

Article

Real-Time Multi-Robot Mission Planning in Cluttered Environment

Zehui Lu , Tianyu Zhou  and Shaoshuai Mou * School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN 47907, USA;
lu846@purdue.edu (Z.L.); zhou1043@purdue.edu (T.Z.)

* Correspondence: mous@purdue.edu

Abstract: Addressing a collision-aware multi-robot mission planning problem, which involves task allocation and path-finding, poses a significant difficulty due to the necessity for real-time computational efficiency, scalability, and the ability to manage both static and dynamic obstacles and tasks within a complex environment. This paper introduces a parallel real-time algorithm aimed at overcoming these challenges. The proposed algorithm employs an approximation-based partitioning mechanism to partition the entire unassigned task set into several subsets. This approach decomposes the original problem into a series of single-robot mission planning problems. To validate the effectiveness of the proposed method, both numerical and hardware experiments are conducted, involving dynamic obstacles and tasks. Additionally, comparisons in terms of optimality and scalability against an existing method are provided, showcasing its superior performance across both metrics. Furthermore, a computational burden analysis is conducted to demonstrate the consistency of our method with the observations derived from these comparisons. Finally, the optimality gap between the proposed method and the global optima in small-size problems is demonstrated.

Keywords: multi-robot systems; mission planning; task allocation



Citation: Lu, Z.; Zhou, T.; Mou, S. Real-Time Multi-Robot Mission Planning in Cluttered Environment. *Robotics* **2024**, *13*, 40. <https://doi.org/10.3390/robotics13030040>

Academic Editors: Pan Zhao, Minghui Zheng, Yu Zhang and Naira Hovakimyan

Received: 30 January 2024

Revised: 22 February 2024

Accepted: 26 February 2024

Published: 28 February 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Autonomous unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) can replace humans for dangerous tasks such as surveillance and search-and-rescue. Recently, some receding-horizon motion planning methods [1–4] guide an autonomous robot to explore and go to a destination in a complex environment. These methods typically require a planning hierarchy. On top of this hierarchy, a path planner such as [5–9] generates a sequence of sparse way-points based on the perception of the environment. A group of autonomous robots has more capabilities than a single robot in applications such as surveillance, information sensing, navigation, and search-and-rescue. Suppose collision-free, non-conflict sparse paths for the robot team can be generated and adapted at run-time. In that case, the team can explore a complex environment and perform complicated missions efficiently.

In this paper, a task is defined as a location of interest that one robot must visit. Given a set of robots and tasks, a collision-aware multi-robot mission planning (MRMP) problem is defined twofold, i.e., finding optimal and conflict-free task allocations for robots and then generating collision-free paths such that robots can visit these task positions. The former is categorized as a multi-agent task allocation (MATA) problem, and the latter is defined as a multi-agent path-finding (MAPF) problem. The optimal objective of MAPF is typically to minimize the total traveling distance. For a multi-robot system, real-time mission planning in a cluttered environment is necessary when deploying autonomous robots in a complex environment, especially when obstacles and tasks are dynamic. A hardware example and a simulation example of MRMP problems are shown in Figure 1 and Figure 2, respectively. This paper considers MRMP problems defined as ST-SR-TA (Single-Task Robots, Single-Robot Tasks, Time-Extended Assignment) problems [10]. Here, tasks are assumed to be

homogeneous and independent of each other, i.e., no temporal logic requirements; robots are assumed to be homogeneous regarding mission functionality. Since this problem is proven to be NP-hard [10], there is a trade-off for MRMP problems between real-time performance and optimality. Furthermore, the scalability of an underlying algorithm, in terms of the number of robots and tasks, is crucial in multi-robot systems.

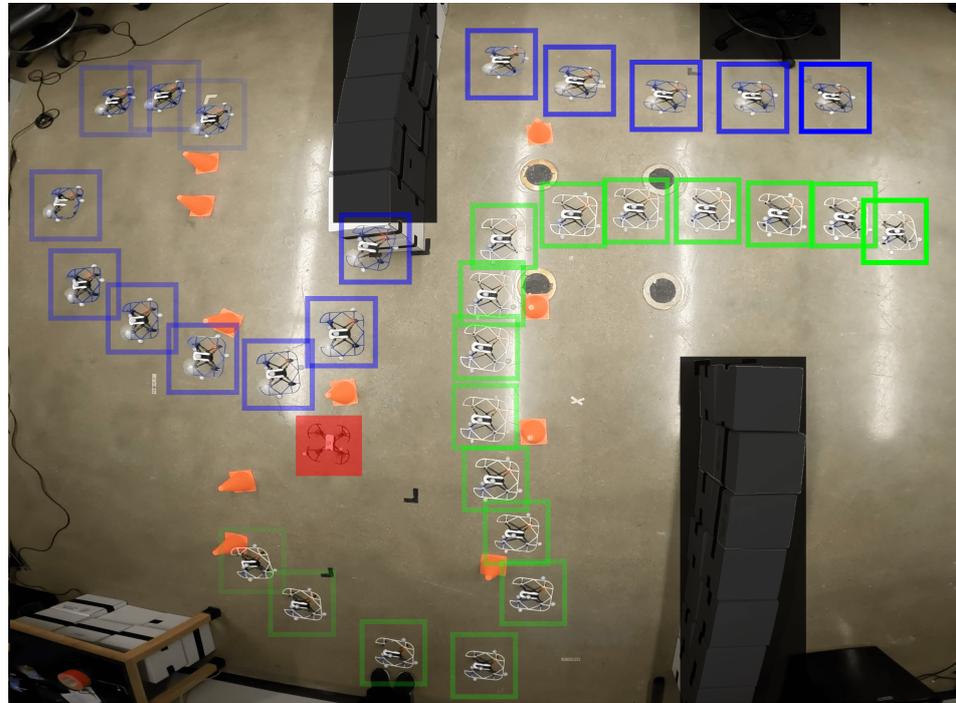


Figure 1. A screenshot of an experiment on multi-robot mission planning with a dynamic obstacle. Orange cones represent tasks and black areas indicate no-fly zones. A manually controlled quadrotor with red shading represents a dynamic obstacle with infinite height. Two quadrotors with blue/green boxes are the ego robots and transparency indicates time. Details are described in Section 4.1. The complete video can be found as Video S1 in Supplementary Materials.

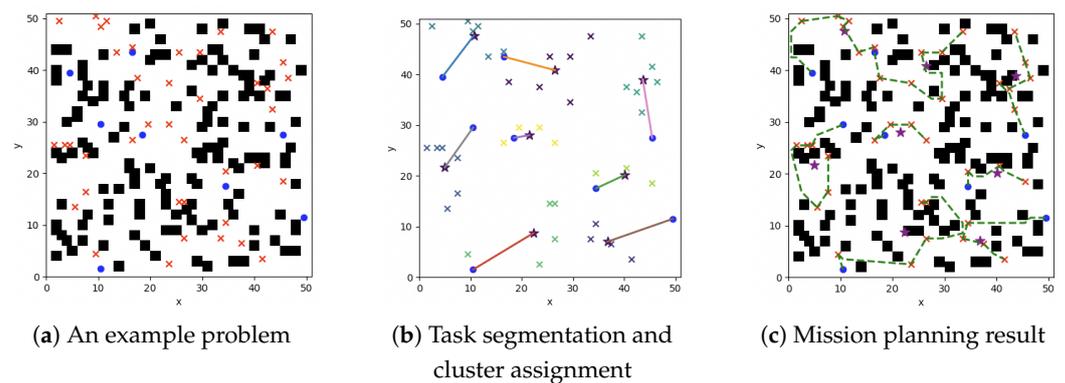


Figure 2. An example for MRMP with 8 robots and 40 tasks. (a) shows the problem in a 50×50 grid map with 150 obstacles, where the blue dots and red crosses indicate the positions of robots and tasks, respectively; (b) shows the task segmentation and cluster assignment result, where those tasks in the same color are within the same cluster; the purple stars indicate the positions of cluster centroids and an edge between a robot and a cluster centroid represents assignment; (c) shows the task allocation orders and collision-free paths, where the dashed lines in green indicate the paths. The computation time is 44.6 ms.

