Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

It Costs to Get Costs!
A Heuristic-Based Scalable Goal Assignment Algorithm for Multi-Robot Systems

Aakash and Indranil Saha

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

Abstract

The goal assignment problem for a multi-robot application involves assigning a unique goal to each robot to minimize the total cost of movement for all the robots. A significant step in the state-of-the-art algorithms solving this problem is to find the cost associated with each robot-goal pair. For a large multi-robot system with many robots and many goals in a complex workspace, the computation time required to find the paths for all robot-goal pairs may become prohibitively large. We present an algorithm that solves the optimal goal assignment problem without computing the paths between all the robot-goal pairs. Instead, our algorithm computes the obstacle-free optimal path for any robot-goal pair in a demand-driven way, i.e., if it is absolutely required to ensure the optimality of the goal assignment. We evaluate our algorithm extensively on both randomly generated and standard workspaces for hundreds of robots. Our experimental results demonstrate that the proposed algorithm achieves an order-of-magnitude speedup over the state-of-the-art baseline algorithm. To the best of our knowledge, our algorithm is the first one to solve the multi-robot optimal goal assignment problem without computing the paths for all robot-goal pairs explicitly, guaranteeing high scalability.

1 Introduction

Several multi-robot applications such as warehouse management (Li et al. 2021; Chen 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 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 so that the total cost of movements by the robots to their designated goal locations is minimized. The assignment problem with the above-mentioned objective is known as *linear sum assignment problem* (Burkard and Cela 1999).

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. Once these costs are known, the well-known primal-dual Hungarian algorithm (Kuhn 1955; Munkres 1957) can be applied to find the optimal assignment minimizing the total cost. Several multi-robot goal assignment and path planning algorithms (Turpin et al. 2013; Madridano et al. 2019; Turpin, Michael, and Kumar 2013; Turpin et al. 2014; Hönig et al. 2018) do compute the costs *for all* robot-goal pairs as an integral part of the algorithm. An efficient way of computing these costs is to employ *Dijkstra's shortest path algorithm* (Dijkstra 1959) for each robot to find the shortest paths from its initial location to all the goal locations. However, this method scales poorly with the size of the workspace and the number of robots and goals. Thus, this step acts as a hindrance against solving the goal assignment problem for a large-scale multi-robot system having hundreds of robots and goals in a centralized manner.

To circumvent the computational challenge of the centralized approach, several researchers have developed decentralized solutions to the multi-robot goal assignment problem (Choi, Brunet, and How 2009; Giordani, Lujak, and Martinelli 2010, 2013; Liu and Shell 2012; Chopra et al. 2017). 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 number 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.

In this paper, we propose a *scalable centralized* algorithm for solving the multi-robot optimal goal assignment problem. Our algorithm is based on the bipartite graph-based implementation of the primal-dual Hungarian algorithm that is widely used to solve the assignment problem. 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 a heuristic cost which is guaranteed to be a lower bound of the cost associated with the shortest obstacle-free path for a robot-goal pair. We compute the obstacle-free optimal path for any robot-goal pair in a demand-driven way, i.e., if it is absolutely required to ensure the optimality of the goal assignment. This strategy enables us to find the optimal goal assignment without computing the obstacle-free path for a significant number of robot-goal pairs, leading to a high degree of scalability.

The problem addressed in this paper is different from the conventional Multi-Agent Path Finding (MAPF) problem (Stern et al. 2019) which assumes that the goal for each robot is given and the objective is to find the collision-free paths for the robots. Recently proposed CBS-TA (Hönig et al. 2018) attempts to solve the task assignment problem together with optimal collision-free path finding, but such an algorithm scales poorly with the number of robots. The objective of our work is to have a solution to the optimal goal assignment problem for a large multi-robot system with hundreds of robots within a short duration such that the robots can embark on their individual optimal paths towards their respective goals as early as possible. Eliminating collisions statically for such large multi-robot systems is computationally infeasible. Rather, we assume that the robots have a local collision avoidance mechanism using which they can avoid collisions with other robots and any dynamic obstacles they come across during their movement (Alonso-Mora et al. 2010; Snape et al. 2010; van den Berg et al. 2011; Hennes et al. 2012; Chen et al. 2017; Long et al. 2018).

We implement our algorithm in Python and evaluate it through thorough experimentation. We evaluate our algorithm on randomly generated 2D workspaces and benchmark 2D and 3D workspaces available in the literature. As the baseline, we consider a Hungarian method-based goal assignment algorithm where the costs for all robot-goal pairs are computed, as is done in several prior work (e.g. (Turpin et al. 2013, 2014)). Our experimental results show that the proposed algorithm outperforms the baseline algorithm by an order of magnitude for most problem instances.

The rest of the paper is organized as follows. In Section 2, we introduce the problem formally and illustrate it with an example. In Section 3, we present our algorithm and its complexity and soundness. We present the results of the evaluation of our algorithm in detail in Section 4. Finally, we conclude in Section 5 with a brief outline of a potential future research direction.

2 Problem

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

A *workspace* is a 2D or a 3D space which is divided by grid lines into square-shaped or cube-shaped cells, respectively. Each cell can be addressed using its coordinates. Some cells may be occupied by obstacles. A typical multi-robot application would require the robots to complete several tasks in a workspace. To complete those tasks, the robots need to reach the locations where the tasks can be completed. These locations are called the *goal locations*. Each goal (i.e., task) can be assigned to at most one robot, while each robot can also be assigned to at most one goal. We assume that an assignment is feasible for each robot and goal, i.e., no robot (goal) is trapped within some obstacles in a way that it cannot be connected with any goal (robot). Our aim is to find which goal should be assigned to which robot so that the total cost of movement of all the robots due to the resultant assignment is minimized.

2.1 Problem Definition

We define the problem formally in this subsection.

