IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 24, 2019, accepted January 9, 2020, date of publication January 17, 2020, date of current version January 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2967061

Task Allocation for Multi-Agent Systems Based on Distributed Many-Objective Evolutionary Algorithm and Greedy Algorithm

JING ZHOU[1,2], XIAOZHE ZHAO[1], XIAOPAN ZHANG[3], DONGDONG ZHAO[4], (Member, IEEE), AND HUANHUAN LI[5]

[1]Faculty of Management and Economics, Dalian University of Technology, Dalian 116024, China
[2]Operation Software and Simulation Institute, Dalian Navy Academy, Dalian 116018, China
[3]School of Resources and Environmental Engineering, Wuhan University of Technology, Wuhan 430070, China
[4]School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China
[5]School of Computer Science, School of Mechanical Engineering and Electronic Information, China University of Geosciences, Wuhan 430074, China

Corresponding author: Huanhuan Li (julylhh@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 71701208.

ABSTRACT Task allocation is a key issue in multi-agent systems, and finding the optimal strategy for task allocation has been proved to be an NP-hard problem. Existing task allocation methods for multi-agent systems mainly adopt distributed full search strategies or local search strategies. The former requires a lot of computation and communication costs, while the latter cannot ensure the diversity and quality of solutions. Therefore, in this paper, we combine a distributed many-objective evolutionary algorithm called D-NSGA3 with a greedy algorithm to search the task allocation solutions, and we comprehensively consider the constraints related to space, time, energy consumption and agent function in multi-agent systems. Specifically, D-NSGA3 is used to optimize multiple objectives simultaneously so as to ensure the search capability and the diversity of solutions. Alternate search between D-NSGA3 and the greedy algorithm is conducted to enhance the local optimizing ability. Experimental results show that the proposed method can effectively solve large-scale task allocation problems (e.g., the number of agents is not less than 250, and that of tasks is not less than 1000). Compared with the existing work called MSEA, the proposed method could achieve better and more diverse solutions.

INDEX TERMS Evolutionary algorithm, Greedy algorithm, multi-agent system, NSGA3, task allocation.

I. INTRODUCTION

Internet of Things and Artificial Intelligence technologies have made great progress in the past decade, and meanwhile, multi-agent systems [1] begin to be widely employed in real-world applications, such as unmanned systems [2], intelligent distributed traffic signal control systems [3], UAV formation combat systems [4], social networks [5], smart manufacturing [6], collaborative fault diagnosis systems [7], and robot rescue systems [8]. In a specific scenario, agents usually need to finish specific tasks, such as firefighting, excavation, obstacle clearing, crowd evacuation, rescue, and transportation of materials to designated locations.

Task allocation is one of the most important issues in multi-agent systems and finding the optimal solutions for the task allocation has been proved to be an NP-hard problem [9], [10]. The goal of task allocation is to optimize the performance or benefits of task execution, such as maximizing the number of successfully executed tasks and minimizing the time and resource consumption of task execution. In addition to optimizing objectives, a large number of constraints generally need to be satisfied during task allocation. For example, different types of agents may be able to perform different tasks due to their different configuration, performance, load and functions, e. g., agents dedicated to firefighting can only perform firefighting tasks, and agents equipped with digging tools can perform multiple tasks like digging, obstacles clearing and rescue. Besides, different agents and tasks may distribute at different geographic locations, and tasks may only be valid in a certain time interval (e. g., a rescue task in the fire disaster needs to be completed in a short time, otherwise, victims may have been killed or saved, and the

The associate editor coordinating the review of this manuscript and approving it for publication was Ran Cheng.

task will be disabled). Therefore, it is necessary to consider the corresponding spatial/resources and temporal constraints during task allocation [9]. In many applications, agents are limited to only be able to communicate with the control server or the agents close to them, and the environment surrounding the agents would change over time. In such cases, it is more promising to solve the task allocation problem in the distributed manner.

Presently, many distributed algorithms for task allocation have been proposed, and they can be divided into the following classes: distributed full search algorithms [11]–[15], distributed local search algorithms [16]–[18], algorithms based on auction mechanism [19], distributed particle swarm optimization [20], and distributed ant colony algorithm [21]. The distributed full search algorithms can obtain optimal solutions for task allocation, but they require high communication and computation cost, and they would be impractical when solving large-scale problems. The other four kinds of algorithms are either unable to guarantee the quality or diversity of solutions, or unable to optimize multiple objectives simultaneously or do not consider the large amount of constraints in task allocation problems.

