Multi-robot Task Allocation in the Environment with Functional Tasks

Fuhan Yan[1*] , **Kai Di**[2*]
[1]College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing, China.
[2]School of computer science and engineering, Southeast University, Nanjing, China.
yanfh@cqupt.edu.cn; dikai@seu.edu.cn.

Abstract

Multi-robot task allocation (MRTA) problem has long been a key issue in multi-robot systems. Previous studies usually assumed that the robots must complete all tasks with minimum time cost. However, in many real situations, some tasks can be selectively performed by robots and will not limit the achievement of the goal. Instead, completing these tasks will cause some functional effects, such as decreasing the time cost of completing other tasks. This kind of task can be called "functional task". This paper studies the multi-robot task allocation in the environment with functional tasks. In the problem, neither allocating all functional tasks nor allocating no functional task is always optimal. Previous algorithms usually allocate all tasks and cannot suitably select the functional tasks. Because of the interaction and sequential influence, the total effects of the functional tasks are too complex to exactly calculate. We fully analyze this problem and then design a heuristic algorithm. The heuristic algorithm scores the functional tasks referring to linear threshold model (used to analyze the sequential influence of a functional task). The simulated experiments demonstrate that the heuristic algorithm can outperform the benchmark algorithms.

1 Introduction

Multi-robot task allocation (MRTA) is one of the most crucial problems in multi-robot systems and is also fundamental for many other robotic problems such as search, rescue, and patrol. Therefore, multi-robot task allocation problems have been widely studied in previous works [Wei *et al.*, 2020; Koenig *et al.*, 2008; Sullivan *et al.*, 2019; Suslova and Fazli, 2020; Yan *et al.*, 2019]. In previous studies of MRTA, it is usually assumed that all the tasks must be completed, and the allocation algorithms allocate all these tasks to robots. However, in many real situations, some tasks can be selectively performed by robots and will not limit the achievement of the goal. In other words, even if the robots perform none of these tasks, the robots can also achieve the goal. Nevertheless, after these tasks are completed, some effects can be generated. This kind of task can be called "functional task". Meanwhile, the tasks that must be completed by robots can be called "compulsory task". The motivation is described as the following example.

- **Multi-robot real-time information collection**: Some robots need to collect the real-time information of some locations. A robot needs to visit a target location and then upload the real-time information (obtained by camera or other sensors) to the base by wireless communication. Uploading the real-time information of the target locations are the "compulsory tasks", and the optimization objective is to minimize the time cost. In this case, the robots can selectively set up the simple communication enhancement facilities (such as deployable antenna). Then, the robots can upload the data faster. These are the "functional tasks". If the data size of a target location is quite small, the robot needs not to perform functional task. If the data size is quite large, the robot can save more time than the time cost of setting up a simple communication enhancement facility.

This paper studies the multi-robot task allocation with functional tasks. In most cases, neither allocating all functional tasks nor allocating no functional task is always optimal. Previous algorithms cannot suitably allocate the functional tasks and may lead to large time costs. We fully analyze this problem and design a heuristic algorithm. The functional tasks are firstly scored respectively based on linear threshold model. According to the individual scores, the combination of functional tasks is selected. The robots allocate the selected functional tasks, and then allocate the compulsory tasks assuming the selected functional tasks have been completed.

The simulated experiments are presented to test the performance of the proposed algorithm. The experimental data demonstrate that the proposed heuristic algorithm can outperform the benchmark algorithms when functional tasks exist in the environment.

The rest of this paper is organized as follows. In Section 2, we briefly introduce the related works. Section 3 presents the problem formulation. Section 4 shows the problem analyses and benchmark algorithms. The heuristic algorithm is shown in Section 5. The experiments are provided in Section 6. Finally, we conclude this paper in Section 7.

[*]Co-corresponding Authors

2 Related Works

2.1 Multi-robot Task Allocation Problems

The multi-robot task allocation (MRTA) problem is one of the core issues in multi-robot systems. In some studies, the robots just need to visit various locations that represent tasks, and a task is completed after being visited [Wei *et al.*, 2020; Koenig *et al.*, 2008; Sullivan *et al.*, 2019]. Besides, some studies analyze this kind of model with some extra complex restrictions, such as time windows [Suslova and Fazli, 2020], heterogeneous skill requirements [Emam *et al.*, 2020], etc.

Some studies analyze the factors related to the performing processes (completing task is not just visiting). For instance, a single robot is not able to complete a task because the skill requirement set of this task is a superset of the skill set of the robot [Lee, 2018]. In this case, several robots need to perform a task simultaneously. Besides, some studies analyze the multi-robot task allocation with precedence constrained (PC-MRTA) [McIntire *et al.*, 2016; Bischoff *et al.*, 2020]. Some tasks cannot be performed until certain tasks are completed.