Problem 1. *Consider a multi-robot application in a grid-based workspace, where the set S of start locations of 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.*

Let the Boolean variable x_{ij} represent the assignment of robot $i \in [R]$ to goal $j \in [G]$. The variable x_{ij} takes the value 1 if robot i is assigned to goal j, otherwise it is 0. Each robot can be assigned to at most one goal, and each goal can be served by at most one robot. Let c_{ij} denote the cost of movement between $s_i \in S$ and $f_j \in F$, where $i \in [R]$ and $j \in [G]$. Find the robot-goal assignments for the multi-robot application such that the total cost of movements of all the robots due to the resultant assignment (without considering the overhead for collision avoidance) is minimized.

Mathematically, the problem can be written as:

$$\texttt{minimize} \quad \sum_{i=1}^{R}\sum_{j=1}^{G} c_{ij}x_{ij},$$

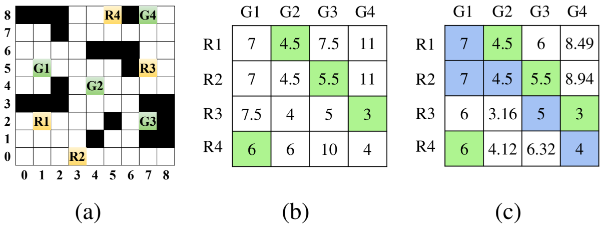
subject to

$$\begin{aligned}
&R < G : \forall\, i \in [R],\ \sum_{j=1}^{G} x_{ij} = 1; \forall j \in [G],\ \sum_{i=1}^{R} x_{ij} \leq 1;\\
&R > G : \forall\, i \in [R],\ \sum_{j=1}^{G} x_{ij} \leq 1; \forall j \in [G],\ \sum_{i=1}^{R} x_{ij} = 1;\\
&R = G : \forall\, i \in [R],\ \sum_{j=1}^{G} x_{ij} = 1; \forall j \in [G],\ \sum_{i=1}^{R} x_{ij} = 1;\\
&x_{ij} \in \{0, 1\}. \qquad (1)
\end{aligned}$$

The cost information needs to be known to solve the above problem. Finding such information incurs a *cost* too in terms of the computation time. It is because, in order to discover the cost of movement between two cells in a workspace, the shortest obstacle-free path between them has to be computed. Let us denote such cost by the phrase *actual movement cost*. Computing such costs for all the robot-goal pairs requires solving several complex path planning problems, which chokes the scalability. Our goal is to design an algorithm that computes these costs judiciously for only the necessary robot-goal pairs while finding a solution to the goal assignment problem.

2.2 Example

Consider the 9×9 workspace shown in Figure 1(a). The black-colored cells denote the obstacles. It has four robots



(a)

	G1	G2	G3	G4
R1	7	🟩 4.5	7.5	11
R2	7	4.5	🟩 5.5	11
R3	7.5	4	5	🟩 3
R4	🟩 6	6	10	4

(b)

	G1	G2	G3	G4
R1	🟦 7	🟩 4.5	6	8.49
R2	🟦 7	🟦 4.5	🟩 5.5	8.94
R3	6	3.16	🟦 5	🟩 3
R4	🟩 6	4.12	6.32	🟦 4

(c)

Figure 1: An example problem

(R1, R2, R3, and R4) and four goals (G1, G2, G3, and G4). In such a 2D workspace, we consider 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.

Figure 1(b) shows the cost matrix having the actual movement cost for each robot-goal pair. The optimal goal assignment is depicted by the green cells. Now, can we have the same optimal goal assignment, without computing all the actual movement costs? Figure 1(c) shows the transformed cost matrix, which is an outcome of our approach. Here, only the colored cells (blue and green) have the actual movement costs. Note that we obtain the same optimal assignment (shown in green) without computing all the actual movement costs. In the next section, we present our algorithm to achieve this goal.

3 Algorithm

In this section, we present the details of our algorithmic solution. Its distinguishing feature is that it avoids computing the actual movement cost for many robot-goal pairs. It frames a strategy that judiciously computes only the necessary costs without compromising the optimality. We use the primal-dual method (Ford and Fulkerson 1956) to solve Problem 1, since it has been shown in (Carpaneto, Martello, and Toth 1984) and (McGinnis 1983) that such methods are comparatively more efficient than the primal ones. For this, we first translate Problem 1 into its graph-theoretic form (Lawler 1976) which we call the primal problem. We then consider the corresponding dual problem. Our algorithm solves both the primal and the dual problems simultaneously. It maintains a feasible dual solution while trying to construct a primal solution that satisfies complementary slackness (with the dual solution) (Papadimitriou and Steiglitz 1982).

3.1 Graph Theoretic Formulation

Problem 1 can be formally expressed in graph theoretic form as follows.

Problem 2. *Given a bipartite graph $\mathcal{G} = ([R], [G], E)$ with bipartition $([R], [G])$ and a cost function $\mathtt{cost} : E \rightarrow \mathbb{R}$, find a maximum cardinality matching M of minimum cost, where cost of M is given by:*

$$\mathtt{cost}(M) = \sum_{(u,v)\in M} \mathtt{cost}(u, v).$$

Problem 2 is also known as the *Minimum cost bipartite matching problem*. Its dual uses the notion of feasible vertex labeling, which is presented below.

Feasible vertex labeling: Given a bipartite graph $\mathcal{G} = ([R], [G], E)$ with bipartition $([R], [G])$, a cost function $\mathtt{cost} : E \rightarrow \mathbb{R}$, 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 the vertices such that:

$$\mathtt{cost}(u, v) \geq Z_R(u) + Z_G(v) \qquad \forall (u, v) \in E.$$

We now define the dual of Problem 2.