Evolutionary algorithm, one of the most widely used intelligent optimization algorithms, has the optimizing ability comparable to the particle swarm optimization [22]. In the past decade, a lot of researches have been devoted to distributed evolutionary algorithms [23], [24], but on which few task allocation methods for multi-agent systems are formed. And in most of the methods, simultaneous optimization of multiple objectives in task allocation is not fully considered. In view of the above, this paper comprehensively considers constraints related to space, time, energy consumption and function, etc., in task allocation, and simultaneously optimizes four objectives, i.e. maximizing the number of successfully executed tasks, maximizing the benefits of performing tasks and minimizing task execution time and resource consumption. This paper adopts a distributed many-objective evolutionary algorithm called D-NSGA3 [25], [26] to optimize multiple objectives simultaneously and ensure the diversity of solutions. Furthermore, this paper combines the greedy algorithm to compensate for the poor ability of D-NSGA3 for local optimization. The experimental results show that the proposed algorithm can effectively solve large-scale task allocation problems (the number of agents $\geq$ 250, and that of tasks $\geq$1000). Compared with the existing work called MSEA [26], the proposed algorithm could find better and more diverse solutions. Overall, our contributions are listed as follows:

(1) We propose a greedy algorithm that is designed based on the chromosome structure used in D-NSGA3 for solving the task allocation problem in multi-agent systems.

(2) We propose an approach for combining the D-NSGA3 and the greedy algorithm, which could have higher effectiveness than D-NSGA3, and we apply the approach to task allocation in multi-agent systems.

(3) Several experiments are conducted to demonstrate the effectiveness of the proposed approach, and results show that our approach is able to solve large-scale task allocation problems.

The rest of this paper is arranged as follows: Section II introduces the formal description of the task allocation problem in multi-agent systems; Section III introduces the combination scheme of D-NSGA3 algorithm and a greedy algorithm proposed in this paper; Section IV presents the experimental results and analysis; The proposed work is discussed in Section V; Section VI presents the related work about task allocation methods; Section VII summarizes the work in this paper.

II. PRELIMINARIES

This section introduces the formal description of task allocation problem in multi-agent systems. The elements are defined as follows:

(1) **Agent**: $A = \{a_1 \ldots a_m\}$, where m is the number of agents. The initial positions of the agents are defined as $L_A = \{l_{a1} \ldots l_{am}\}$, where l_{ai} $(i = 1 \ldots m)$ represents the initial position of the agent a_i. Each agent will be equipped with a certain amount of resources (e.g. energy and electricity) at the beginning for executing tasks, and the resources are denoted as: $R = \{r_1 \ldots r_m\}$, where r_i $(i = 1 \ldots m)$ represents the amount of resources allocated to the agent a_i.

(2) **Task**: $V = \{v_1 \ldots v_n\}$, where n is the number of tasks. The locations of the tasks are defined as $L_V = \{l_{v1 \ldots} l_{vn}\}$, where $l_{vi}(i = 1 \ldots n)$ represents the current location of the task v_i. We assume that each task is limited to be executable among a certain time interval, which is denoted by the earliest executable time and the latest executable time, i.e. $T_V = \{[low_1, up_1] \ldots [low_n, up_n]\}$, where low_i is the earliest executable time and up_i is the latest executable time. The time cost by executing tasks is represented as$TC = \{tc_1 \ldots tc_n\}$, where tc_i represents the time taken to execute the task v_i. The amount of resources required to execute the tasks is expressed as $RC = \{rc_1 \ldots rc_n\}$, where rc_i $(i = 1 \ldots n)$ represents the amount of resources needed to execute the task v_i. The benefits gained by executing tasks are expressed as$Gain = \{gain_1 \ldots gain_n\}$, where $gain_i$ $(i = 1 \ldots n)$ represents the benefits achieved by executing the task v_i.

(3) **Allocation Relation**: $\boldsymbol{\pi} = \{\pi_1 \ldots \pi_n$, where $\pi_i(i = 1 \ldots n)$ represents the allocation of the task v_i. For example, $\pi_i = v_i \rightarrow a_j$ indicates that the task $v_i$is assigned to the agent a_j for execution (it also implies a_j can successfully execute v_i), and $\pi_j = v_j \rightarrow \emptyset$ indicates that $v_j$is not assigned to any agents. For simplicity, $\pi_i = v_i \rightarrow a_j$ and $\pi_j = v_j \rightarrow \emptyset$ are simplified as $\pi_i = a_j$ and $\pi_j = Null$ respectively in this paper. Similar to [9], [15], [26], [27], to simplify the problem model, this paper assumes that time is discretized (e.g. in milliseconds), and each agent can only perform one task at a piece of time, and each task only needs be executed and completed by one agent.