1.1. Related Work

The literature on MRMP problems can be divided into two categories, i.e., solving MATA and MAPF problems sequentially or in an integrated way.

The methods related to MATA can be mainly categorized as auction-based and searching-based methods. Auction-based approaches are derived from a concept in finance where each agent aims to maximize their reward by giving higher bids. The process must consider maximizing a global reward and include conflict resolution. Ref. [11] utilizes auction-based protocols to bid task assignments. CBBA (Consensus-Based Bundle Algorithm) [12] employs a decentralized consensus procedure for task conflict resolution and then generates task allocation for agents. IACA (Iterated Auction Consensus Algorithm) [13] proposes a similar iterative but resilient auction process and can remove malicious bids during the auction. Ref. [14] proposed an auction-based algorithm to deal with task allocation problems with time window constraints. Ref. [15] utilizes Behavior Tree to coordinate a sequence of actions among agents with the auction process. Ref. [16] produces task sequences with minimum communications by combining the greedy algorithm and the auction process. Although the auction-based approaches are decentralized, the process of auction and conflict resolution can be time-consuming, especially when the problem size is large. In addition, the auction heuristic barely includes environmental information, e.g., the impact of obstacles on the cost/reward. Thus, the auction result is not necessarily optimal when obstacles are present and may even lead to a bad solution.

Search-based methods rely on a fixed structure of information, e.g., the number of assigned tasks for each agent is known and fixed. Ref. [17] proposes a decentralized genetic algorithm (GA) to search a task sequence parallelly. Ref. [18] proposes a graph-based search method to allocate tasks to agents given a finite linear temporal logic objective, where the allocation order is partially known. Ref. [19] builds an Optimized Directed Roadmap Graph (ODRM) by sampling first and then navigates agents on this graph. Although searching for paths on an ODRM is faster than on the most common occupancy grid map, generating and updating such a graph at run-time can be time-consuming in a cluttered and dynamic environment.

Additionally, researchers have explored other methodologies for solving MATA problems in recent years. Ref. [20] investigates a heterogeneous MATA problem, where the objective is to minimize the maximum travel cost for any agents. Ref. [21] proposes a deep reinforcement learning algorithm that utilizes policy gradient updates to determine the optimal allocation schedule for each robot. Ref. [22] approaches MATA as a potential game, using this framework to reach a mutually agreeable task assignment by identifying a Nash equilibrium among the agents. Ref. [23] formulates MATA as a cooperative game, leveraging the Shapley value to compute the average marginal contribution of each robot. By ranking and clustering robots and tasks based on their Shapley value, the initial problem can be partitioned into smaller, more manageable sub-problems.

Since most of the recent literature focuses on the integration of MATA and MAPF problems, this paper omits the literature on MAPF problems. As for the literature on solving MATA and MAPF problems sequentially, it is mainly categorized into auction-based and search-based methods. Based on CBBA, Ref. [24] first generates task sequences without any obstacle information and then utilizes Dijkstra's algorithm [5] to find collision-free paths given the sequences. Ref. [25] proposes a two-stage GA-based approach where each agent first determines its task sequence using a genetic algorithm and then negotiates with other agents to exchange tasks if that reduces the cost. Then, collision-free paths are generated similarly to the method in [24].

There are also some special cases of MRMP problems that have raised significant interest, such as multi-robot pickup and delivery [26], and vehicle routing problems. Some special specifications are adopted for these problems. For example, the task set for each agent is prescribed; each agent can only be assigned one task; the initial positions for agents are the same; etc. This paper considers a general MRMP problem without these special specifications.

There is also some literature on the integrated MRMP methods. Ref. [27] focuses on simultaneous task allocation and planning for a complex goal that consists of temporal logic sub-tasks. Ref. [27] emphasizes the capability of a heterogeneous robot team to perform a complex goal, whereas the MRMP problem in this paper focuses on homogeneous agents and tasks. Ref. [28], as a fully centralized optimization-based method, first obtains a single tour that connects all the tasks without any obstacle information by solving a traveling salesman problem; then, it uses a heuristic policy to partition the tour to generate a task allocation sequence for each agent; finally, it generates collision-free paths. Although Ref. [28] deals with the same problem as this paper, its computation time is stably around 55 s, with 5–20 agents and 10–50 tasks on a map with random obstacles. Its success rate varies from 0.35 to 1.0, depending on the number of agents.

From the methodology perspective, there are primarily three types of methods for MRMP problems with homogeneous robots/tasks and no temporal logic constraints, i.e., decentralized auction-based, distributed GA-based (genetic algorithm), and centralized optimization-based methods. Decentralized auction-based methods, as mentioned above, suffer from inefficient auction and negotiation processes and a lack of obstacle information during the auction process. Distributed GA-based methods might have good real-time performance for small-size problems but they notably depend on the selection of GA parameters. Also, many methods assume the number of assigned tasks for each robot is known and fixed, whereas this paper does not. As for optimization-based methods, they barely utilize obstacle information in the first place and not in a distributed manner, i.e., directly solving the entire allocation problem.

1.2. Contributions, Organization, and Notations

This paper proposes a real-time parallel multi-robot mission planning algorithm, named RPM, for homogeneous robots and tasks. RPM first utilizes obstacle information as heuristics to approximate the cost of an ordered task allocation and path sequence by a metric from an unordered task set. With this approximation, RPM can partition the entire problem into several parallel sub-problems and distribute them to each robot. Then, each robot finds an optimal task allocation and path sequence for each sub-problem. Due to the approximation and the parallel manner, RPM makes a balance between computational performance and scalability. The main contributions are as follows:

1. A parallel real-time algorithm, RPM;
2. Capability of handling dynamic obstacles and tasks in a cluttered environment at run-time;
3. Good scalability in terms of the number of robots and tasks, and relatively good optimality;
4. Computational burden analysis for RPM.

The rest of this paper is organized as follows: Section 2 defines the multi-robot mission planning problem and formulates it as an intractable optimization problem. Section 3 introduces the proposed algorithm, RPM, in three phases. Section 4 shows several experiments with static/dynamic obstacles and tasks, conducts the scalability and optimality comparisons with an existing algorithm, analyzes the computational burden for RPM, and shows the optimality gap among different algorithms in small-size problems. Section 5 concludes the paper and discusses future improvements and challenges.

Notations. For a point $p \in \mathbb{R}$, $\{p\} \subset \mathbb{R}$ denotes a set containing that point as its only element. Set subtraction is $A \setminus B = \{x \in A \mid x \notin B\}$. \mathbb{Z} denotes the integer set. \mathbb{Z}_+ denotes the positive integer set. The cardinality of a set A is denoted as $|A|$.