Problem 3. *Given a bipartite graph $\mathcal{G} = ([R], [G], E)$ with bipartition $([R], [G])$, a cost function $\mathtt{cost} : E \rightarrow \mathbb{R}$, and the vertex labeling functions $Z_R : [R] \rightarrow \mathbb{R}$ and $Z_G : [G] \rightarrow \mathbb{R}$, find a feasible vertex labeling of maximum cost, where the cost of labeling is given by:*

$$\mathtt{cost}(Z_R, Z_G) = \sum_{u\in[R]} Z_R(u) + \sum_{v\in[G]} Z_G(v).$$

We describe the algorithm in the next subsection. Note that we use the particular case wherein $R \leq G$ for the ease of exposition. However, with minor changes, the proposed solution also holds for the case where $G < R$.

3.2 Algorithm Description

We present our goal assignment algorithm formally in Algorithm 1. It uses the Euclidean distance (i.e., the aerial distance disregarding any obstacles) between a robot's start location and a goal location as the *heuristic cost*, or *H-cost* for short. In the rest of the paper, we shall refer to the actual optimal movement cost (after taking the obstacles into account) between the same locations as the *actual cost*, or *A-cost* for short. Note that between any two locations, an H-cost is never greater than the corresponding A-cost.

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* (RRA*) (Silver 2005). RRA* is designed for the scenario where multiple robots have a common goal. It commences by searching the path in the reverse direction, i.e., from the common goal to a particular robot. While it does so, it uses the conventional open-list (OL) and closed-list (CL) to keep track of nodes processed during the graph search, just as in the A* search algorithm (Hart, Nilsson, and Raphael 1968). For the exploration of subsequent paths from the same goal, RRA* reuses the nodes already present in OL and CL to resume the search instead of starting from scratch. However, our algorithm, for a particular robot, may explore optimal obstacle-free paths (and corresponding A-costs) to more than one goals. Thus, we adapt the concept of RRA* and develop FRA* in which the search executes in forward direction from a common robot (rather than a common goal). Once the goal (to which the path needs to be computed) changes for a particular robot, the heuristic cost for all nodes in OL is evaluated for the new goal. The search then resumes from the node having

```
Algorithm 1: Goal Assignment for Multi-Robot Systems

  Global: $C, T, P, nexp, OL, CL$
1  procedure solve_goal_assignment ($WS$, $S$, $F$)
2      $R = |S|,\ \ G = |F|$
3      for $i = 1$ to $R$ do
4          for $j = 1$ to $G$ do
5              $C(i)(j) =$ get_Euclidean_distance$(S(i), F(j))$
6              $T(i)(j) =$ '$h$',  $P(i)(j) = [\,]$
7          end
8      end
9      $acost_{min} =$ explore_min_actual_cost$(WS, S, F)$
10     $\forall\, i \in [R] : Z_R(i) = acost_{min}(i)$
11     $\forall\, j \in [G] : Z_G(j) = 0$
12     $\mathcal{G}_{eq} =$ get_equality_subgraph$(Z_R, Z_G)$
13     $M =$ get_initial_match$(\mathcal{G}_{eq})$
14     $M =$ maximize_match$(\mathcal{G}_{eq}, M)$
15     while $M$ is not a total matching do
16         $\langle R_c, G_c\rangle =$ get_min_vertex_cover$(\mathcal{G}_{eq}, M)$
17         $\Delta =$ collect_slacks$(R, G, R_c, G_c, Z_R, Z_G)$
18         $\delta_{min} =$ get_min_slack$(WS, S, F, Z_R, Z_G, \Delta)$
19         uncover_actual_cost$(WS, S, F, R_c, G_c, Z_R, Z_G,$
             $\delta_{min})$
20         $\forall\, i \in R_c \qquad : Z_R(i) = Z_R(i) - \delta_{min}$
21         $\forall\, j \in [G] \setminus G_c : Z_G(j) = Z_G(j) + \delta_{min}$
22         $\mathcal{G}_{eq} =$ get_equality_subgraph$(Z_R, Z_G)$
23         $M =$ maximize_match$(\mathcal{G}_{eq}, M)$
24     end
25     return $M$
26 end
27 procedure explore_min_actual_cost $(WS, S, F)$
28     $R = |S|,\ \ G = |F|$
29     for $i = 1$ to $R$ do
30         $acost_{min}(i) = \infty$
31         $OL(i) = [\,],\ \ CL(i) = [\,]$
32         $\langle hcost_{min}, hindex\rangle =$ find_min_cost$(i, G,$ '$h$'$)$
33         while ($(hindex \neq -1)$ & $(hcost_{min} \leq acost_{min}(i))$)
             do
34             $\langle c, p, OL(i), CL(i)\rangle =$
                 FRAStar$(WS, S(i), F(hindex), OL(i), CL(i))$
35             $C(i)(hindex) = c,\ \ P(i)(hindex) = p$
36             $T(i)(hindex) =$ '$a$'
37             $nexp = nexp + 1$
38             $\langle hcost_{min}, hindex\rangle =$ find_min_cost$(i, G,$ '$h$'$)$
39             $\langle acost_{min}(i), -\rangle =$ find_min_cost$(i, G,$ '$a$'$)$
40         end
41     end
42     return $acost_{min}$
43 end
44 procedure collect_slacks ($R$, $G$, $R_c$, $G_c$, $Z_R$, $Z_G$)
45     for each uncovered robot $i \in [R] \setminus R_c$ do
46         for each uncovered goal $j \in [G] \setminus G_c$ do
47             $\delta = C(i)(j) - (Z_R(i) + Z_G(j))$
48             $\Delta.add(\langle i, j, \delta, T(i)(j)\rangle)$
49         end
50     end
51     return $\Delta$
52 end
53 procedure get_min_slack ($WS$, $S$, $F$, $Z_R$, $Z_G$, $\Delta$)
54     while True do
55         $\langle \Delta_{min}, index\rangle =$ find_min_delta$(\Delta)$
56         if $\Delta_{min}.t ==$ '$a$' then
57             return $\Delta_{min}.\delta$
58         else
59             $\langle c, p, OL(\Delta_{min}.i), CL(\Delta_{min}.i)\rangle =$
               FRAStar$(WS, S(\Delta_{min}.i), F(\Delta_{min}.j]),$
               $OL(\Delta_{min}.i), CL(\Delta_{min}.i))$
60             $C(\Delta_{min}.i)(\Delta_{min}.j) = c$
61             $P(\Delta_{min}.i)(\Delta_{min}.j) = p$
62             $T(\Delta_{min}.i)(\Delta_{min}.j) =$ '$a$'
63             $nexp = nexp + 1$
64             $\delta_{new} = C(\Delta_{min}.i)(\Delta_{min}.j) - (Z_R(i){+}Z_G(j))$
65             Update: $\Delta(index) = \langle i, j, \delta_{new},$ '$a$'$\rangle$
66         end
67     end
68 end
69 procedure uncover_actual_cost ($WS$, $S$, $F$, $R_c$, $G_c$, $Z_R$,
70 $Z_G$, $\delta_{min}$)
71     $R = |S|,\ \ G = |F|$
72     for each uncovered robot $i \in [R] \setminus R_c$ do
73         for each uncovered goal $j \in [G] \setminus G_c$ do
74             $\delta = C(i)(j) - (Z_R(i) + Z_G(j))$
75             if $T(i)(j) ==$ '$h$' and $\delta == \delta_{min}$ then
76                 $\langle c, p, OL(i), CL(i)\rangle =$
                     FRAStar$(WS, S(i), F(j), OL(i), CL(i))$
77                 $C(i)(j) = c,\ \ P(i)(j) = p$
78                 $T(i)(j) =$ '$a$'
79                 $nexp = nexp + 1$
80             end
81         end
82     end
83 end
```