2.2 The Classical Algorithms Used in MRTA

The time complexity of the algorithm exactly solving MRTA is not acceptable. Even if the tasks are completed after being visited, MRTA is still np-hard [McIntire *et al.*, 2016]. Therefore, most allocation algorithms are approximate or heuristic.

Many metaheuristics can be used in MRTA, and they are problem independent frameworks that can also be used in other optimization problems. The metaheuristics that have been used for task allocation include Genetic Algorithms [Yuan *et al.*, 2013], Simulated Annealing [Behnck *et al.*, 2015], and Ant Colony Optimisation [Chen *et al.*, 2018]. They can be tuned to converge within certain time limits. However, when the solution space is quite large or there are some complex restrictions, they may converge slowly.

The auction-based algorithms are usually faster and widely used to allocate tasks. Among these, the most popular frame is the sequential single-item (SSI) auction [Koenig *et al.*, 2008; Sullivan *et al.*, 2019; McIntire *et al.*, 2016]. Although there are many auction-based algorithms, they cannot effectively solve the proposed problem. The reason is that functional tasks are optional, and which functional tasks should be allocated has not been analyzed. Nevertheless, SSI can still be selected as an important basic technology in this paper. Then, we concretely introduce the version of SSI auction [Koenig *et al.*, 2008; Sullivan *et al.*, 2019] that will be used in this paper:

- **Sequential Single-Item (SSI) Auction**. During each round of SSI, all robots bid on all unassigned tasks and the auctioneer then assigns one additional (previously unassigned) task to robots. The bidding rule is: each robot tries to insert the task being bid on into the task sequence consisting of the currently allocated tasks to it, and the bid is the minimum time cost of the new sequence. The winner determination rule is: allocates the task which has the largest difference between the lowest bid and second lowest bid (in the event of a tie, the task with the lowest bid is assigned).

3 Problem Formalization

This paper studies the multi-robot task allocation with functional tasks (FT-MRTA). In this section, we first introduce the basic settings of MRTA. Then, the functional tasks and optimization objective are introduced.

3.1 Basic Settings

- **Environment**: the environment is modeled as a two-dimensional grid world, and the size is $X \times Y$. Some robots are initially located at a waiting area that is at the center of the world. Let $R = \{r_1, r_2, ...\}$ denote the set of robots. There can be more than one robot in a same cell of the grid world. Some tasks are distributed in the world, and let $T = \{t_1, t_2, ...\}$ denote the set of tasks. There can be at most one task in a cell of the grid world. Let $d(t_i, t_j)$ denote the Manhattan distance between task t_i and task t_j, and let $d(r_i, t_j)$ denote the Manhattan distance between robot r_i and task t_j.
- **Robots**: all the robots are initially at the center of the world, i.e., $(\frac{X}{2}, \frac{Y}{2})$. Let $d(O, t_j)$ denote the Manhattan distance between $(\frac{X}{2}, \frac{Y}{2})$ and task t_j. In each time step, each robot can move up, down, left or right to the adjacent cell. Certainly, the robot can also stay in a cell.
- **Tasks**: each task $t_i \in T$ is associated with a requirement parameter $\nu_i \in (0, +\infty)$ that determines how hard to complete t_i. A task can be performed only by one robot. When a robot and a task are in the same cell, the robot can begin to perform the task. If the robot begins to perform, it cannot stop until completing this task. When performing task t_i, the robot can make $\nu_i \leftarrow \max(0, \nu_i - 1)$ in each time step. When $\nu_i = 0$, task t_i is completed. Therefore, the time cost of completing t_i is $\lceil \nu_i \rceil$.

3.2 Compulsory Tasks and Functional Tasks

The tasks can be divided into two sub-sets. Let T_c denote the set of compulsory tasks and T_f denote the set of functional tasks. $|T| = |T_c| + |T_f|$. For the sake of simplification, it is assumed that $T_c = \{t_1, t_2, ..., t_{|T_c|}\}$ and $T_f = \{t_{1+|T_c|}, t_{2+|T_c|}, ..., t_{|T|}\}$.

- **Compulsory tasks**: task $t_i (1 \leq i \leq |T_c|)$ must be completed and can directly limit the achievement of the goal.
- **Functional Tasks**: if a functional task is completed, the time cost of completing other tasks will be decreased. Besides, each functional task t_i is associated with an influence parameter $f_i \in (1, +\infty)$ and can only influence the tasks in a limited range. Only the functional tasks that are completed before t_j begins to be performed can influence t_j. Therefore, let $t_i << t_j$ denote that t_i can be completed before t_j begins to be performed.

Overall, after some functional tasks are completed, the time cost of completing task t_j is $\lceil g(T_f^j)\nu_j \rceil$. Here, $T_f^j = \{t_k | t_k \in T_f, d(t_k, t_j) \leq \varepsilon, t_k << t_j\}$. The parameter ε denotes the influence range. Without loss of generality, $g(x)$ can be set as different functions while $g(x) \in (0, 1]$ and $g(\emptyset) = 1$ should be satisfied. For instance, $g(T_f^j) = \frac{1}{1+ln \prod_{t_k \in T_f^j} f_k}$.