(4) **Execution Sequence**: $Q = \{q_1 \ldots q_m\}$, where $q_i = \{q_{i1} \ldots q_{ik}\}$ denotes the sequence of k tasks to be

executed by the agent a_i (k might be different for different agents), for example $q_i = \{v_2, v_3, v_6\}$ indicates tasks v_2, v_3 and v_6 are allocated to the agent a_i, and a_i will execute these three tasks, respectively.

(5) **Objectives**: The four most widely studied objectives in task allocation are considered in this paper:

1) Maximizing the number of successfully executed tasks:
$$f_1 = maximize_{\forall\pi} \sum_{i=1}^{n} \{\pi_i \neq Null\} \quad (1)$$
where $\{\pi_i \neq Null\}$ returns 1 when the condition$\pi_i \neq Null$ is true, otherwise, it returns 0.

2) Maximizing the benefits of executing tasks:
$$f_2 = maximize_{\forall\pi} \sum_{i=1}^{n} \{\pi_i \neq Null\} \times gain_i \quad (2)$$

3) Minimizing the resources consumed in executing tasks:
$$f_3 = minimize_{\forall Q} \sum_{i=1}^{m} \Big(travel\_cost\left(l_{ai}, l_{vq_{i1}}\right) + \sum_{j=2}^{|q_i|} travel\_cost\left(l_{vq_{i(j-1)}}, l_{vq_{ij}}\right) + \sum_{j=1}^{|q_i|} rc_{q_{ij}}\Big) \quad (3)$$
where $travel\_cost\left(l_x, l_y\right)$ denotes the resources required for the agent to move from locationl_x to l_y, and $|q_i|$ is the length of the sequence q_i.

4) Minimizing the maximum time spent by agents to execute tasks:
$$f_4 = minimize_{\forall Q} max_{i=1}^{m} Tcost(q_i, |q_i|) \quad (4)$$
where $Tcost(q_i, j)$ denotes the time spent by the agent a_i to execute the first j tasks in q_i.$Tcost\,(q_i, 1) = travel\_time\left(l_{ai}, l_{vq_{i1}}\right) + tc_{q_{i1}}$, and for $j = 2 \ldots |q_i|$ we have:
$$\begin{aligned} Tcost\,(q_i, j) = \max\{&Tcost\,(q_i, j-1) \\ &+travel\_time\left(l_{vq_{i(j-1)}}, l_{vq_{ij}}\right), low_{q_{ij}}\} \\ &+tc_{q_{ij}} \end{aligned}$$
where $travel\_time\left(l_x, l_y\right)$ represents the time needed for the agent to move from the location l_x to l_y.

(6) **Constraint Conditions:** Generally, time, resources (or space), and function constraints are involved in the task allocation, which are formally expressed as:

1) Time Constraints: if an agenta_i ($1 \leq i \leq m$) wants to successfully execute the task v_j, it must arrive at the location of v_j and execute the task before time up_j. The execution sequence q_i needs to meet the following requirements:
$$\begin{aligned} g_{i1}^{T} &= arrival\_time(q_i, 1) \\ &= travel\_time\left(l_{ai}, l_{vq_{i1}}\right) \leq up_{q_{i1}} \end{aligned}$$
For $2 \leq j \leq |q_i|$, we have:
$$\begin{aligned} g_{ij}^{T} &= arrival\_time\,(q_i, j) \\ &= Tcost\,(q_i, j-1) + travel\_time\left(l_{vq_{i(j-1)}}, l_{vq_{ij}}\right) \\ &\leq up_{q_{ij}} \end{aligned} \quad (5)$$
where $arrival\_time\,(q_i, j)$ denotes the time when a_i arrives at the position of the jth task in q_i.

2) Resource Constraints: If the agent a_i ($1 \leq i \leq m$) wants to successfully execute a task v_j, its current resources should be sufficient to support itto move to v_j and execute the task, that is:
$$\begin{aligned} g_{ij}^{R} &= resource\_cost\,(q_i, j) = travel\_cost(l_{ai}, l_{vq_{i1}}) \\ &+ \sum_{k=2}^{j} travel\_cost\left(l_{vq_{i(k-1)}}, l_{vq_{ik}}\right) \\ &+ \sum_{k=1}^{j} rc_{q_{ik}} \leq r_i \end{aligned} \quad (6)$$
where $resource\_cost(q_i, j)$ denotes the amount of resources required for $a_i$to execute the first$j$ tasks in q_i.