the least f-value in OL, thereby helping greatly in expediting the search process. In Algorithm 1, we use FRAStar as a library function implementing the FRA* algorithm, which takes the workspace, robot's start location, goal location, OL and CL (for a robot) as inputs. It provides the optimal obstacle-free path, corresponding A-cost, and updated OL and CL as outputs. The updated OL and CL are reused in subsequent path searches.

We now describe Algorithm 1 in detail. The first procedure solve_goal_assignment captures the *main* module that invokes other procedures to solve the multi-robot goal assignment problem. It takes the workspace WS, the set S having the start locations of the robots, and the set F having the goals locations as inputs. It begins by computing H-cost for each robot-goal pair and stores it 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 (line 6). The paths are initialized in the 2D matrix P (line 6).

The procedure `explore_min_actual_cost` discovers the minimum A-cost $acost_{min}(i)$ for each robot $i \in [R]$ without computing all its A-costs *naively*. It does so in the following way: At the outset, a particular robot i has an H-cost for each of the goals. So, the procedure searches for its minimum H-cost $hcost_{min}$ and replaces it with the corresponding A-cost. This step is repeated until robot i's current minimum H-cost exceeds its current minimum A-cost (line 33). It is because in that case, there is no need to explore the H-costs further since they are under-approximation of A-costs, and if at all explored, will they emerge into higher A-costs only. `FRAStar` function is used to compute the A-cost (line 34). Whenever an H-cost is replaced by its corresponding A-cost, we also do the following: (i) save the path information (line 35), (ii) update the cost-type in matrix T (line 36) and (iii) keep track of the number of path explorations in the variable $nexp$ (line 37). For a robot, an auxiliary procedure `find_min_cost` fetches the minimum cost of a particular cost-type (heuristic or actual).

Once the minimum A-cost is available for each robot, the process to formulate an equality subgraph $\mathcal{G}_{eq}$ begins. Usually, an *equality subgraph* $\mathcal{G}_{eq} = ([R], [G], E_{Z_R,Z_G})$ is a subgraph of a bipartite graph $\mathcal{G} = ([R], [G], E)$, such that its edge set E_{Z_R,Z_G} is given by:

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

As Algorithm 1 computes the A-costs between the robot-goal pairs in a demand-driven way, it does not have the 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 $\mathcal{G}_{eq}$. Symbolically, the robots and the goals form the bipartition of the vertex set of $\mathcal{G}_{eq}$ (see Figure 2(c) for an example). An initial dual solution in the form of an initial feasible labeling of the vertices is obtained wherein the initial label for the robots is their corresponding minimum A-cost, while the labels for the goals are set as zero (lines 10-11). Edges of $\mathcal{G}_{eq}$ are obtained using the labels (line 12). Now, as the construction of $\mathcal{G}_{eq}$ is complete, a primal solution in the form of maximum cardinality matching M is obtained in it (lines 13-14). 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 (i.e., if the primal solution is feasible), the process terminates, and the following are reported as output: (i) M: the final robot-goal assignments, (ii) $C(i)(j)$ where $(i, j) \in M$: cost of final robot-goal assignments, (iii) $P(i)(j)$ where $(i, j) \in M$: path for final robot-goal assignments, and (iv) $nexp$: the number of path explorations. However, in case the current M leaves one or more robots unassigned, additional steps are required to update M until it has an assignment for each robot.

The additional steps begin by finding the minimum vertex cover $\langle R_c, G_c \rangle$ corresponding to M (line 16), which enables the identification of covered robots and covered goals. The procedure `collect_slacks` is invoked to collect the *slack* δ of edge between each pair of uncovered robot and uncovered goal (line 17). δ for an edge (u, v) is computed as:

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

A collection data structure Δ is used to store the slack along with other information. It consists of tuples, where each tuple has the following information: (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. Individual elements in a tuple are accessed using the dot operator.

Procedure `get_min_slack` finds the minimum slack δ_{min} (from Δ) that is computed on A-cost (line 18). The equality subgraph should not have edges with H-cost (as matching should be computed on A-cost edges only). So only when the update of labels (lines 20-21) happens with δ_{min} computed on A-cost, a new edge with A-cost is certain to appear in the subsequently revised equality subgraph (line 22). So, if δ_{min} was computed on A-cost, then it is used to update the labels. However, if δ_{min} was computed on H-cost, then the corresponding A-cost is explored for the concerned robot-goal pair (lines 58-66) and the slack is updated using this A-cost. Furthermore, Δ is updated with the revised slack before the process is repeated to search for δ_{min}. Note that this procedure invokes `find_min_delta` procedure (line 55), which searches for 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.