3.3 Optimization Objective

The robots must complete all the tasks in set T_c. The optimization objective is the time cost of completing all the tasks in set T_c. Let $S_i = \{t_{i_1}, t_{i_2}, ..., t_{i_w}\}$ denote the task sequence allocated to robot r_i. $S_i \cap S_j = \emptyset$ for any r_i, $r_j \in R$. If $\cup_{r_i \in R} S_i = T_c$, the time cost for robot r_i to complete the tasks in S_i is

$$C(S_i) = d(O, t_{i_1}) + \sum_{1 \le k \le w-1} (\lceil \nu_{i_k} \rceil + d(t_{i_k}, t_{i_{k+1}})) + \lceil \nu_{i_w} \rceil. \tag{1}$$

As the robots take one time step to move to the neighbouring cell, $d(..., ...)$ can also denote the moving time.

If the robots select to perform some tasks in T_f, the sequence S_i needs to satisfy $T_c \subseteq \cup_{r_i \in R} S_i$. As the functional tasks can influence the time cost of completing other tasks, the time cost for robot r_i to complete the tasks in S_i becomes:

$$C'(S_i, \cup_{r_j \in R} S_j) = d(O, t_{i_1}) + \sum_{1 \le k \le w-1} (\lceil \nu_{i_k} g(T_f^{i_k}) \rceil + d(t_{i_k}, t_{i_{k+1}})) + \lceil \nu_{i_w} g(T_f^{i_w}) \rceil. \tag{2}$$

Thus, the final objective of FT-MRTA is defined as:

$$\text{minimize } \max_{r_i \in R} C'(S_i, \cup_{r_j \in R} S_j),$$

subject to:

$$\begin{gathered} S_i \cap S_j = \emptyset, \forall r_i, r_j \in R; \\ S_i = \{t_{i_1}, t_{i_2}, ..., t_{i_w}\}; \\ T_c \subseteq \cup_{r_j \in R} S_j; \\ g(x) \in (0, 1], g(\emptyset) = 1; \\ T_f^j = \{t_k | t_k \in T_f, d(t_k, t_j) \le \varepsilon, t_k << t_j\}; \\ T_f^j \subseteq \cup_{r_j \in R} S_j. \end{gathered}$$

4 Problem Analyses and Benchmark Algorithms

The essential objective of the robots is to complete all compulsory tasks. All the compulsory tasks can be allocated based on the SSI algorithm (See also section 2.2). The first baseline can be set as allocating all compulsory tasks without any functional task based on the SSI algorithm. Nevertheless, if the effect of functional tasks is large enough, this baseline is easy to outperform. We present a simple case in figure 1. In figure 1, $|R| = 1$, $|T_c| = 1$, $|T_f| = 1$ and $\varepsilon = 10$. The difference between figure 1 (a) and (b) is ν_2. In this simple case, the effect of functional task t_2 is easy to analyze:

- In (a), if robot r_1 only performs the compulsory task t_1, the time cost is $d(r_1, t_1) + \nu_1 = 10 + 50 = 60$. This is the baseline. If robot r_1 first performs t_2 and then performs t_1, the time cost is $d(r_1, t_1) + \nu_2 + d(t_1, t_2) + \nu_1 g(\{t_2\}) = 5 + 5 + 5 + 25 = 40$. Therefore, allocating functional task t_2 is better.
- In (b), if allocating t_2, the time cost is $5 + 40 + 5 + 25 = 75$. Ignoring functional task t_2 becomes better.

In the above case, it is easy to know whether the robot should allocate the functional task. However, if $|R|$, $|T_c|$ and $|T_f|$ become quite large, exactly analyze each functional task is infeasible. In this case, the greedy algorithm frame can be used. The robots firstly allocate all compulsory tasks based on the SSI algorithm and obtain the task sequences. Then, the robots try to insert the functional tasks into the sequences and select the best location greedily. This method is inspired by [McIntire *et al.*, 2016]. Figure 2 shows this process:

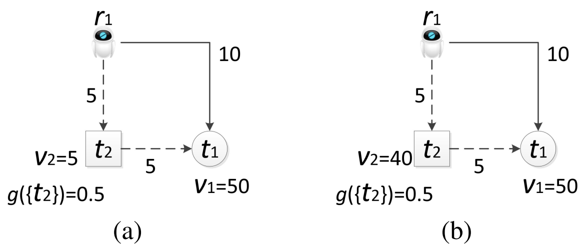

Figure 1: Case study one.

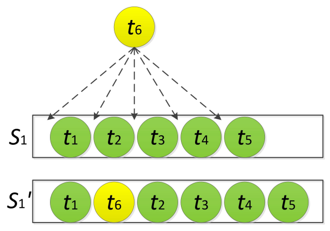

Figure 2: Case study two.

