A.; Wall, F.; and Bettstetter, C. 2019. Drone Delivery Systems: Job Assignment and Dimensioning. *Auton. Robots*, 43(2): 261–274.

Gross, O. 1959. The Bottleneck Assignment Problem. Technical Report P-1620, The Rand Corporation, Santa Monica, California.

Hennes, D.; Claes, D.; Meeussen, W.; and Tuyls, K. 2012. Multi-Robot Collision Avoidance with Localization Uncertainty. In *AAMAS*, 147–154.

Leiserson, C. E.; Rivest, R. L.; Cormen, T. H.; and Stein, C. 1994. *Introduction to Algorithms*, volume 3. MIT press Cambridge, MA, USA.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *AAAI*, 11272–11281.

Ma, H.; and Koenig, S. 2016. Optimal Target Assignment and Path Finding for Teams of Agents. In *AAMAS*, 1144–1152.

MacAlpine, P.; Price, E.; and Stone, P. 2015. SCRAM: Scalable Collision-Avoiding Role Assignment with Minimal-Makespan for Formational Positioning. In *AAAI*.

Okumura, K.; and Défago, X. 2022. Solving Simultaneous Target Assignment and Path Planning Efficiently with Time-Independent Execution. In *ICAPS*, 270–278.

Silver, D. 2005. Cooperative Pathfinding. *Aiide*, 1: 117–122.

Snape, J.; Van Den Berg, J.; Guy, S. J.; and Manocha, D. 2010. Smooth and Collision-Free Navigation for Multiple Robots under Differential-Drive Constraints. In *IROS*, 4584–4589.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *SoCS*, 151–158.

Sturtevant, N. 2012. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2): 144 – 148.

Tian, Y.-T.; Yang, M.; Qi, X.-Y.; and Yang, Y.-M. 2009. Multi-Robot Task Allocation for Fire-Disaster Response Based on Reinforcement Learning. In *2009 International Conference on Machine Learning and Cybernetics*, volume 4, 2312–2317.

Turpin, M.; Michael, N.; and Kumar, V. 2013. Concurrent Assignment and Planning of Trajectories for Large Teams of Interchangeable Robots. In *ICRA*, 842–848.

Turpin, M.; Mohta, K.; Michael, N.; and Kumar, V. 2013. Goal Assignment and Trajectory Planning for Large Teams of Aerial Robots. In *RSS*.

Turpin, M.; Mohta, K.; Michael, N.; and Kumar, V. 2014. Goal Assignment and Trajectory Planning for Large Teams of Interchangeable Robots. *Auton. Robots*, 37(4): 401–415.

van den Berg, J. P.; Snape, J.; Guy, S. J.; and Manocha, D. 2011. Reciprocal Collision Avoidance with Acceleration-Velocity Obstacles. In *ICRA*, 3475–3482.

Yu, J.; and LaValle, S. M. 2013. Multi-Agent Path Planning and Network Flow. In *Algorithmic foundations of robotics X*, 157–173. Springer.