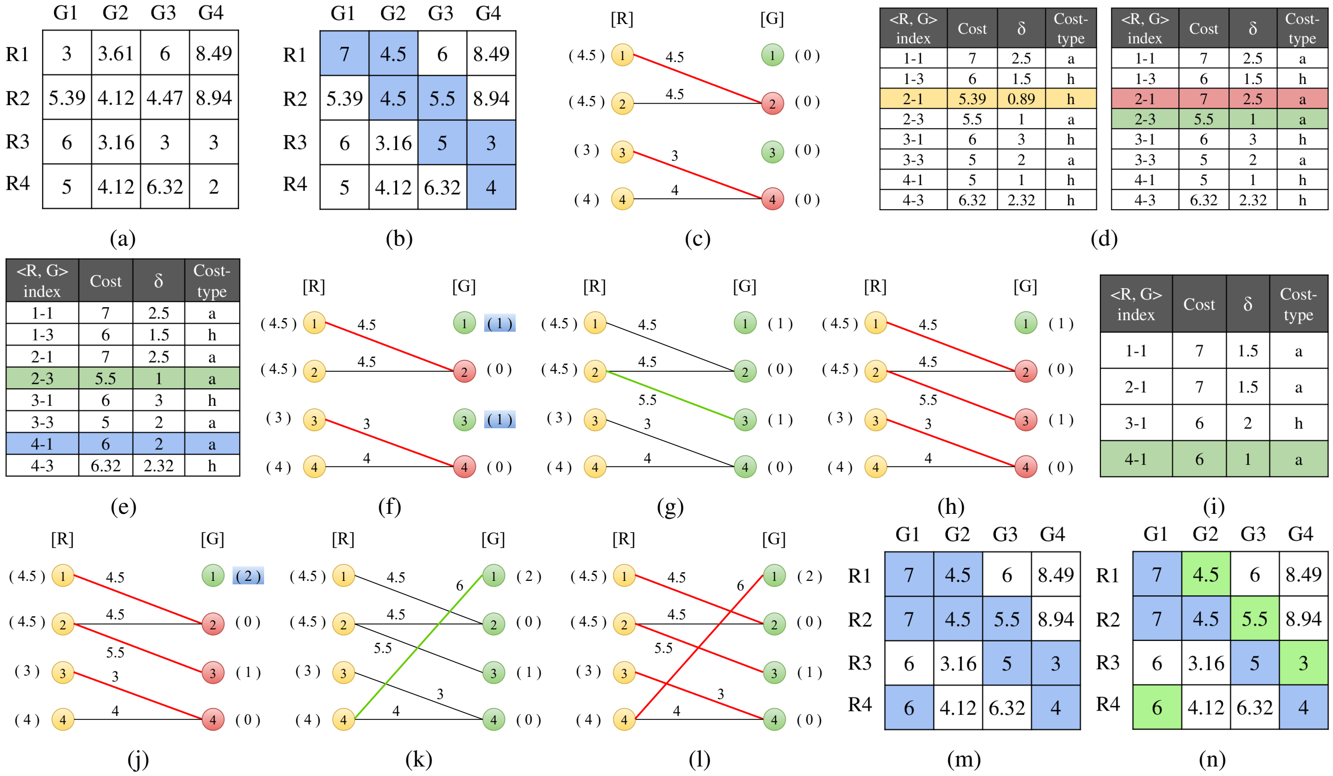
For the uncovered robot - uncovered goal pairs whose δ are computed on H-costs and are equal to δ_{min}, the `uncover_actual_cost` procedure replaces their H-costs by corresponding A-costs (line 19). This is done to ensure that the explored A-costs whose δ are still equal to δ_{min} can participate in the subsequently revised $\mathcal{G}_{eq}$ (line 22).

The dual solution is modified as follows. The labels of covered robots are decreased by δ_{min} (line 20) whereas the labels of uncovered goals are increased by δ_{min} (line 21). This label-update rule ensures that a new edge in $\mathcal{G}_{eq}$ would appear only between an uncovered robot and an uncovered goal. $\mathcal{G}_{eq}$ is revised using the updated labels (line 22) and an attempt is made to maximize the current M in $\mathcal{G}_{eq}$ (line 23). The process loops until M becomes a total matching.

3.3 Example

Figure 2 presents an illustration of the step-by-step execution of Algorithm 1 on the multi-robot goal assignment problem introduced in Figure 1(a). Figure 2(a) shows the cost matrix having the H-costs for each robot-goal pair. Figure 2(b) shows the transformed cost matrix after the exploration of minimum A-cost for each robot. Blue-colored cells have A-costs, while the uncolored cells have H-costs.

Note the presence of two blue cells for robot R1. Initially, $hcost_{min}$ for R1 is 3 (for G1). With an aim to discover $acost_{min}$ for R1, we use the FRA* search algorithm to explore the A-cost corresponding to $hcost_{min}$, which yields 7 (see R1-G1 cell in Figure 2(b)). The new $hcost_{min}$ is 3.61 (for G2), which is less than the current $acost_{min}$ (7). So, there is a possibility that the A-cost corresponding to this new $hcost_{min}$ can be less than the current $acost_{min}$.