- $|R| = 1$, $|T_c| = 5$ and $|T_f| = 1$. Robot r_1 obtains sequence S_1 based on SSI algorithm. Then, r_1 tries to insert t_6 into S_1 at different locations. For instance, r_1 inserts t_6 after t_1 to get S'_1. If $C'(S'_1, (\cup_{r_j \in R \setminus r_1} S_j) \cup \{S'_1\}) < C'(S_1, \cup_{r_j \in R} S_j)$, r_1 updates the task sequence (S_1 is replaced by S'_1). Finally, t_6 will be inserted at the best location or not be inserted.

When $|R| > 1$ and $|T_f| > 1$, two greedy allocation algorithms can be presented. One is a fast version while the other is a slow version. In the fast version, $C'(S_i, \cup_{r_j \in R} S_j)$ is replaced by a easily estimated value because calculating $C'(S_i, \cup_{r_j \in R} S_j)$ is time-consuming. In detail, $EC(S_i)$ is the estimated time cost that can be calculated easily:

$$EC(S_i) = d(O, t_{i_1}) + \sum_{1 \le k \le w-1} (\lceil \nu_{i_k} g(T_f^{i_k} \cap S_i) \rceil + d(t_{i_k}, t_{i_{k+1}})) + \lceil \nu_{i_w} g(T_f^{i_w} \cap S_i) \rceil \tag{3}$$

Here, $EC(S_i)$ is calculated only based on S_i without considering other sequences. Then, if $r_j = \text{argmax}_{r_i \in R} EC(S_i)$, the functional tasks will be inserted in S_j. The fast greedy allocation algorithm is shown in Algorithm 1. In Algorithm 1, line 2 selects S_u that is associated with largest $EC(S_i)$. Line 4-8 select the functional tasks that may influence S_u. Line 9-16 try to insert the functional tasks into S_u.

The time complexity of calculating $EC(S_i)$ is about $O(\frac{|T_c|}{|R|})$. The time complexity of analyzing all positions is about $O(\frac{|T_c|}{|R|} \times \frac{|T_c|}{|R|})$. Then, the worst time complexity of

Algorithm 1 Greedy task allocation (fast)

```
Input: Task sequences $S_1, S_2 \dots S_{|R|}$ based on SSI
Output: $S_1, S_2 \dots S_{|R|}$
1: while True do
2:     $r_u \leftarrow \text{argmax}_{r_i \in R} EC(S_i)$ (See also equation 3)
3:     $T_r \leftarrow \emptyset, b \leftarrow EC(S_u), p_{best} \leftarrow \emptyset, t_{best} \leftarrow \emptyset$
4:     for all $t_i \in T_f$ do
5:         if $\exists t_j \in S_u$ satisfies $d(t_i, t_j) \leq \varepsilon$ and $i \neq j$ then
6:             $T_r \leftarrow T_r \cap \{t_i\}$.
7:         end if
8:     end for
9:     for all $t_i \in T_r$ do
10:        for all positions $p$ in current schedule $S_u$ do
11:            Insert $t_i$ in $S_u$ at $p$ to get $S'_u$
12:            if $EC(S'_u) < b$ then
13:                $b \leftarrow EC(S'_u), p_{best} \leftarrow p, t_{best} \leftarrow t_i$
14:            end if
15:        end for
16:    end for
17:    if $p_{best} \neq \emptyset$ then
18:        Insert $t_{best}$ in $S_u$ at best position $p_{best}$
19:        $T_f \leftarrow T_f \backslash t_{best}$
20:    else
21:        return $S_1, S_2 \dots S_{|R|}$
22:    end if
23: end while
```

Algorithm 2 Greedy task allocation (slow)

```
Input: Task sequences $S_1, S_2 \dots S_{|R|}$ based on SSI
Output: $S_1, S_2 \dots S_{|R|}$
1: while True do
2:     $b \leftarrow \max_{r_i \in R} C'(S_i, \cup_{r_j \in R} S_j)$ (See also equation 2)
3:     $p_{best} \leftarrow \emptyset, t_{best} \leftarrow \emptyset, u \leftarrow \emptyset$
4:     for all $t_i \in T_f$ do
5:         for all $S_j \in \{S_1, S_2 ... S_{|R|}\}$ do
6:             for all positions $p$ in current schedule $S_j$ do
7:                 Insert $t_i$ in $S_j$ at $p$ to get $S'_j$
8:                 Estimate time cost of each sequence as $c_k$ for
                   all $r_k \in R$ (See also equation 4)
9:                 $b' \leftarrow \max_{r_k \in R} c_k$
10:                if $b' < b$ then
11:                    $b \leftarrow b', p_{best} \leftarrow p, u \leftarrow j, t_{best} \leftarrow t_i$
12:                end if
13:            end for
14:        end for
15:    end for
16:    if $p_{best} \neq \emptyset$ then
17:        Insert $t_{best}$ in $S_u$ at best position $p_{best}$
18:        $T_f \leftarrow T_f \backslash t_{best}$
19:    else
20:        return $S_1, S_2 \dots S_{|R|}$
21:    end if
22: end while
```

Algorithm 1 is $O(|T_f| \times |T_f| \times \frac{|T_c|}{|R|} \times \frac{|T_c|}{|R|}) = O(\frac{|T_f|^2 |T_c|^2}{|R|^2})$.