2. Problem Formulation

The configuration space, $X \subseteq \mathbb{R}^n$, is the set of all positions reachable by a robot. Denote a robot position set $\mathcal{X} = \{p_1, \dots, p_{n_a}\}$ of n_a robots and $p_i \in X$ as the position of robot i . Denote a task positions set $\mathcal{T} = \{t_1, \dots, t_{n_t}\}$ of n_t tasks and $t_i \in X$ as the position of task i . Define the robot and tasks index sets $\mathcal{I} \triangleq \{1, \dots, n_a\}$ and $\mathcal{J} \triangleq \{1, \dots, n_t\}$, respectively. Suppose that a robot completes a task when the distance between two entities is less than a prescribed non-negative constant ϵ , i.e., $\|p_i - t_j\|_2 \leq \epsilon$, $\epsilon \geq 0$. Denote an obstacle positions set as $\mathcal{O} = \{o_1, \dots, o_{n_o}\}$ of n_o obstacles, where $o_i \in X$ is the position of obstacle i . Denote $\mathcal{P}_i \triangleq (p_i^0, p_i^1, \dots, p_i^{n_{p,i}-1}) \subset X$ as an ordered sequence of positions associated with robot i which denotes a path starting from p_i^0 and ending at $p_i^{n_{p,i}-1}$, where $n_{p,i} \triangleq |\mathcal{P}_i|$ denotes the number of positions in \mathcal{P}_i .

Inspired by [12], the collision-aware MATA problem is written as the following integer programming:

$$\min_{x, r_1, \dots, r_{n_a}} \sum_{i=1}^{n_a} \sum_{j=1}^{n_t} c_{ij}(x_i, r_i, \mathcal{O}) x_{ij} \quad (1a)$$

$$\text{s.t.} \quad \sum_{j=1}^{n_t} x_{ij} \leq n_t, \quad \forall i \in \mathcal{I}, \quad (1b)$$

$$\sum_{i=1}^{n_a} x_{ij} = 1, \quad \forall j \in \mathcal{J}, \quad (1c)$$

$$\sum_{i=1}^{n_a} \sum_{j=1}^{n_t} x_{ij} = n_t, \quad (1d)$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{I} \times \mathcal{J}, \quad (1e)$$

where $x_{ij} = 1$ if task j is assigned to robot i and 0 otherwise; $x_i \in \{0, 1\}^{n_t}$ is the task assignment vector for robot i ; x_{ij} is the j -th element of x_i ; $x = [x_1' \dots x_{n_a}']' \in \{0, 1\}^{n_t n_a}$. The vector $r_i \in \{\mathcal{J} \cup \{\emptyset\}\}^{n_a}$ denotes an ordered sequence of tasks, i.e., the task allocation order, for robot i ; its k -th element is $j \in \mathcal{J}$ if task j is the k -th task of robot i 's assignment; $r_i = \emptyset$ if robot i has no assignment. The collision-aware cost of task j being assigned to robot i followed by an order r_i is defined by $c_{ij}(x_i, r_i, \mathcal{O}) \geq 0$. In the context of mission planning, this cost typically represents traveling distance, fuel consumption, etc. Constraint (1b) indicates that each robot can at most be assigned n_t tasks; (1c) requires that each task must be assigned to only one robot; (1d) enforces that every task must be assigned.

Given a particular assignment x , the task allocation order r_i for each robot i is not unique. Therefore, an implicit mapping from x to the task allocation order set $\mathcal{R} \triangleq \{r_1, \dots, r_{n_a}\}$ needs to be determined. How to find the mapping from x to \mathcal{R} is also a part of the problem (1).

Given an order set \mathcal{R} and the current positions of robots \mathcal{X} , the collision-aware MAPF problem is written as follows:

$$\min_{\mathcal{P}_1, \dots, \mathcal{P}_{n_a}} \sum_{i=1}^{n_a} \ell_i(\mathcal{P}_i) \quad (2a)$$

$$\text{s.t.} \quad p_i^0 = p_i, \quad \forall i \in \mathcal{I}, \quad (2b)$$

$$\mathcal{R} \text{ is determined by (1)}, \quad (2c)$$

$$\mathcal{P}_i \text{ satisfies the order } r_i, \quad \forall i \in \mathcal{I}, \quad (2d)$$

$$\mathcal{P}_i \cap \mathcal{O} = \emptyset, \quad \forall i \in \mathcal{I}, \quad (2e)$$

where $\ell_i(\mathcal{P}_i) = \sum_{j=0}^{|\mathcal{P}_i|-2} \|p_i^{j+1} - p_i^j\|_2$ is the traveling distance of path \mathcal{P}_i . This paper assumes that $\mathcal{P}_i \cap \mathcal{O} = \emptyset$ if and only if $\|p_i^j - o_k\|_2 \geq \delta > 0 \quad \forall p_i^j \in \mathcal{P}_i$ and $\forall o_k \in \mathcal{O}$.

Based on (1) and (2), the collision-aware MRMP problem in this paper is formulated as follows:

$$\min_{\mathbf{x}, \mathcal{R}, \mathcal{P}} \sum_{i=1}^{n_a} \sum_{j=1}^{n_t} c_{ij}(\mathbf{x}_i, \mathbf{r}_i, \mathcal{O}) x_{ij} \quad (3a)$$

$$\text{s.t.} \quad \sum_{j=1}^{n_t} x_{ij} \leq n_t, \forall i \in \mathcal{I}, \quad (3b)$$

$$\sum_{i=1}^{n_a} x_{ij} = 1, \forall j \in \mathcal{J}, \quad (3c)$$

$$\sum_{i=1}^{n_a} \sum_{j=1}^{n_t} x_{ij} = n_t, \quad (3d)$$

$$x_{ij} \in \{0, 1\}, \forall (i, j) \in \mathcal{I} \times \mathcal{J}, \quad (3e)$$

$$\mathcal{P} \triangleq \{\mathcal{P}_1, \dots, \mathcal{P}_{n_a}\} \text{ is determined by (2),} \quad (3f)$$

$$\mathcal{R} \text{ is determined by (1),} \quad (3g)$$

where $\sum_{j=1}^{n_t} c_{ij}(\mathbf{x}_i, \mathbf{r}_i, \mathcal{O}) x_{ij}$ evaluates robot i 's collision-aware traveling distance given a particular assignment and allocation order.

Solving the task assignment \mathbf{x} , the allocation order \mathcal{R} , and the collision-free path \mathcal{P} altogether is challenging because \mathbf{x} , \mathcal{R} , and \mathcal{P} are coupled together in (1)–(3). Furthermore, the collision-aware MRMP problem (3) is not even tractable since it is proven to be NP-hard [10]. This paper attempts to obtain a sub-optimal solution to the collision-aware MRMP problem scalably and in real-time, especially when the environment is unconstructed and cluttered, and the obstacles and tasks are potentially dynamic.

3. Algorithm

This paper proposes a real-time parallel algorithm RPM to obtain a sub-optimal solution to (3) in a scalable way. Instead of considering the exact coupled cost $c_{ij}(\mathbf{x}_i, \mathbf{r}_i, \mathcal{O})$, RPM utilizes task-based heuristics to approximate the cost of an ordered path by an unordered task set. With this approximation, RPM can partition the entire task set into several subsets and assign each task subset to one robot given the unordered heuristics. Then, each robot only needs to solve a sub-problem, i.e., a single-robot mission planning problem. Specifically, RPM consists of three phases: (1) Task Segmentation: partitioning the entire task set into several subsets; (2) Cluster Assignment: assigning each robot a task subset; (3) Single-Robot Mission Planning: finding an optimal task allocation order and collision-free path for each robot.