	G1	G2	G3	G4
R1	3	3.61	6	8.49
R2	5.39	4.12	4.47	8.94
R3	6	3.16	3	3
R4	5	4.12	6.32	2

(a)

	G1	G2	G3	G4
R1	7	4.5	6	8.49
R2	5.39	4.5	5.5	8.94
R3	6	3.16	5	3
R4	5	4.12	6.32	4

(b)

(c)

<R, G> index	Cost	δ	Cost-type
1-1	7	2.5	a
1-3	6	1.5	h
2-1	5.39	0.89	h
2-3	5.5	1	a
3-1	6	3	h
3-3	5	2	a
4-1	5	1	h
4-3	6.32	2.32	h

<R, G> index	Cost	δ	Cost-type
1-1	7	2.5	a
1-3	6	1.5	h
2-1	7	2.5	a
2-3	5.5	1	a
3-1	6	3	h
3-3	5	2	a
4-1	5	1	h
4-3	6.32	2.32	h

(d)

<R, G> index	Cost	δ	Cost-type
1-1	7	2.5	a
1-3	6	1.5	h
2-1	7	2.5	a
2-3	5.5	1	a
3-1	6	3	h
3-3	5	2	a
4-1	6	2	a
4-3	6.32	2.32	h

(e)

(f)

(g)

(h)

<R, G> index	Cost	δ	Cost-type
1-1	7	1.5	a
2-1	7	1.5	a
3-1	6	2	h
4-1	6	1	a

(i)

(j)

(k)

(l)

	G1	G2	G3	G4
R1	7	4.5	6	8.49
R2	7	4.5	5.5	8.94
R3	6	3.16	5	3
R4	6	4.12	6.32	4

(m)

	G1	G2	G3	G4
R1	7	4.5	6	8.49
R2	7	4.5	5.5	8.94
R3	6	3.16	5	3
R4	6	4.12	6.32	4

(n)

Figure 2: Step-by-step illustration of Algorithm 1 on the problem introduced in Figure 1

Therefore, the A-cost for the R1-G2 pair is explored, which yields 4.5. Now, the new $hcost_{min}$ is 6 (for G3), which is greater than the current $acost_{min}$ (4.5). So, there is no need to explore A-cost corresponding to new $hcost_{min}$ as that would result in a cost greater than or equal to 6 (thus, leaving the current $acost_{min}$ unchanged to 4.5). This is the reason for the presence of two blue cells for R1.

Figure 2(c) shows the equality subgraph. The vertices on the left denote the robots, whereas the ones on the right denote the goals. The initial feasible labeling, using the minimum A-costs, is provided in parentheses. Edges in the maximum cardinality matching M are shown in red. We see that robots R2 and R4 are unmatched. As the current matching is not a total matching, we proceed according to line 15 and line 16 of Algorithm 1 and find the minimum vertex cover. The goals G2 and G4 are in the minimum vertex cover and are shown as red-colored vertices. At this stage, R1, R2, R3, and R4 are the uncovered robots, while G1 and G3 are the uncovered goals as they do not belong to minimum vertex cover. In Figure 2(d), we illustrate the search procedure for minimum slack δ_{min} that is computed on A-cost. At first, 0.89 is retrieved as the minimum slack, but it is computed on H-cost. So, this H-cost is replaced by exploring the A-cost, and the slack is updated to 2.5. On searching the minimum slack again, we get 1, which is computed on A-cost. So, δ_{min} is 1. Once, δ_{min} is available, those H-costs for which δ equals δ_{min} are replaced by corresponding A-costs (see Figure 2(e)). Here, the green row corresponds to δ_{min}, while the blue row represents the case for which δ, when computed on H-cost, equals δ_{min}, and due to which the corresponding A-cost is explored. The labels of vertices are updated using the rule in lines 20 and 21 of Algorithm 1. In Figure 2(f), as there are no covered robots in the minimum vertex cover, there is no change in the labels of robots. However, for the uncovered goals G1 and G3, the labels are increased by δ_{min} (shown in blue color). The updated equality subgraph, having a new green-colored edge between R2-G3, is shown in Figure 2(g).

In Figure 2(h), the maximum cardinality matching is found on the updated equality subgraph, and here we see that R4 is still unassigned. Thus, one more iteration is required, which is shown in Figure 2(i)-(l). We find in Figure 2(l) that all the robots are assigned. Figure 2(m) reflects the A-costs explored till now in blue color. Figure 2(n) depicts the resultant goal-assignment in green-colored cells.

3.4 Theoretical Guarantees

Algorithm 1 provides the following guarantees.

Theorem 1 (Correctness). *Algorithm 1 provides an optimal solution to the multi-robot goal assignment problem (Problem 1). That is, either the robots or the goals, whichever is less in number, get assigned completely by the algorithm, and the total cost of the resultant assignment is minimum.*

Theorem 2 (Time complexity). *The worst-case time complexity of Algorithm 1 is $\mathcal{O}((\min(R, G))^2 \times (RG\Psi))$, where Ψ denotes the number of cells in the workspace.*

For ease of comparison, consider the case when R and G are equal. The time complexity of Algorithm 1, which incorporates path planning into the solution of the goal assignment problem, boils down to $\mathcal{O}(R^4\Psi)$. The algorithm that solves the path planning problem for all robot-goal pairs before employing the Hungarian method for goal assignment has the worst-case time complexity of $\mathcal{O}(R\Psi\log(\Psi) + R^3)$. Despite the poor worst-case complexity, our algorithm performs well in practice, which we show empirically in Section 4.3.

4 Evaluation

4.1 Experimental Setup

We evaluate our algorithm by comparing it with a baseline algorithm that computes the optimal path *for each* robot-goal pair and then finds an optimal assignment using the Hungarian method. We have implemented both the baseline algorithm and the proposed algorithm in Python[1].

We evaluate our algorithm on randomly generated 2D workspaces and benchmark 2D and 3D workspaces. We use two evaluation metrics: (a) the number of robot-goal pairs for which the actual movement cost is explored and (b) the total execution time. We also report the speedup (SU) that our algorithm achieves over the baseline algorithm.

We run all the experiments in a desktop machine with Intel® Core$^{\text{TM}}$ i7-8700 CPU @ 3.20 GHz processor, 32 GB RAM and Ubuntu 20.04. We run each experiment for 20 times and report the mean and standard deviation for the evaluation metrics. In Table 1 and Table 2, all the reported results except the speedup are rounded to nearest integers.