In the slow version, all the sequences and functional tasks are analyzed iteratively. The slow greedy allocation algorithm is shown in Algorithm 2. In line 7 of Algorithm 2, t_i is inserted in S_j at p to get S'_j. Then, in line 8,

$$c_k = \begin{cases} C'(S'_j, (\cup_{r_k \in R \backslash r_j} S_k) \cup \{S'_j\}), & k = j; \\ C'(S_k, (\cup_{r_k \in R \backslash r_j} S_k) \cup \{S'_j\}), & k \neq j. \end{cases} \tag{4}$$

When a functional task is inserted into any sequence, c_k of all robots will be updated. Then, if the expected maximum time cost decreases, the corresponding parameters are recorded. In line 17, the recorded functional task is truly inserted into the recorded task sequence. The time complexity of calculating c_k for all sequences is $O(|R| \times |T_c|)$. Then, the worst time complexity of Algorithm 2 is $O(|T_f| \times |T_f| \times |R| \times \frac{|T_c|}{|R|} \times |R| \times |T_c|) = O(|T_f|^2 |T_c|^2 |R|)$. It means that Algorithm 2 runs much more slowly than Algorithm 1. Therefore, Algorithm 2 runs too slowly to be practical, but can be used as a benchmark algorithm to measure other algorithms.

5 The Heuristic Algorithm based on Scoring the Functional Tasks

In Algorithm 2, each functional task tries to be inserted in each task sequence. The robot causing maximum time cost can be influenced by the functional task completed by another robot. Thus, Algorithm 2 can make the maximum time cost decrease more than Algorithm 1. Nevertheless, the time complexity of Algorithm 2 is unacceptable. In this case, we try to design a heuristic algorithm, which simultaneously satisfies: 1) runs fast; 2) the functional task can also be inserted into any other task sequence while influencing the task sequence associated with the maximum time cost.

5.1 Scoring the Functional Tasks

In the heuristic algorithm, the first step is scoring all functional tasks. The direct effect of a functional task on compulsory tasks can be easily calculated. However, there may be an indirect effect of a functional task. Figure 3 shows an instance:

- In figure 3, $|R| = 1, |T_c| = 1, |T_f| = 2$ and $\varepsilon = 10$. It is assumed that robot r_1 is initially at the position of task t_2 but has not yet perform t_2. If r_1 only performs t_1, the time cost is $8 + 8 + 50 = 66$. This is the baseline. If r_1 selects task sequence $\{t_3, t_1\}$, the time cost is $8 + 50 + 8 + 0.5 \times 50 = 91$. If r_1 selects task sequence $\{t_2, t_1\}$, the time cost is $5 + 8 + 8 + 50 = 71$. Here, t_2 cannot directly influence t_1 because $d(t_2, t_1) > \varepsilon$. Neither $\{t_3, t_1\}$ nor $\{t_2, t_1\}$ is suitable. However, if r_1 selects task sequence $\{t_2, t_3, t_1\}$, the time cost is $5 + 8 + 0.2 \times 50 + 8 + 0.5 \times 50 = 56$. In this case, t_2 directly influences t_3 and then t_3 influences t_1.

Based on the above instance, it can be known: 1) the indirect effect of each functional task should not be ignored; 2) only if the subsequent functional task is completed, the preceding functional task can indirectly influence the compulsory tasks out of range. This sequential influence process is quite similar to the influence diffusion models in social networks. Therefore, when $|R|$, $|T_c|$ and $|T_f|$ become large, the functional tasks can be scored referring to the influence dif-

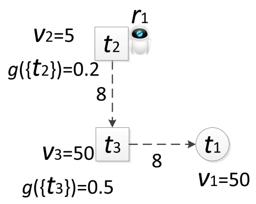

Figure 3: Case study three.

Algorithm 3 Scoring algorithm

Input: Active task set T_a
Output: $Score$

```
1: $Score \leftarrow 0, n \leftarrow 0$
2: Inactive task set $T_{ina} = T \backslash T_a$
3: while True do
4:     for all $t_i \in T_{ina}$ do
5:         $EG_i = g(\{t_k | t_k \in T_a, t_k \in T_f, d(t_k, t_i) \leq \varepsilon\})$
6:     end for
7:     for all $t_i \in T_{ina}$ do
8:         if $EG_i < TS$ then
9:             Move $t_i : T_{ina} \rightarrow T_a$
10:            if $t_i \in T_c$ then
11:                $Score \leftarrow Score + \gamma^n \nu_i (1 - EG_i)$
12:            end if
13:        end if
14:    end for
15:    $n \leftarrow n + 1$
16:    if $T_{ina}$ and $T_a$ do not change in this loop then
17:        return $Score$
18:    end if
19: end while
```

fusion models.