3.1. Task Segmentation

The entire task set \mathcal{T} is partitioned into n_a clusters $\{\mathcal{T}_1, \dots, \mathcal{T}_{n_a}\}$, where each cluster possibly includes many tasks. Note that \mathcal{T}_i has not been assigned to any robots yet. The tasks within a cluster have a minimal distance to the centroid of this cluster. An iterative k-means clustering algorithm [29] is used here, which minimizes the summation of the within-cluster sum of squares (WCSS), i.e.,

$$\min_{\mathcal{T}_1, \dots, \mathcal{T}_{n_a}} \sum_{i=1}^{n_a} \sum_{\mathbf{t} \in \mathcal{T}_i} \|\mathbf{t} - \mathbf{c}_i\|_2^2 \quad (4a)$$

$$\text{s.t.} \quad \mathcal{T} = \cup_{i=1}^{n_a} \mathcal{T}_i, \quad (4b)$$

$$\mathcal{T}_i \cap \mathcal{T}_j = \emptyset, \forall i \neq j, \quad (4c)$$

$$\mathbf{c}_i = (\sum_{\mathbf{t} \in \mathcal{T}_i} \mathbf{t}) / |\mathcal{T}_i|, \forall i, \quad (4d)$$

where $\mathcal{T}_i = \{\mathbf{t}_j \mid \forall j \in \mathcal{I}_{c,i}\}$ and $\mathcal{I}_{c,i}$ is the task index set that is associated with the tasks within cluster \mathcal{T}_i ; $\mathbf{c}_i \in \mathbb{R}^n$ is the centroid of tasks within \mathcal{T}_i . Denote $\mathcal{C} \triangleq \{\mathbf{c}_1, \dots, \mathbf{c}_k\}$.

As described in (3), the objective is to minimize the total traveling distance. However, the cost of each robot visiting a known task set is unknown before a task allocation order is determined. Hence, for each task subset \mathcal{T}_i , an ordered sequence's length is approximated by an unordered set's WCSS, i.e., $\sum_{\mathbf{t} \in \mathcal{T}_i} \|\mathbf{t} - \mathbf{c}_i\|_2^2$, since the tasks within \mathcal{T}_i have a smaller

WCSS associated with c_i than $c_j \forall j \neq i$. The task segmentation problem (4) can be solved iteratively and the details are in Algorithm 1. An example is shown in Figure 2.

Algorithm 1: Task Segmentation

Input: $\mathcal{T}, N \in \mathbb{Z}_+$
1 Initialize $\{\mathcal{T}_1, \dots, \mathcal{T}_{n_a}\}, \{\mathcal{I}_{c,1}, \dots, \mathcal{I}_{c,n_a}\}, \mathcal{C}$ by k-means++ [30], $iter = 0$
2 **while** $iter < N$ **do**
3 **for** task $t_i = t_1$ to t_{n_t} **do**
4 $idx \leftarrow$ the index of t_i 's nearest centroid
5 $\mathcal{I}_{c,idx}.append(i)$
6 **for** $j = 1$ to n_a **do**
7 $c_j \leftarrow$ mean of all tasks within cluster j
8 $iter \leftarrow iter + 1$
9 **for** $i = 1$ to n_a **do** $\mathcal{T}_i \leftarrow \{t_j \mid \forall j \in \mathcal{I}_{c,i}\}$
10 **return** $\{\mathcal{T}_1, \dots, \mathcal{T}_{n_a}\}, \{\mathcal{I}_{c,1}, \dots, \mathcal{I}_{c,n_a}\}, \mathcal{C}$

3.2. Cluster Assignment

Since the entire task set is partitioned into several subsets, the assignment of each subset needs to be determined, which is formulated as an integer linear programming (5). $y_{ij} = 1$ if robot i is assigned with cluster j and 0 otherwise. $w_{ij} \triangleq \|p_i - c_j\|_2^2 + \sum_{t \in \mathcal{T}_j} \|t - c_j\|_2^2$ defines the cost of cluster j being assigned to robot i , where the first term evaluates how far robot i is from cluster j and the second term estimates the cost of robot i visiting all the tasks within cluster j .

$$\min_{\mathbf{y}} \quad \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I}_{c,j}} w_{ij} y_{ij} \quad (5a)$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}} y_{ij} = 1, \forall j \in \mathcal{I}_{c,j}, \quad (5b)$$

$$\sum_{j \in \mathcal{I}_{c,j}} y_{ij} \leq 1, \forall i \in \mathcal{I}, \quad (5c)$$

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I}_{c,j}} y_{ij} = n_a, \quad (5d)$$

$$y_{ij} = \{0, 1\}, \forall (i, j) \in \mathcal{I} \times \mathcal{I}_{c,j}. \quad (5e)$$

Constraint (5b) ensures that each cluster must be assigned with one robot; (5c) guarantees that each robot can be at most assigned to one cluster; (5d) enforces no unassigned cluster being left. Constraint (5c) is compatible with a situation where the number of agents is greater than the number of nonempty clusters. This situation can happen at run-time when some tasks are completed. Note that when the number of clusters is not equal to n_a , the constraints (5c)–(5d) need to be revised accordingly. The cluster assignment problem (5) can be solved by some constrained integer linear programming solvers such as SCIP [31] and OR-Tools [32]. Denote $\hat{\mathcal{T}}_i$ as the task cluster assigned to robot i . Details about the cluster assignment are shown in Algorithm 2. An example is shown in Figure 2.

Algorithm 2: Cluster Assignment

Input: $\{\mathcal{T}_1, \dots, \mathcal{T}_{n_a}\}, \mathcal{X}, \mathcal{C}$
1 **for** robot $i = 1$ to n_a **do**
2 **for** cluster $j = 1$ to n_a **do**
3 $w_{ij} \leftarrow \|p_i - c_j\|_2^2 + \sum_{t \in \mathcal{T}_j} \|t - c_j\|_2^2$
4 $\mathbf{y}^* \leftarrow$ Solve (5) by a numerical solver
5 $\{\hat{\mathcal{T}}_1, \dots, \hat{\mathcal{T}}_{n_a}\} \leftarrow$ parse_result($\{\mathcal{T}_1, \dots, \mathcal{T}_k\}, \mathbf{y}^*$)
6 **return** $\{\hat{\mathcal{T}}_1, \dots, \hat{\mathcal{T}}_{n_a}\}$

3.3. Single-Robot Mission Planning

After each robot is assigned a task cluster, the task allocation orders and the collision-free paths need to be determined. This problem can be distributed to n_a robots parallelly and robot i solves its sub-problem by formulating it as a traveling salesperson problem (TSP), where the nodes are the robot itself and its assigned tasks. A path-finding algorithm generates collision-free paths for every pair of nodes and the length of these paths is the traveling cost from one node to another. Lazy Theta* [9] is used here due to fewer line-of-sight checks. This problem can be modeled as an integer linear program (6) with a Miller–Tucker–Zemlin (MTZ) formulation [33],

$$\min_{\mathbf{z}, \mathbf{u}} \sum_{i=1}^{n_m} \sum_{j=1}^{n_m} d_{ij} z_{ij} \quad (6a)$$

$$\text{s.t. } z_{ij} \in \{0, 1\}, \forall i, j = 1, \dots, n_m, \quad (6b)$$

$$u_i \in \mathbb{Z}, \forall i = 2, \dots, n_m, \quad (6c)$$

$$u_i - u_j + n_m z_{ij} \leq n_m - 1, 2 \leq i \neq j \leq n_m, \quad (6d)$$

$$1 \leq u_i \leq n_m - 1, \quad (6e)$$

$$\sum_{i=1}^{n_m} z_{ij} = 1, \forall j = 2, \dots, n_m, \quad (6f)$$

$$\sum_{j=1}^{n_m} z_{ij} = 1, \forall i = 1, \dots, n_m, \quad (6g)$$

$$\sum_{i=1}^{n_m} z_{i1} = 0, \quad (6h)$$

where $n_m \triangleq |\hat{\mathcal{T}}_i| + 1$ denotes the number of nodes; node 1 always indicates the robot's current position; $z_{ij} = 1$ if the robot goes from node i to node j , $z \in \{0, 1\}^{n_m^2}$; $\mathbf{u} \in \mathbb{Z}^{n_m-1}$ is a dummy variable to indicate tour ordering such that $u_i < u_j$ implies node i is visited before node j ; d_{ij} is the cost of the robot traveling from node i to node j , which is the length of the underlying collision-free path.