3) Function Constraints: In real-world applications, agents may have different functions and different tasks that can be performed. Each agent can only perform the tasks supported by its functions. Therefore:

a) For an agent a_i ($1 \leq i \leq m$), we have:
$$g_i^{FA} = \left\{v_{i_1}, \ldots, v_{i_b}\right\} :\to a_i \quad (7)$$
This constraint defines that the functions of a_i only support executing the tasks in the set $\left\{v_{i_1}, \ldots, v_{i_b}\right\}$.

b) For a taskv_j($1 \leq j \leq n$), we have:
$$g_j^{FT} = v_j :\to \left\{a_{j_1}, \ldots, a_{j_c}\right\} \quad (8)$$
This constraint defines that v_j can only be allocated to the agents in the set $\left\{a_{j_1}, \ldots, a_{j_c}\right\}$ for execution according to its function requirements.

III. TASK ALLOCATION BASED ON D-NSGA3 AND GREEDY ALGORITHM

This paper mainly combines a distributed many-objective evolutionary algorithm called D-NSGA3 and a greedy algorithm to solve the above task allocation problem in multi-agent systems. The following firstly presents the hybrid framework of D-NSGA3 and the greedy algorithm, then introduces chromosome encoding, crossover and mutation strategies and the evaluation model adopted in this paper, and finally introduces the greedy algorithm.

A. FRAMEWORK OF PROPOSED METHOD

The NSGA3 algorithm was proposed by Deb and Jain [25] in 2013, and it mainly improves the NSGA-II algorithm to solve high-dimensional optimization problems, i.e. optimization problems with more than 3 objectives. This paper combines NSGA3 with the Master-Slave distributed model [24], and the framework is shown in Figure 1.

In multi-agent systems, the command and control server (CCS) can act as a master node to be responsible for the population initialization, iterative evolution, crossover, mutation, selection operations in NSGA3. In many scenarios, as the environment in which the agents are located may change at any time, task-related indicators and information (e.g. the energy consumption required to execute the task and the benefits from executing the task) will also change, which will

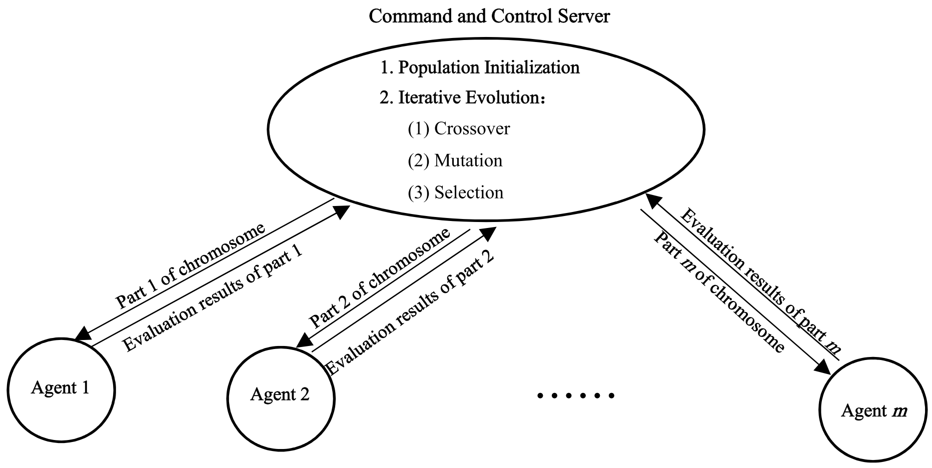

FIGURE 1. **Master-Slave model used by D-NSGA3.**

eventually leads to changes in the evaluation of task allocation strategies. On the other hand, as the communication speed and scope are limited, an agent can only perceive the surrounding environment information. Therefore, agents can only evaluate the part of the task allocation strategy that involves this information. Hence, an agent or a group of locally and perceptively interconnected agents is taken as a slave node in this paper. Master node will divide and distribute the task allocation strategy (i.e. chromosome in NSGA3) according to the evaluation capability of each slave node. And slave nodes will return the evaluation results to the master node after evaluation. In multi-agent systems, CCS can better complete most of the operations during the evolution with its strong computing power and communication broadband. While agents can only complete lightweight computation and communication, because of its limited computing and communication capabilities. Therefore, the Master-Slave model is very suitable for multi-agent systems.