Influence diffusion models in social networks have been widely studied and are usually used to model the diffusion of virus, information, opinion, etc [Yan *et al.*, 2014; Yan *et al.*, 2016; Yang *et al.*, 2019]. In these models, some nodes in the network are initially active while others are inactive. Then, the inactive nodes may be activated according to some rules. The two most classical models are the independent cascade model [Yan *et al.*, 2014; Yan *et al.*, 2016] and linear threshold model [Yang *et al.*, 2019]. Most influence diffusion models are variants of the two classical models. In the independent cascade model, active nodes try to independently activate other nodes with some probabilities. In the linear threshold model, a node is activated if the (weighted) number of active neighbors exceeds a threshold. If the thresholds are deterministic, the linear threshold model is deterministic. Therefore, we select the linear threshold model to score the functional tasks.

Algorithm 3 shows this process. In line 5, EG_i is used to quantize the influence of the active functional tasks. Smaller EG_i means larger influence. In line 8, TS is the threshold of the linear threshold model. In line 11, γ is a discount parameter and is used to decrease the score of indirect influence. The score is the decrement of the time cost of completing compulsory tasks. In line 16-17, if there is no new activated node, the

Algorithm 4 Task set selecting algorithm

Input: Task set T_c and T_f
Output: Task set T_s

```
1: for all $t_i \in T_f$ do
2:     $Score_i = \frac{Scoring(t_i)}{v_i + d(O, t_i)}$ (See also Algorithm 3)
3: end for
4: Sorting the scores of each functional task from largest to
   smallest to get $Score_{o_1}, Score_{o_2} ... Score_{o_{|T_f|}}$
5: $T_s = \{t_{o_1}\}, Cost \leftarrow \nu_{o_1} + d(O, t_{o_1})$
6: if $Score_{o_1} > 1$ then
7:     for $k \leftarrow o_2$ to $o_{|T_f|}$ do
8:         $Score_{best} \leftarrow$Scoring$(T_s)$
9:         if Scoring$(T_s \cup \{t_k\}) - Score_{best}$ >max$(0, \nu_k +
           d(O, t_k) - Cost)$ then
10:            $T_s \leftarrow T_s \cup \{t_k\}, Cost \leftarrow \frac{Cost + v_k + d(O, t_k)}{2}$
11:        end if
12:    end for
13:    if $|T_s| < |R|$ then
14:        for $k \leftarrow o_2$ to $o_{|T_f|}$ & $|T_s| < |R|$ do
15:            if $Score_k > 1$ then
16:                $T_s \leftarrow T_s \cup \{t_k\}$
17:            end if
18:        end for
19:    end if
20: end if
21: return $T_s$
```

diffusion process ends. All the tasks have the same threshold and discount. In the experiments, the two parameters are set heuristically: $TS = 0.5$, $\gamma = 0.5$.

The time complexity of calculating EG_i is at most $O(|T_f|)$. Then, the worst time complexity of Algorithm 3 is $O(|T_f||T|)$.

5.2 Selecting the Combination of Functional Tasks

The input of Algorithm 3 is a task set. It means Algorithm 3 can score the combination of tasks instead of only scoring a single functional task. However, analyzing all combinations of functional tasks is not feasible. Therefore, we firstly score each functional task singly. The effects of different functional tasks may overlap. Directly selecting several functional tasks associated with high scores is not necessarily suitable. In this case, we can gradually add the functional tasks to a set according to the score ranking and use Algorithm 3 to analyze whether the score of the set increases. This process is shown in Algorithm 4.

In Algorithm 4, if the score of a single functional task is larger than the time cost of completing this task, the task is considered "useful". Line 1-4 sort the functional tasks according to the scores and time costs. In line 6, if the best functional task is still not "useful", none of the functional tasks will be selected. Otherwise, in line 7-12, the functional tasks are gradually added to a set according to the score ranking. In line 13-19, if the number of selected functional tasks is smaller than $|R|$, some "useful" tasks can be extra selected to seek the load balance. This is a heuristic setting.