Constraints (6c)–(6e) guarantee only one tour covering all nodes [33]. Constraints (6f)–(6g) ensure that each node is visited from another node and from each node there is a departure to another node. Constraint (6h) indicates that the robot does not go back to its initial position after visiting all the tasks. (6h) can be changed if the robot needs to go back to a base. To ensure that there is no collision between robots, each robot considers the other robots as obstacles. Details are shown in Algorithm 3. An example is shown in Figure 2.

Algorithm 3: Parallel Multi-Robot Mission Planning

Input: $\{\hat{\mathcal{T}}_1, \dots, \hat{\mathcal{T}}_{n_a}\}, \mathcal{X}, \mathcal{O}$

- 1 // n_a robots parallelly execute the content in **parfor**
- 2 **parfor** robot $i = 1$ to n_a **do**
- 3 Initialize P_{lib} as empty
- 4 **for** $start, goal$ in $(\hat{\mathcal{T}}_i \cup \{p_i\})$ **do**
- 5 $\mathcal{O}_{now} \leftarrow \mathcal{O} \cup \mathcal{X} \setminus \{p_i\}$
- 6 $P_{start,goal} \leftarrow \text{path_finding}(start, goal, \mathcal{O}_{now})$
- 7 $P_{lib}.append(P_{start,goal})$
- 8 **for** node $i = 1$ to $1 + |\hat{\mathcal{T}}_i|$ **do**
- 9 **for** node $j = 1$ to $1 + |\hat{\mathcal{T}}_i|$ **do**
- 10 $P_{i,j} \leftarrow \text{load_path}(P_{lib}, i, j)$
- 11 $d_{ij} \leftarrow \text{compute_cost}(P_{i,j})$
- 12 $\mathbf{z}^*, \mathbf{u}^* \leftarrow \text{solve (6) by a numerical solver}$
- 13 $\mathcal{P}_i, \mathbf{r}_i \leftarrow \text{parse_path}(P_{lib}, \mathbf{z}^*, \mathbf{u}^*)$
- 14 **return** $\{\mathcal{P}_1, \dots, \mathcal{P}_{n_a}\}, \{\mathbf{r}_1, \dots, \mathbf{r}_{n_a}\}$

3.4. RPM at Run-Time

This subsection illustrates how RPM operates at run-time. First, RPM utilizes k-means++ [30] to initialize the cluster centroids. During the mission, the centroids from the previous iteration are the initial centroids for the next iteration. As some tasks are completed, the number of nonempty clusters n_c might be less than n_a . If $n_c < n_a$, one needs to remove the empty clusters and revise constraint (5d) as $\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I}_{c,j}} y_{ij} = n_c$. Note that all the constraints are compatible with the case where $n_c < n_a$. If there exist dynamic obstacles and tasks, RPM updates their information (positions) at each iteration. More details are shown in Algorithm 4.

Algorithm 4: RPM at Run-time

```

1 Initialize  $\mathcal{C}$  by k-means++ [30]
2 while  $\mathcal{T} \neq \emptyset$  do
3    $\mathcal{T} \leftarrow$  update task set
4    $\mathcal{X} \leftarrow$  update robot position
5    $\mathcal{O} \leftarrow$  update obstacle
6    $\{\mathcal{T}_1, \dots, \mathcal{T}_k\}, \mathcal{C} \leftarrow$  Algorithm 1 with previous  $\mathcal{C}$ 
7   remove empty task cluster
8    $\{\hat{\mathcal{T}}_1, \dots, \hat{\mathcal{T}}_{n_a}\} \leftarrow$  Algorithm 2
9    $\{\mathcal{P}_1, \dots, \mathcal{P}_{n_a}\}, \{r_1, \dots, r_{n_a}\} \leftarrow$  Algorithm 3
10  robots move one step along  $\{\mathcal{P}_1, \dots, \mathcal{P}_{n_a}\}$ 
11  time moves one step forward
12   $t_j \leftarrow$  current assigned task of robot  $i, \forall i \in \mathcal{I}$ 
13  delete task  $t_j$  if  $\|p_i - t_j\|_2 \leq \epsilon, \forall i \in \mathcal{I}$ 

```

4. Comparisons and Experiments

This section presents several experiments with static/dynamic obstacles/tasks and conducts scalability and optimality comparisons between RPM and a decentralized method [24]. From here on, CBBA is interchangeable with the method in [24] because it consists of CBBA and posterior path-finding. In addition, this section analyzes the computational burden for RPM and presents the optimality gap in small-size problems.

RPM is written in C++ and compiled as a Python library to be invoked. The integer programs in Algorithms 2 and 3 are solved by OR-Tools [32]. The C++ implementation utilizes multithreading as parallelization, i.e., the parfor in Line 2, Algorithm 3. First, a main thread, i.e., the central robot, runs Algorithms 1 and 2. Then, the results of Algorithm 2 are distributed to multiple robots/threads, where each thread runs Algorithm 3 parallelly for each robot. All the results are obtained by a computer with a 2.8 GHz Intel Core i7-7700HQ CPU and 16 GB memory. This implementation does not require a GPU but one can accelerate it with a GPU if needed.

4.1. Experiments

The test area is 6 m \times 5.6 m and the grid map size is 120 \times 112. RPM executes real-time mission planning for two Parrot Mambo quadrotors. In the experiments, each quadrotor follows the discrete paths returned from RPM. Then a low-level trajectory tracking controller (<https://github.com/zehuilu/Mambo-Tracking-Interface> (accessed on 25 February 2024)) broadcasts the desired control commands given the desired paths to each Mambo individually. Some details are explained in Figure 1. In the case of a dynamic task, a cone moves from one side to another side and RPM updates its planning result accordingly. Footage for these experiments is included in Video S1 of Supplementary Materials.

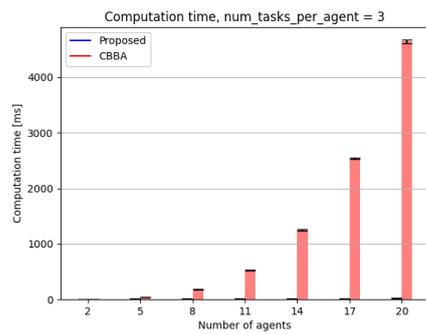
4.2. Comparison with Increased Number of Robots

Sections 4.2 and 4.3 show scalability comparisons between RPM and [24]. The grid map is 50 \times 50. Given a particular number of robots n_a and tasks n_t , there are 100 different

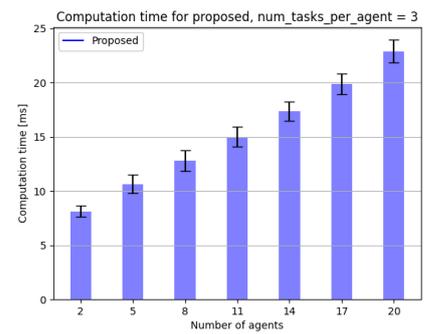
scenarios where the positions of robots and tasks are generated randomly. For each scenario, there are 200 randomly generated obstacles; each method runs 20 times, and the average computation time and total distance are collected.