4.2 Baseline

As the baseline, we consider an algorithm which first computes the optimal obstacle-free path for each robot-goal pair and then uses the Hungarian method to find the optimal goal assignment. For the first step, we consider two algorithms: (a) FRA* and (b) Dijkstra's shortest path algorithm. To compare our algorithm with the best possible baseline, we perform a study to find which among the two possibilities is better. Figure 3 shows the result of this comparative study. We note that the baseline implementation using Dijkstra's algorithm wins consistently over the one using FRA* in all the cases. This happens due to the reason that when we deal with a large number of robots and goals, and the number of FRA* calls is high, the overheads that come along with FRA*, such as calculating and updating H-costs for the nodes in OL, outweigh its quality of being an informed search. Thus, we compare our algorithm with the baseline implemented using Dijkstra's algorithm in all our experiments presented in the following subsection.

4.3 Experimental Results

Randomly Generated Workspaces Table 1 shows the experimental results on randomly generated 2D workspaces having an equal number of robots and goals. We vary three paramaters to thoroughly examine the efficacy of our algorithm: a) workspace size, b) the number of robots, and c) the obstacle density (OD) which is the percentage of cells of a workspace that are occupied by obstacles.

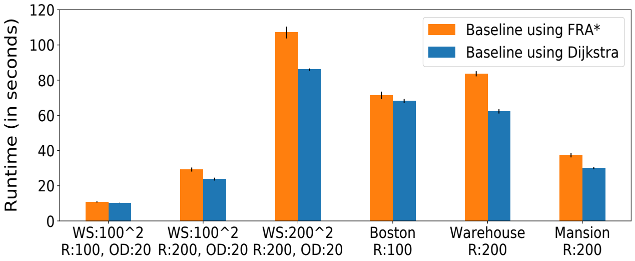


Figure 3: A Comparative Study of Two Baseline Possibilities

The workspace size varies in the first two blocks of the table. We observe that owing to the demand-driven computation of A-costs in our algorithm, the number of robot-goal pairs for which the A-costs are computed are significantly less as compared to that in the baseline. The baseline is bounded to explore A-costs between each robot-goal pair, which becomes prohibitive as the workspace size increases. This advantage that our algorithm has over the baseline grows with the increase in the workspace size and is reflected by the increasing speedup.

The number of robots R varies in the next two blocks of Table 1. For different values of R, Algorithm 1 consistently records better runtime than the baseline. But we observe a decline in the speedup with an increase in R (and G), for which the reason is as follows. In general, as R and G increase, the rate of increase in runtime is more for the goal assignment than the path computation due to the inherent time complexities. But for the given values of workspace size and R (and G), in the case of baseline, the increment in the time required to solve goal assignment is meager in front of the increment in the time required to compute paths (since the baseline computes the paths for all robot-goal pairs). While in Algorithm 1's case, the increment in the time needed to solve goal assignment is considerable when compared to the increment in the time required to compute paths (since our approach computes quite a few paths).

The obstacle density varies in the last two blocks of Table 1. The speedup declines with an increase in OD and this is because of the following. The runtime of baseline decreases since with more obstacles in a fixed-size workspace, Dijkstra's shortest path algorithm has fewer cells to process. On the other hand, our algorithm shows nearly consistent runtime, and the intuition behind it is that with the increase in obstacles, the benefit gained by the decrease in the number of cells to process is negated by the increase in the number of computations of A-cost by our algorithm. Note that with an increase in OD, the difference between H-cost and corresponding A-cost tends to increase which leads to an increase in the number of A-cost computations. Overall, from the experimental results in Table 1, we infer

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

WS Size	OD	R	# Exp	Runtime (s)		SU
			Algo 1	Baseline	Algo 1	
100^2	20	100	798±137	10±0	1±0	10.0
200^2	20	100	812±232	40±1	3±1	13.3
300^2	20	100	732±124	94±1	5±1	18.8
400^2	20	100	642±115	175±2	7±2	25.0
100^2	20	200	2388±686	23±1	5± 1	4.6
200^2	20	200	1764±439	85±1	8± 2	10.6
300^2	20	200	1853±487	197±2	14± 4	14.1
400^2	20	200	1901±365	355±4	22± 4	16.1
100^2	20	100	758±173	10±0	1±0	10.0
100^2	20	200	2169±461	23±1	5±1	4.6
100^2	20	300	4027±987	38±1	11±2	3.5
100^2	20	400	5238±1237	55±2	18±3	3.1
200^2	20	100	733±159	40±0	3±1	13.3
200^2	20	200	2141±452	86±1	10±2	8.6
200^2	20	300	3336±767	137±2	20±4	6.9
200^2	20	400	5054±783	193±3	35±6	5.5
100^2	10	100	581±80	12±0	1±0	12.0
100^2	15	100	677±122	11±0	1±0	11.0
100^2	20	100	966±187	10±0	1±0	10.0
100^2	25	100	900±129	9±0	1±0	9.0
200^2	10	200	1427±294	104±1	8±2	13.0
200^2	15	200	1661±286	95±1	9±1	10.6
200^2	20	200	1911±451	84±1	9±2	9.3
200^2	25	200	2312±609	81±2	10±2	8.1

Table 1: Experimental Results on Random Workspace

that the speedup of our algorithm is inversely proportional to the congestion in the workspace.

Standard Benchmark Workspaces In Table 2, we compare the performance of Algorithm 1 with that of the baseline algorithm for standard workspaces available in the literature (Stern et al. 2019; Sturtevant 2012). We consider four real-world city workspaces (Boston, Paris, Sydney and Shanghai) in 2D and observe that Algorithm 1 outperforms the baseline significantly for each of them. Algorithm 1 also proves it versatility by outperforming the baseline on other different types of 2D workspaces such as warehouse, mansion and den. Mansion and den are taken from the Dragon Age video games.

The last workspace in Table 2 is a 3D workspace resembling an environment in Warframe game (Brewer and Sturtevant 2018). In a 3D workspace, we consider that a robot can move in 26 directions. 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.