The time complexity of line 1-3 is $O(|T_f| \times |T_f| \times |T|)$. The time complexity of line 4 is at most $O(|T_f| \times |T_f|)$. The time complexity of line 5-16 is at most $O(|T_f| \times |T_f| \times |T| + |T_f|) = O(|T_f|^2|T|)$. Therefore, the worst time complexity of Algorithm 4 is $O(|T_f|^2|T|)$. If $|T_f| = 40, |T_c| = 40, |R| = 10$, it is known that $O(|T_f|^2|T|) = 128000$, $O(\frac{|T_f|^2|T_c|^2}{|R|^2}) = 25600$, $O(|T_f|^2|T_c|^2|R|) = 25600000$. Algorithm 4 runs slowly than Algorithm 1 but much faster than Algorithm 2.

Finally, the heuristic algorithm can be summarized as: 1) select a set of functional tasks based on Algorithm 4; 2) Allocate the selected functional tasks based on the SSI algorithm and obtain the task sequences; 3) assuming the selected functional tasks have been completed (the time costs of completing other tasks have decreased), insert all compulsory tasks into the sequences based on the SSI algorithm.

In the full algorithm, Algorithm 4 is performed once, and SSI is performed twice. The worse complexity of SSI is about $O(|R||T|)$. $O(|R||T|)$ is much smaller than the complexity $O(|T_f|^2|T|)$ of Algorithm 4. In addition, $O(|T_f|^2|T|)$ has already included the complexity of Algorithm 3. Therefore, the complexity of the full algorithm can also be represented by $O(|T_f|^2|T|)$. In the two greedy algorithms, $O(|R||T|)$ is also relatively small.

6 Simulated Experiments

Our experiments were conducted via simulation. Some parameters are fixed: $XY = 100 \times 100$; $|T_c| = 40$; f_i is randomly set in [1,5]; ν_i is randomly set in $[1, \nu_{max}]$; $g(T_f^j) = \frac{1}{1+ln\prod_{t_k \in T_f^j} f_k}$. All tasks are randomly uniformly distributed in the grid world. Each experiment is performed with 1000 replications, and the average data are shown in the experimental figures. In each replication, the random values are reset.

As shown in figure 4, compared with basic SSI, all other algorithms can greatly decrease the time costs. In figure 4 (a), (b) and (c), when the abscissa value is small but not the smallest [1], SSI+Greedy (slow) outperforms all other algorithms. Along with the increase of the abscissa value, the heuristic algorithm becomes the best. It means that the advantage of the heuristic algorithm is more obvious when the influence of the functional tasks is greater (i.e. when the parameters $\varepsilon, \nu_{max}, |T_f|$ are larger). In figure 4 (d), larger $|R|$ is slightly equivalent to smaller $|T_f|$. When $|R|$ increases, the advantage of the heuristic algorithm becomes less obvious.

As the greedy algorithms can only be used after the sequences of the compulsory tasks are fixed, it may lead to the locally optimal solution. When ε, ν_{max} and $|T_f|$ are small, the functional task may just be able to influence very few task sequences. In this case, local optimal solution is very close to global optimal solution. The heuristic algorithm allocates compulsory tasks after functional tasks; thus, the sequences of the compulsory tasks may be different from those obtained by directly allocating compulsory tasks. Therefore, if ε, ν_{max} and $|T_f|$ are large, the heuristic algorithm further decreases the maximum time cost than SSI+Greedy (slow).

[1]It must be explained that the leftmost data points in figure 4 (b) are obtained with $\nu_{max} = 1$ instead of $\nu_{max} = 0$.

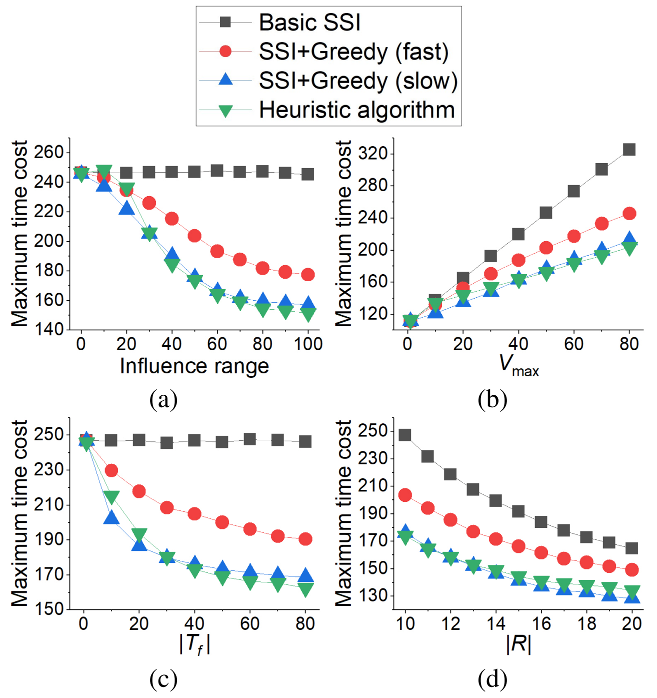

Figure 4: The experimental data. Except the variable parameter (abscissa axis), the other parameters are set as: $\varepsilon = 50, \nu_{max} = 50, |T_f| = 40, |R| = 10$.