Although [24] utilizes Dijkstra’s algorithm [5] as the path-finder, this paper replaces Dijkstra’s algorithm with Lazy Theta* [9] as the path-finder of [24] to present a fair comparison, regarding the computation. In other words, this paper eliminates the performance difference between the two path-finders although Lazy Theta* is faster, occupies less memory, and generates shorter paths due to any-angle movement.

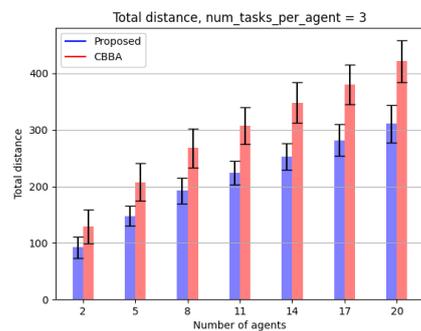
In Figure 3a,b, the computation time of [24] is increased exponentially and is up to over 4.5 s when there are 20 robots and 60 tasks, whereas the computation time of RPM is increased linearly, in the order of milliseconds. The fully centralized method [28] has a similar scenario with a 32×32 map and random obstacles. According to Figure 3 of [28], it takes about 55 s to generate sequences for 5–20 robots and 10–50 tasks. This paper omits the comparison with [28] because [28] is not a real-time algorithm. The CBBA’s computation time is increased exponentially because all robots need to take auctions iteratively and repeat for every task. The negotiation process for each task is more time-consuming and less efficient when n_a is larger, whereas, for RPM, the increased n_a only raises the burden of Algorithms 1 and 2 slightly. The most computationally heavy part of RPM is finding the collision-free path between every pair of nodes in each sub-problem, i.e., Line 4–Line 7 of Algorithm 3. Since Algorithm 3 is distributed over robots, the increased n_a does not raise the computational load significantly. Section 4.4 analyzes the computational burden of RPM and shows consistency with the comparisons.



(a) Computation time [ms] of the two methods



(b) Computation time [ms] of RPM



(c) Total distance of the two methods

Figure 3. The computation time and total distance results of two methods, RPM and [24], with an increased number of robots. The number of unassigned tasks is $n_t = 3n_a$. $N = 300$.

As for optimality (total distance), RPM outperforms [24] because RPM utilizes the global information of tasks and robots in Algorithms 1 and 2, while [24] performs an auction for one task at a time. Thus, the fully decentralized auction process does not utilize global information, resulting in less optimality. Moreover, the bid price in [24] is the Euclidean

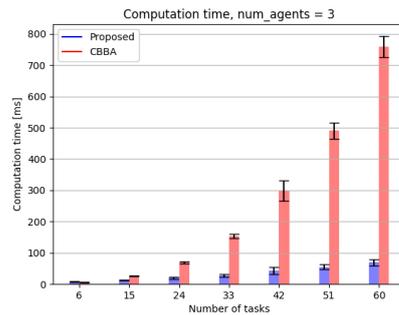
distance between robot and task, and [24] only generates collision-free paths after the task order is determined. In a cluttered environment, the Euclidean distance is not the actual cost. For completeness, Section 4.4 further analyzes and compares the computational burden if [24] utilizes collision-aware cost as the bid price.

On the other hand, as the number of robots increases, the optimality difference between the two methods is roughly the same. This observation is caused by the constant ratio between the numbers of unassigned tasks and robots. When this ratio is kept constant, for CBBA, the amount of negotiation between every two robots for every task is roughly the same as the number of robots increases.

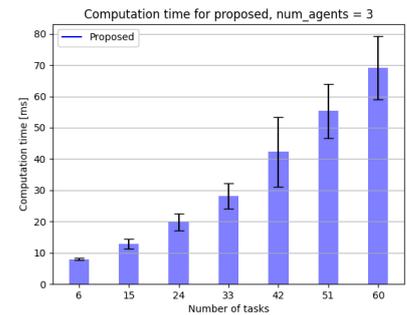
4.3. Comparison with Increased Number of Tasks

In Figure 4a,b, CBBA’s computation time is increased exponentially and is about 0.75 s for 60 tasks and 3 robots, whereas the computation time of RPM is increased almost linearly. The increasing rate of CBBA’s computation time in Figure 4a is much less than Figure 3a because there is less negotiation among robots and thus the auction for each task needs fewer iterations when n_a is smaller.

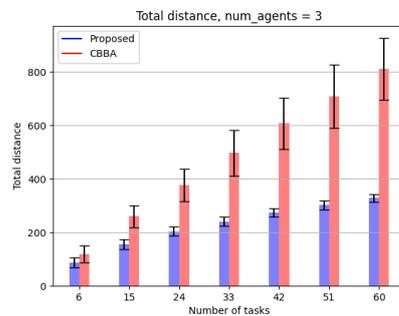
The increasing rate of RPM’s computation time in Figure 4b is greater than Figure 3b because the linearly increased n_t leads to the computational burden increasing quadratically (see Section 4.4). Nevertheless, the magnitude of computation time is still relatively small because each robot only needs to deal with a task subset due to Algorithm 1. The detailed analysis is shown in Section 4.4. Figure 4c shows that RPM outperforms CBBA regarding optimality. These comparisons show that, by using some global information in a parallel manner, RPM achieves better performance than a decentralized method and a centralized method.



(a) Computation time [ms] of the two methods



(b) Computation time [ms] of RPM



(c) Total distance of the two methods

Figure 4. The computation time and total distance results of two methods, RPM and [24], with an increased number of tasks. The number of robots is fixed at 3. $N = 300$.

The optimality gap between the two methods in Figure 4c increases as the number of tasks increases since the total traveling distance by CBBA is increased. This is because

the ratio between the number of unassigned tasks and robots is increasing, which leads to more negotiation between every two robots for every task.

4.4. Computational Burden Analysis

RPM approximates the traveling cost from one node to another by the length of the underlying collision-free path. An intuitive way to improve the optimality of CBBA is to utilize the lengths of collision-free paths as bid prices. This subsection analyzes the computational burden of RPM and this approach.

To find all possible paths, each robot connects to all the tasks and every two tasks connect. Thus, the total number of paths \hat{N}_p for CBBA is

$$\hat{N}_p = n_t P_2 + n_a \cdot n_t = n_t(n_t + n_a - 1), \quad (7)$$

where $n_t P_2 = \frac{n_t!}{(n_t-2)!}$ is the number of permutations for selecting two elements from a total of n_t elements.