The framework of combining D-NSGA3 with the greedy algorithm (called D-NSGA3-Greedy for simplicity) is shown in algorithm 1, and its general idea is as follows:

1) The master node first randomly generates an initial population with a size of N;
2) The master node segments the population according to the evaluation capability of each slave node, and sends segments to corresponding slave nodes for evaluation;
3) Slave nodes evaluate the segments sent from the master node, and send back the results;
4) The master node collects the results from slave nodes, and calculates the overall objective values of each individual;
5) The master node generates the offspring population through crossover and mutation operators;
6) The master node and slave nodes evaluate the offspring individuals cooperatively;
7) The master node performs non-dominated sorting on the union set of parent population and offspring population to form H non-dominated layers $F_1 \ldots F_H$ and finds out the individuals of the first h non-dominated layers to make $|F_1 \cup \ldots \cup F_{h-1}| < N \leq |F_1 \cup \ldots \cup F_h|$;
8) The master node determines the population for next generation through the selection operation. Specifically, if $|F_1 \cup \ldots \cup F_h| = N$, then the individuals of the first h layers are directly selected for the next generation. Otherwise, $N - |F_1 \cup \ldots \cup F_{h-1}|$ individuals are selected from the critical layer F_h through reference points, and they are taken for the next generation together with the individuals in $F_1 \ldots F_{h-1}$ layers. Due to the limited space, this paper will not introduce the details of NSGA3 (such as the selection operator based on reference points), please refer to [25] for details.
9) By every D generations, the greedy algorithm in Section III.C is used to locally optimize the selected individuals.
10) Repeat the above steps 5-9 until the population converges or the algorithm exceeds the maximum number of iterations.

B. CHROMOSOME ENCODING, INITIALIZATION, CROSSOVER, MUTATION AND EVALUATION

In EAs, the chromosome encoding, initialization, crossover and mutation strategies take significant effect on the converging speed and diversity of population evolution, thus affect the quality of solutions. In this paper, the chromosome is encoded as: $x = x_1 \ldots x_m$, where x_i is a sequence of tasks that the agent a_i attempts to perform. For example, in view of the function constraints of a_i is $g_i^{FA} = \left\{v_{i_1}, \ldots, v_{i_b}\right\} :\to a_i$, then $x_i = v_{j_1}, \ldots, v_{j_b}$ is a permutation of $v_{i_1}, \ldots, v_{i_b}$, indicating that a_i will consider the execution of tasks $v_{j_1}, \ldots, v_{j_b}$in order. If, after executing tasks$v_{j_1}, \ldots, v_{j_k}, a_i$ cannot execute $v_{j_{k+1}}$ due to the lack of time or resources, then a_i will skip this task and consider executing the next task$v_{j_{k+2}}$. If a_i can execute the task v_j under the condition of satisfying the constraints, then subsequent agents $a_{i+1} \ldots a_m$ will no longer consider executing v_j. For example, suppose there are two agents and

Algorithm 1 D-NSGA3-Greedy Algorithm

```
Input: Population size ($N$), maximum number of iterations ($MaxIterN$), reference point set ($RP$)
Output: Pareto solution set
Master node executes:
 1: $i \leftarrow 1$;
 2: Randomly initialize population $P_i$ with $N$ individuals;
 3: Segment each individual in $P_i$ and send each segment to the corresponding slave node for evaluation;
----------------------------------------
    Each slave node executes:
 4: Evaluate individual segments after receiving from the master node, and return the evaluation results to the master node;
----------------------------------------
    Master node executes:
 5: Receive all evaluation results from slave nodes, and calculate the overall objective values of each individual in $P_i$;
 6: While $i \le MaxIterN$ do
 7:    Carry out crossover and mutation operators on $P_i$ to generate an offspring population$Q_i$;
 8:    Segment each individual in $Q_i$ and send each segment to the corresponding slave node for evaluation;
----------------------------------------
    Each slave node executes:
 9: Evaluate individual segments after receiving from the master node, and return the evaluation results to the master node;
----------------------------------------
    Master node executes:
10: Receive all evaluation results from slave nodes, and calculate the overall objective values of each individual in $Q_i$;
11: According to the objective values, perform non-dominated sorting on $P_i \cup Q_i$ to obtain the non-dominated layers $F_1 \ldots F_H$;
12: Find the $h$ s.t. $|F_1 \cup \ldots \cup F_{h-1}| < N \le |F_1 \cup \ldots \cup F_h|$;
13:    If $|F_1 + \ldots + F_h| = N$ then
14:       $\leftarrow P_{i+1} F_1 \cup \ldots \cup F_h$;
15:    Else
16:       $\leftarrow P_{i+1} F_1 \cup \ldots \cup F_{h-1}$;
17:       Find other $N - |F_1 \cup \ldots \cup F_{h-1}|$ individuals based on reference points in $RP$, and add them to $P_{i+1}$;
18:    End if
19:    If$i \% D$ =0then
20:       Foreach individual$x \in P_{i+1}$do;
21:          $Greedy(x)$;
22:       End for
23:    End if
24:    If the population has converged then
25:       Return $F_1$ as the pareto solution set;
26:    End if
27:    $i \leftarrow i$+1;
28: End while
29: Return $F_1$ as the pareto solution set.
```