Different Number of Robots and Goals Table 3 presents the results for cases having unequal number of robots and goals in random workspaces. The speedup shows the scalability that Algorithm 1 offers to multi-robot systems. Note that the speedup achieved when $R = 100$ and $G = 150$ is much greater than the case when $R = 100$ and $G = 100$. This is because in the former case, the scope of conflict among the robots reduces and it takes fewer iterations to obtain the matching.

WS Size	R	# Exp	Runtime (s)		SU
		Algo 1	Baseline	Algo 1	
Boston	100	1193±291	67±1	9±3	7.4
256×256	200	4010±1234	139±2	26±6	5.4
Paris	100	1184±282	65±0	11±5	5.9
256×256	200	3527±895	138±2	27±9	5.1
Sydney	100	751±228	70±1	5±2	14.0
256×256	200	1850±493	147±1	13±4	11.3
Shanghai	100	782±204	70±1	5±2	14.0
256×256	200	1905±320	145±1	14±3	10.4
Warehouse	100	735±223	28±0	2±1	14.0
123×321	200	2303±658	62±1	10±2	6.2
Mansion	100	2224±485	12±0	5±1	2.4
270×133	200	7264±1787	28±1	15±3	1.9
Den	100	1765±675	39±1	12±5	3.3
257×256	200	5580±1904	83±1	30±9	2.8
Warframe	100	492±39	7333±71	58±13	126.4
146×54×105	200	1394±197	14613±557	152±55	96.1

Table 2: Experimental Results on Benchmark Workspace

WS Size	OD	R	G	# Exp	Runtime (s)		SU
				Algo 1	Baseline	Algo 1	
100^2	20	50	100	168±18	5.2±0.1	0.1±0.0	52.0
		100	150	432±61	10.8±0.4	0.5±0.2	21.6
		100	200	374±37	12.6±0.6	0.5±0.1	25.2
		200	300	930±111	28.2±1.7	1.8±0.3	15.7
100^2	20	100	50	127±22	10.6±0.3	0.1±0.0	106.0
		150	100	326±46	17.2±0.5	0.4±0.1	43.0
		200	100	271±21	27.1±0.7	0.4±0.1	67.8
		300	200	690±103	47.9±2.6	1.7±0.4	28.2

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

5 Conclusion

We have presented a scalable centralized algorithm to solve the optimal goal assignment problem for multi-robot systems. Experimental results show that our method can outperform the state-of-the-art method by an order of magnitude both in 2D and 3D environments irrespective of whether the number of robots is equal to the number of goals or not. Our algorithm thus paves the way for the deployment of large-size multi-robot systems with hundreds of robots for time-critical applications, which to the best of our knowledge, was not possible before. In the future, we would like to address the variant of the goal assignment problem where a robot is allowed to be assigned with multiple goals (Chen et al. 2021).

References

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.

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

Burkard, R. E.; and Cela, E. 1999. Linear Assignment Problems and Extensions. In *Handbook of combinatorial optimization*, 75–149. Springer.

Carpaneto, G.; Martello, S.; and Toth, P. 1984. *The Problem of Assignment : methods and algorithms*. IAC.

Chen, Y. F.; Liu, M.; Everett, M.; and How, J. P. 2017. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *ICRA*, 285–292.

Chen, Z.; Alonso-Mora, J.; Bai, X.; Harabor, D. D.; and Stuckey, P. J. 2021. Integrated Task Assignment and Path Planning for Capacitated Multi-Agent Pickup and Delivery. *IEEE Robotics and Automation Letters*, 6(3): 5816–5823.

Choi, H.-L.; Brunet, L.; and How, J. P. 2009. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Trans. Robotics*, 25(4): 912–926.

Chopra, S.; Notarstefano, G.; Rice, M.; and Egerstedt, M. 2017. A Distributed Version of the Hungarian Method for Multirobot Assignment. *IEEE Trans. Robotics*, 33(4): 932–947.

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.

Ford, L. R.; and Fulkerson, D. R. 1956. A primal-dual algorithm for linear programs. *Linear Inequalities and Related Systems, Annals of Mathematics Study*, 38: 171–181.

Giordani, S.; Lujak, M.; and Martinelli, F. 2010. A Distributed Algorithm for the Multi-Robot Task Allocation Problem. In *Trends in Applied Intelligent Systems*, 721–730.

Giordani, S.; Lujak, M.; and Martinelli, F. 2013. A distributed multi-agent production planning and scheduling framework for mobile robots. *Computers and Industrial Engineering*, 64(1): 19–30.

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.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Hennes, D.; Claes, D.; Meeussen, W.; and Tuyls, K. 2012. Multi-robot collision avoidance with localization uncertainty. In *AAMAS*, 147–154.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J.; and Ayanian, N. 2018. Conflict-based search with optimal task assignment. In *AAMAS*, 757–765.

Kuhn, H. W. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2): 83–97.

Lawler, E. 1976. Combinatorial Optimization: Networks and Matroids, Holt, Reinhart, and Winston, N. *New York*.

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.

Liu, L.; and Shell, D. 2012. A Distributable and Computation-flexible Assignment Algorithm: From Local Task Swapping to Global Optimality. In *RSS*.

Long, P.; Fan, T.; Liao, X.; Liu, W.; Zhang, H.; and Pan, J. 2018. Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning. In *ICRA*, 6252–6259.

Madridano, Á.; Al-Kaff, A.; Gómez, D. M.; and de la Escalera, A. 2019. Multi-Path Planning Method for UAVs Swarm Purposes. In *ICVES*, 1–6.

McGinnis, L. F. 1983. Implementation and testing of a primal-dual algorithm for the assignment problem. *Operations Research*, 31(2): 277–291.

Munkres, J. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1): 32–38.

Papadimitriou, C. H.; and Steiglitz, K. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc.

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*. Berlin, Germany.

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.