As for RPM, the upper bound \bar{N}_p for the number of paths to be found for each robot is n_t , i.e.,

$$\bar{N}_p \triangleq \sup \max(|\hat{\mathcal{T}}_1|, \dots, |\hat{\mathcal{T}}_{n_a}|) = n_t. \quad (8)$$

Denote $\text{ceil}(\cdot) : \mathbb{R} \mapsto \mathbb{Z}$ as the ceiling function, and $\text{ceil}(x)$ as the least integer greater than or equal to x . Since the entire task set is partitioned into n_a subsets and the path-finding for each robot is parallel, the lower bound \underline{N}_p is

$$\underline{N}_p \triangleq \inf \max(|\hat{\mathcal{T}}_1|, \dots, |\hat{\mathcal{T}}_{n_a}|) \triangleq n_c = \text{ceil}(n_t/n_a). \quad (9)$$

Hence, the maximum number of paths N_p for RPM is

$$n_c + n_c P_2 \leq N_p \leq n_t + n_t P_2 \Rightarrow \left(\frac{n_t}{n_a}\right)^2 \lesssim N_p \leq n_t^2. \quad (10)$$

Combining with (7) yields

$$1 < 1 + \frac{n_a - 1}{n_t} \leq \frac{\hat{N}_p}{\bar{N}_p} \lesssim \left(1 + \frac{n_a - 1}{n_t}\right)n_a. \quad (11)$$

Since $(n_t/n_a)^2 \lesssim N_p \leq n_t^2$, when n_a is increased linearly and the ratio of n_t to n_a is a constant $a \triangleq n_t/n_a$, the lower bound of N_p increases linearly as $\underline{N}_p = a^2 n_a$. This conclusion is consistent with Figure 3b. Based on observation of comparisons, the actual computational burden of RPM is skewed towards the lower bound. When n_t is increased linearly and n_a is fixed, \underline{N}_p increases quadratically to n_t . In addition, the standard deviation of computation time in Figure 4b is increasingly larger than in Figure 3b. This observation appears because the number of assigned tasks for each robot $|\hat{\mathcal{T}}_1|, \dots, |\hat{\mathcal{T}}_{n_a}|$ tends to be more diverse as n_t increases and n_a is constant. As for Figure 3b, the task-robot ratio is fixed and thus the deviation remains relatively the same when n_a increases.

As for replacing the bid cost as the length of a collision-free path, the extra computational burden of CBBA is greater than the actual burden of RPM. The difference between the two upper bounds is $n_t(n_a - 1)$, which increases linearly as n_t or n_a increases. When the task-robot ratio is fixed and n_a increases, \hat{N}_p/\bar{N}_p is still greater than 1 and it increases with a rate of $1/n_a$. The upper bound \hat{N}_p/\bar{N}_p increases with a rate of n_a . When $n_a \gg n_t$, $n_a \cdot N_p \leq \hat{N}_p \lesssim n_a^2 \cdot N_p$. Therefore, the computation burden of CBBA is at least n_a times heavier than RPM and up to n_a^2 times heavier. When $n_a \ll n_t$, $N_p \leq \hat{N}_p \lesssim n_a \cdot N_p$. The worst case of RPM is that its computation burden is slightly less than CBBA's but CBBA's burden at most is n_a times greater than RPM's. Thus, task segmentation and parallelizable mission planning benefit run-time computation. Revising the bid prices of CBBA is not computationally efficient and, hence, the scalability is not good.

4.5. Optimality Gap in Small-Size Problems

Section 4.5 shows the optimality gap between RPM and the global optimum. The global optimum is found by exhaustive search and thus the search is only feasible in small-size problems. Figure 5 shows the optimality gap with two cases, 2 robots + 4 tasks and 3 robots + 6 tasks. For each case, there are 20 scenarios with different positions of robots, tasks, and obstacles. It is impossible to search a global optimum exhaustively for problems with a larger size since the MRMP problem is NP-hard. The total number of solutions for n_a robots and n_t tasks is $\frac{(n_t+2n_a-1)!n_t!}{(n_a+n_t)!(n_a-1)!}$. For the case with three robots and six tasks, there are 39,600 possible solutions and it takes about 10 s to find a global optimum. For four robots and eight tasks, there are 18,345,600 solutions and the estimated time to find an optimum is 78 min.

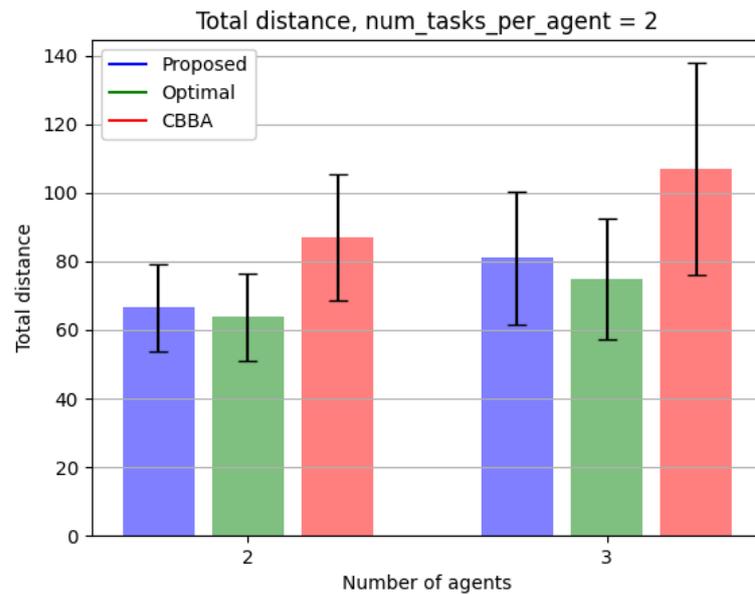


Figure 5. Optimality gap in small-size problems.

In the case of two robots, the optimality gap between RPM and the optimum is on average 4.3% while CBBA's cost is on average 36.3% greater than the optimum. As for another case, RPM's cost is on average 8.3% greater than the optimum whereas CBBA's cost is 43.1% greater than the optimum. The RPM's optimality gap increases when the problem size increases since the task segmentation algorithm cannot explore all the permutations of the number of assigned tasks for each robot. Nevertheless, the algorithm makes the MRMP problem tractable and solves it at run-time.

5. Conclusions

The collision-aware multi-robot mission planning problem is NP-hard but requires real-time computational performance in many applications. This paper presents a real-time parallel algorithm RPM, which partitions the entire task set into several subsets such that each robot can determine the task allocation order and collision-free path parallelly. This process reduces the dimension of the original problem and, hence, makes RPM able to run in real time with good scalability. The above results show that, by using global information in a parallel manner, RPM achieves better performance on both computation and optimality.

There are still numerous challenges in real-time multi-agent mission planning, such as enforcing endurance and capacity constraints, handling dynamic obstacles and tasks with intention prediction, introducing the heterogeneity of agents and tasks, etc. Also, designing distributed algorithms with good optimality and computational performance is another interesting direction.

Supplementary Materials: Video S1 (<https://youtu.be/bT5-EjS9rAk> (accessed on 26 February 2024)): a video includes all the experiments that are mentioned within this paper. Source Code S2 (<https://github.com/zehuilu/Real-time-Multi-Robot-Mission-Planning-in-Cluttered-Environment> (accessed on 26 February 2024)): a repository of all the source codes for the proposed algorithm RPM.

Author Contributions: Conceptualization, Z.L. and S.M.; methodology, Z.L.; software, Z.L. and T.Z.; validation, Z.L.; formal analysis, Z.L.; investigation, Z.L. and T.Z.; resources, S.M.; data curation, Z.L.; writing—original draft preparation, Z.L.; writing—review and editing, Z.L., T.Z. and S.M.; visualization, Z.L. and T.Z.; supervision, S.M.; project administration, S.M.; funding acquisition, S.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work research was funded by NASA University Leadership Initiative (ULI) under grant number 80NSSC20M0161 and Northrop Grumman Corporation.