five tasks, according to function constraints, if a_1 is able to perform tasks v_1, v_2 and v_3; a_2 is able to perform tasks v_3, v_4 and v_5. Therefore, the chromosome $x = 3\ 1\ 2\ 4\ 3\ 5$ indicates that a_1 will consider executing v_3, v_1 and v_2 in order, and a_2 will consider executing v_4, v_3 and v_5 in order. If a_1 cannot successfully execute v_2 within resource constraints or time constraints after executing v_3 and v_1, then a_1 can successfully execute 2 tasks at last. As v_3 has been allocated to a_1 for execution, a_2 only needs to consider executing v_4 and v_5 in order. The advantages of this encoding are as follows: 1) fixed encoding length for all individuals; 2) no need to consider function constraints when evaluating individuals; 3) simplifying the evaluation process; 4) simplifying initialization, crossover, and mutation operators.

This paper adopts random population initialization. According to the above encoding method, the master node obtains the set of tasks $\{v_{i_1}, \ldots, v_{i_b}\}$that each agent a_i is able to execute according to the function constraints, and then generates a random permutation of$v_{i_1}, \ldots, v_{i_b}$as the part i (i.e. x_i) of an initial individual x.

This paper adopts the single-point crossover operator. For two parent individuals, $x = x_1 \ldots x_m$ and $y = y_1 \ldots y_m$, the master node randomly generates a number k between 1 and m, and then, it exchanges the first k sequences between x and y to obtain the offspring $x' = y_1 \ldots y_k x_{k+1} \ldots x_m$ and $y' = x_1 \ldots x_k y_{k+1} \ldots y_m$.

In mutation operator, this paper adopts a parameter *mutPr* to control the probability of random mutation. If the ith sequence x_i of an individual is selected for mutation according to this probability, then the master node will generate a random permutation of $v_{i_1}, \ldots, v_{i_b}$ to replace the original x_i.

All the objective functions in Section II are converted into a "minimize" form to facilitate the evaluation, specifically, we have:

1) Evaluation function of the first objective:

$$f_1(x) = 1.0 - \frac{\sum_{i=1}^{m} h1\,(x_i)}{n} \tag{9}$$

where $h1\,(x_i) = \sum_{w=1}^{b} Add(j_w)$ denotes the number of tasks in x_i that a_i can successfully execute, and

$$Add\,(j_w) = \begin{cases} 0 & if v_{j_w}\ has\ been\ allocated, \\ & or arrival\_time(x_i^E + v_{j_w}, |x_i^E| + 1) > up_{j_w} \\ & or resource\_cost(x_i^E + v_{j_w}, |x_i^E| + 1) > r_i \\ 1 & otherwise \end{cases}$$

calculates whether a_i can successfully execute v_{j_w}in the current state. x_i^E denotes the current sequence of tasks that have been checked and can be successfully executed (initially the empty set), and $x_i^E + v_{j_w}$ denotes adding v_{j_w} to the sequence x_i^E.

2) Evaluation function of the second objective:

$$f_2(x) = 1.0 - \frac{\sum_{i=1}^{m} h2\,(x_i)}{\sum_{i=1}^{n} gain_i} \tag{10}$$

where $h2(x_i) = \sum_{w=1}^{b} Add(j_w) \times gain_{j_w}$ denotes the benefits that a_i can obtain from executing tasks in x_i.

3) Evaluation function of the third objective:

$$f_3(x) = \sum_{i=1}^{m} \frac{h3(x_i)}{m \times max\_r} \tag{11}$$

where max_r represents the maximum amount of resources an agent can equip, and $h3(x_i) = \sum_{w=1}^{b} Add(j_w) \times \left(travel\_cost\left(l_{last}, l_{vj_w}\right) + rc_{j_w}\right)$ denotes the amount of resources consumed by a_i in executing tasks in x_i.l_{last} represents the location of the last task executed by a_i, and initially l_{last} is the position of a_i.

4) Evaluation function of the fourth objective:

$$f_4(x) = max_{i=1}^{m} \frac{h4(x_i)}{m \times max\_t} \tag{12}$$

where max_t denotes the latest time that all tasks can be completed. And $h4(x_i) = \sum_{w=1}^{k} Add(j_w) \times (travel\_time(l_{last}, l_{vj_w}) + tc_{j_w})$ denotes the time cost by a_i to execute tasks in x_i.