Note that the time complexity of SSI+Greedy (slow) is too large to make this algorithm practical. Although SSI+Greedy (slow) outperforms the heuristic algorithm in a few cases, it can also be concluded that the heuristic algorithm is the best of the tested algorithms.

7 Conclusion and Future Works

This paper studies the multi-robot task allocation with functional tasks. Previous algorithms cannot suitably allocate the functional tasks. In most cases, neither allocating all functional tasks nor allocating no functional task is always optimal. We fully analyze this problem and modify the previous classical algorithm based on the greedy algorithm frame. The modified algorithms are used as the benchmark algorithms. Considering the shortages of the greedy algorithms, we design a heuristic algorithm mainly based on scoring the functional tasks. The simulated experiments are presented to test the proposed algorithms. It can be concluded that the heuristic algorithm outperforms all benchmark algorithms.

In the future work, we will study this problem in a communication-constrained and distributed environment. If the robots are initially distributed in different areas and are under communication-constrained, suitably allocating the compulsory tasks and functional tasks will be more difficult.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61936001, 61932007 and 62076060), the Natural Science Foundation of Chongqing (cstc2019jcyj-cxttX0002), the Key Cooperation Project of Chongqing Municipal Education Commission (HZ2021008).

References

[Behnck *et al.*, 2015] Lucas P Behnck, Dionisio Doering, Carlos Eduardo Pereira, and Achim Rettberg. A modified simulated annealing algorithm for suavs path planning. *Ifac-Papersonline*, 48(10):63–68, 2015.

[Bischoff *et al.*, 2020] Esther Bischoff, Fabian Meyer, Jairo Inga, and Sören Hohmann. Multi-robot task allocation and scheduling considering cooperative tasks and precedence constraints. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3949–3956. IEEE, 2020.

[Chen *et al.*, 2018] Xinye Chen, Ping Zhang, Guanglong Du, and Fang Li. Ant colony optimization based memetic algorithm to solve bi-objective multiple traveling salesmen problem for multi-robot systems. *IEEE Access*, 6:21745–21757, 2018.

[Emam *et al.*, 2020] Yousef Emam, Siddharth Mayya, Gennaro Notomista, Addison Bohannon, and Magnus Egerstedt. Adaptive task allocation for heterogeneous multi-robot teams with evolving and unknown robot capabilities. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7719–7725. IEEE, 2020.

[Koenig *et al.*, 2008] Sven Koenig, Xiaoming Zheng, Craig Tovey, Richard Borie, Philip Kilby, Vangelis Markakis, and Pinar Keskinocak. Agent coordination with regret clearing. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press*, page 101, 2008.

[Lee, 2018] Dong-Hyun Lee. Resource-based task allocation for multi-robot systems. *Robotics and Autonomous Systems*, 103:151–161, 2018.

[McIntire *et al.*, 2016] Mitchell McIntire, Ernesto Nunes, and Maria Gini. Iterated multi-robot auctions for precedence-constrained task scheduling. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1078–1086, 2016.

[Sullivan *et al.*, 2019] Nick Sullivan, Steven Grainger, and Ben Cazzolato. Sequential single-item auction improvements for heterogeneous multi-robot routing. *Robotics and Autonomous Systems*, 115:130–142, 2019.

[Suslova and Fazli, 2020] Elina Suslova and Pooyan Fazli. Multi-robot task allocation with time window and ordering constraints. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6909–6916. IEEE, 2020.

[Wei *et al.*, 2020] Changyun Wei, Ze Ji, and Boliang Cai. Particle swarm optimization for cooperative multi-robot task allocation: a multi-objective approach. *IEEE Robotics and Automation Letters*, 5(2):2530–2537, 2020.

[Yan *et al.*, 2014] Fuhan Yan, Zhaofeng Li, and Yichuan Jiang. Noised diffusion dynamics with individual biased opinion. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, pages 1129–1130, 2014.

[Yan *et al.*, 2016] Fuhan Yan, Zhaofeng Li, and Yichuan Jiang. Controllable uncertain opinion diffusion under confidence bound and unpredicted diffusion probability. *Physica A: Statistical Mechanics and its Applications*, 449:85–100, 2016.

[Yan *et al.*, 2019] Fuhan Yan, Kai Di, Jiuchuan Jiang, Yichuan Jiang, and Hui Fan. Efficient decision-making for multiagent target searching and occupancy in an unknown environment. *Robotics and Autonomous Systems*, 114:41–56, 2019.

[Yang *et al.*, 2019] Lan Yang, Zhiwu Li, and Alessandro Giua. Influence minimization in linear threshold networks. *Automatica*, 100:10–16, 2019.

[Yuan *et al.*, 2013] Shuai Yuan, Bradley Skinner, Shoudong Huang, and Dikai Liu. A new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms. *European journal of operational research*, 228(1):72–82, 2013.