Data Availability Statement: Data are contained within the article and Supplementary Materials.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analysis, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Herbert, S.L.; Chen, M.; Han, S.; Bansal, S.; Fisac, J.F.; Tomlin, C.J. FaSTrack: A modular framework for fast and guaranteed safe motion planning. In Proceedings of the 2017 IEEE 56th Annual Conference on Decision and Control (CDC), IEEE, Melbourne, Australia, 12–15 December 2017; pp. 1517–1522.
2. Kousik, S.; Holmes, P.; Vasudevan, R. Safe, aggressive quadrotor flight via reachability-based trajectory design. In Proceedings of the ASME 2019 Dynamic Systems and Control Conference, American Society of Mechanical Engineers Digital Collection, Park City, UT, USA, 8–11 October 2019.
3. Tordesillas, J.; Lopez, B.T.; How, J.P. Faster: Fast and safe trajectory planner for flights in unknown environments. In Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, Macau, China, 4–8 November 2019; pp. 1934–1940.
4. Danielson, C.; Berntorp, K.; Weiss, A.; Di Cairano, S. Robust motion planning for uncertain systems with disturbances using the invariant-set motion planner. *IEEE Trans. Autom. Control* **2020**, *65*, 4456–4463. [[CrossRef](#)]
5. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [[CrossRef](#)]
6. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [[CrossRef](#)]
7. LaValle, S. Rapidly-exploring random trees: A new tool for path planning. *Res. Rep. 9811* **1998**.
8. Daniel, K.; Nash, A.; Koenig, S.; Felner, A. Theta*: Any-angle path planning on grids. *J. Artif. Intell. Res.* **2010**, *39*, 533–579. [[CrossRef](#)]
9. Nash, A.; Koenig, S.; Tovey, C. Lazy Theta*: Any-angle path planning and path length analysis in 3D. In Proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence, Atlanta, GA, USA, 11–15 July 2010; Volume 24, pp. 147–154.
10. Gerkey, B.P.; Mataric, M.J. A formal analysis and taxonomy of task allocation in multi-robot systems. *Int. J. Robot. Res.* **2004**, *23*, 939–954. [[CrossRef](#)]
11. Michael, N.; Zavlanos, M.M.; Kumar, V.; Pappas, G.J. Distributed multi-robot task assignment and formation control. In Proceedings of the 2008 IEEE International Conference on Robotics and Automation, IEEE, Pasadena, CA, USA, 19–23 May 2008; pp. 128–133.
12. Choi, H.L.; Brunet, L.; How, J.P. Consensus-based decentralized auctions for robust task allocation. *IEEE Trans. Robot.* **2009**, *25*, 912–926. [[CrossRef](#)]
13. Wang, X.; Hudack, J.; Mou, S. Distributed Algorithm with Resilience for Multi-Agent Task Allocation. In Proceedings of the 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS), IEEE, Victoria, BC, Canada, 10–13 May 2021; pp. 112–117.
14. Nunes, E.; McIntire, M.; Gini, M. Decentralized multi-robot allocation of tasks with temporal and precedence constraints. *Adv. Robot.* **2017**, *31*, 1193–1207. [[CrossRef](#)]
15. Tadewos, T.G.; Shamgah, L.; Karimodini, A. On-the-fly decentralized tasking of autonomous vehicles. In Proceedings of the 2019 IEEE 58th Conference on Decision and Control (CDC), IEEE, Nice, France, 11–13 December 2019; pp. 2770–2775.
16. Kim, K.S.; Kim, H.Y.; Choi, H.L. Minimizing communications in decentralized greedy task allocation. *J. Aerosp. Inf. Syst.* **2019**, *16*, 340–345. [[CrossRef](#)]
17. Patel, R.; Rudnick-Cohen, E.; Azarm, S.; Otte, M.; Xu, H.; Herrmann, J.W. Decentralized Task Allocation in Multi-Agent Systems Using a Decentralized Genetic Algorithm. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, Paris, France, 31 May–31 August 2020; pp. 3770–3776.

18. Banks, C.; Wilson, S.; Coogan, S.; Egerstedt, M. Multi-agent task allocation using cross-entropy temporal logic optimization. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, Paris, France, 31 May–31 August 2020; pp. 7712–7718.
19. Henkel, C.; Toussaint, M. Optimized directed roadmap graph for multi-agent path finding using stochastic gradient descent. In Proceedings of the Proceedings of the 35th Annual ACM Symposium on Applied Computing, New York, NY, USA, 30 March–3 April 2020; pp. 776–783.
20. Prasad, A.; Choi, H.L.; Sundaram, S. Min-Max Tours and Paths for Task Allocation to Heterogeneous Agents. *IEEE Trans. Control Netw. Syst.* **2020**, *7*, 1511–1522. [[CrossRef](#)]
21. Park, B.; Kang, C.; Choi, J. Cooperative multi-robot task allocation with reinforcement learning. *Appl. Sci.* **2021**, *12*, 272. [[CrossRef](#)]
22. Bakolas, E.; Lee, Y. Decentralized game-theoretic control for dynamic task allocation problems for multi-agent systems. In Proceedings of the 2021 American Control Conference (ACC), IEEE, New Orleans, LA, USA, 25–28 May 2021; pp. 3228–3233.
23. Martin, J.G.; Muros, F.J.; Maestre, J.M.; Camacho, E.F. Multi-robot task allocation clustering based on game theory. *Robot. Auton. Syst.* **2023**, *161*, 104314. [[CrossRef](#)]
24. Bertucelli, L.; Choi, H.L.; Cho, P.; How, J. Real-time multi-UAV task assignment in dynamic and uncertain environments. In Proceedings of the AIAA Guidance, Navigation, and Control Conference, Chicago, IL, USA, 10–13 August 2009; p. 5776.
25. Choi, H.J.; Kim, Y.D.; Kim, H.J. Genetic algorithm based decentralized task assignment for multiple unmanned aerial vehicles in dynamic environments. *Int. J. Aeronaut. Space Sci.* **2011**, *12*, 163–174. [[CrossRef](#)]
26. Henkel, C.; Abbenseth, J.; Toussaint, M. An optimal algorithm to solve the combined task allocation and path finding problem. In Proceedings of the 2019 IEEE/RISJ International Conference on Intelligent Robots and Systems (IROS), IEEE, Macau, China, 4–8 November 2019; pp. 4140–4146.
27. Schillinger, P.; Bürger, M.; Dimarogonas, D.V. Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. *Int. J. Robot. Res.* **2018**, *37*, 818–838. [[CrossRef](#)]
28. Ren, Z.; Rathinam, S.; Choset, H. MS: A New Exact Algorithm for Multi-agent Simultaneous Multi-goal Sequencing and Path Finding. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, Xian, China, 30 May–5 June 2021; pp. 11560–11565.
29. Lloyd, S. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *28*, 129–137. [[CrossRef](#)]
30. Arthur, D.; Vassilvitskii, S. k-means++: The advantages of careful seeding. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (SODA), SIAM, Miami, FL, USA, 22–26 January 2006; pp. 1027–1035.
31. Achterberg, T. SCIP: Solving constraint integer programs. *Math. Program. Comput.* **2009**, *1*, 1–41. [[CrossRef](#)]
32. Google. OR-Tools. 2010. Available online: <https://developers.google.com/optimization> (accessed on 25 February 2024).
33. Miller, C.E.; Tucker, A.W.; Zemlin, R.A. Integer programming formulation of traveling salesman problems. *J. ACM (JACM)* **1960**, *7*, 326–329. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.