C. GREEDY ALGORITHM

This paper designs a greedy algorithm for locally fast optimization of the individuals in D-NSGA3 so as to accelerate the search and convergence speed, and thus to improve the quality of solutions. The pseudo code is shown in algorithm 2. The main steps are as follows:

1) CCS sets the weight of each objective for an individual x and calculates the weighted sum of four objectives as the optimization objective F;
2) For each sequence x_i, CCS exchanges each pair of positions of two tasks in x_i, and sends the segments of the updated x to each agent for re-evaluation;
3) The agents evaluate the segments of the updated x and return the evaluation results to CCS;
4) CCS receives the evaluation results from agents and calculates the F value. If the F value is not improved, CCS will restore x_i;
5) CCS iteratively performs steps 2)-4) until F cannot be improved and then returns x.

The F value is calculated as follow:

$$F = \sum_{i=1}^{4} w_i \times f_i(x) \tag{13}$$

where w_i, the weight of the objective function $f_i(x)$, can be set according to the importance, urgency and other factors of the objective in a certain scenario. w is used in the function $getFvalue(x)$, which calculates the F value. This paper adopts randomly generated test cases of the task allocation problem for experiments, so we use the reference points in D-NSGA3 as the weights of individuals in the greedy algorithm as they distribute uniformly in the objective domain. For example, for the ith individual x in the population of D-NSGA3, the weight $w = RF[i\% |RF|]$, where '%' is the modulus operator, and |RF| denotes the number of reference points.

Algorithm 2 Greedy Algorithm

```
Input: individual $x$
Output: optimized individual $x$
CCS executes:
1:  $oldF \leftarrow getFvalue(x)$;
2:  do
3:    for $i = 1$ to $m$ do
4:      for $\forall v_{i_j}, v_{i_k} \in x_i \; (j \neq k)$ do
5:        Exchange the positions of $v_{i_j}$ and $v_{i_k}$ in $x_i$;
6:        Send the segments of $x$ to agents for evaluation;
Agents execute:
7:        Receive the segments of $x$ from CCS and conduct evaluation, and return the evaluation results to CCS;
CCS executes:
8:        Receive the evaluation results from the agents and calculate $newF \leftarrow getFvalue(x)$;
9:        if $oldF - newF \leq 0$ then
10:         Restore the positions of $v_{i_j}$ and $v_{i_k}$ in $x_i$;
11:       end if
12:     end for
13:   end for
14: while($x$ is updated)
15: return $x$
```

D. TIME COMPLEXITY

To evaluate the time cost of the proposed method, we estimate the time complexity of computation at both the Master node (i.e. CCS) and each Slave node (i.e. agent).

The Master node is responsible for conducting the population initialization (step 1 in algorithm 1), individual segmentation (step 2 in algorithm 1), calculating the overall objective values (step 5) and population evolution (step 6-8, 10-28). The time complexity of population initialization is $O(N \times m \times n)$ at the worst case (i.e. each agent is able to execute all tasks), where N is the population size and m is the number of agents and n is the number of tasks. The process of individual segmentation also takes $O(N \times m \times n)$ computations at the worst case. The time complexity of calculating the overall objective values is $O(N \times m \times n)$ at the worst case. In the process of population evolution, $maxIterN$ iterations will be carried out at the worst case that the population cannot converge successfully. At each iteration, the crossover operator takes $O(N \times m \times n)$ computations, and the mutation operator takes $O(N \times m \times n)$ computations in step 7. The segmentation in step 8 and the calculation of overall objective values in step 10 also take $O(N \times m \times n)$ computations. In step 11-12, the non-dominated sorting takes $O(N \times log_2 N)$ comparisons. In steps 13-18, if $|F_1 + \ldots + F_h| = N$, then it takes $O(N \times m \times n)$ computations; otherwise, it takes $O(N \times m \times n) + O(N \times |RP|)$ computations where $|RP|$ is the size of RP and it is usually set as a constant. The greedy algorithm is called per D iterations, so it will be called *maxIterN/D* times at the worst case. Each call of the greedy algorithm takes $O(N \times m \times n \times n)$ operators. Overall, for the worst case, the total complexity at the Master node is:

$$\begin{aligned} TC_M &= O(N \times m \times n) + O(N \times m \times n) + O(N \times m \times n) \\ &\quad + maxIterN \times [O(N \times m \times n) + O(N \times m \times n) \\ &\quad + O\left(N \times log_2^N\right) + O(N \times m \times n) + O(N \times |RP|)] \\ &\quad + \frac{maxIterN}{D} \times O(N \times m \times n^2) \\ &= O\left(maxIterN \times N \times \left(m \times n + log_2^N + \frac{m \times n^2}{D}\right)\right). \end{aligned}$$

The Slave nodes are responsible for individual evaluation. For evaluating an individual, it takes $O(m \times n)$ computations. The Slave a_i only evaluates the part x_i, so it takes $O(|x_i|)$ computations, where $|x_i|$ is the number of tasks that a_i is able to execute according to the function constraints. It should evaluate N individuals at each iteration. Moreover, in the greedy algorithm, each Slave node should evaluate $O(m \times |x_i|^2)$ individuals, where the "do …while" loop is assumed to iterate $O(1)$ times. Overall, for the worst case, the total complexity at Slave node a_i is:

$$\begin{aligned} TC_S &= O(|x_i|) \times N \times maxIterN + \frac{maxIterN}{D} \times m \times |x_i|^2 \\ &\quad \times N \times O(|x_i|) \\ &= O\left(|x_i| \times N \times maxIterN \times \left(1 + \frac{m \times |x_i|^2}{D}\right)\right) \end{aligned}$$

$|x_i|$ is usually about $O\left(\frac{n}{m}\right)$, so the average complexity at a Slave node is $O\left(\frac{n}{m} \times N \times maxIterN \times \left(1 + \frac{n^2}{m \times D}\right)\right)$. It is worth mentioning that the computation complexity is analyzed at the worst case, and the computation cost might vary significantly according to different task allocation problems in different scenarios.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

To verify and analyze the performance of the proposed algorithm, several experiments are carried out: 1) Experiments on the performance of D-NSGA3-Greedy algorithm on task allocation problems of different scales; 2) Comparison of D-NSGA3-Greedy algorithm and the algorithm proposed in [26] (called MSEA for simplicity). Similar to [26], this paper assumes that all agents move at the same speed and the time and resources cost is proportional to the distance.

The experimental environment in this paper is as follows: 64-bit Window 7 SP1; 3.40GHz Intel i7-6700 CPU; 8G Memory; programming environment: Visual Studio 2013.

In the experiments, m agents and n tasks are randomly generated. Specifically, for the initial position $l_{ai} = (x, y)$ $(i = 1 \ldots m)$ of an agent a_i, x and y are random integers from [0, 100). The quantity of resources r_i equipped with each agent is fixed to 500 to facilitate the experiment. For the position $l_{vi} = (x, y)$ $(i = 1 \ldots n)$ of each task v_i, x and y are also set as random integers in [0, 100). The earliest executable time low_i of v_i is a random number between [0, 100], and

TABLE 1. Results of D-NSGA3-Greedy for m = 10 and n=40.

f_1	f_2	f_3	f_4
0.33	0.28	0.42	0.78
0.33	0.33	0.41	0.93
0.35	**0.25**	0.41	0.88
0.35	0.31	0.40	0.88
0.35	0.33	0.40	0.72
0.35	0.34	0.39	0.77
0.38	0.40	0.38	0.72
0.40	0.27	0.38	0.79
0.40	0.28	0.37	0.87
0.40	0.31	0.34	0.74
0.43	0.35	0.33	0.66
0.43	0.37	0.32	0.82
0.43	0.43	0.32	0.70
0.48	0.40	0.30	**0.64**
0.50	0.34	0.32	0.68
0.50	0.46	0.28	**0.64**
0.50	0.48	0.26	0.73
0.50	0.51	0.25	0.89
0.53	0.38	0.29	0.79
0.53	0.48	0.25	0.79
0.55	0.45	0.26	0.70
0.55	0.51	0.24	0.66
0.55	0.56	0.23	0.68
0.60	0.57	0.22	0.89
0.63	0.55	0.21	0.68
0.63	0.57	0.27	**0.64**
0.63	0.62	0.26	**0.64**
0.68	0.66	0.22	0.65
0.70	0.69	**0.20**	0.73

the latest executable time up_i is a random number between $[low_i, 150)$. The time tc_i required to execute the task v_i is a random number between [1, 10), and the resource rc_i required to execute the task v_i is a random number between [0, 100). The benefits$gain_i$ from executing the task v_i is a random number between [0, 1). For function constraints, this paper assumes that each task can be executed by at least one agent (otherwise the task can be directly excluded), therefore we firstly randomly distribute all tasks into the executable task sets of agents. Besides, additional 0~$4n/m$ tasks are randomly added to the executable task set of each agent to increase the difficulty of the problem. The generated dataset is available at: http://www.wut-dscl.cn/in_data/.

We adopt the implementation version of NSGA3 by Chiang [28] in the experiments. The maximum number of iterations in D-NSGA3-Greedy algorithm is set to 500, the crossover probability is 1.0, and mutation probability is $\frac{1}{\text{ChromosomeLength}}$. The parameter D is set to 250 by default. The data precision in the greedy algorithm is set to 0.001.

A. VARYING PROBLEM SCALE

We test the task allocation problem of three scales, that is, m =10, n =40; m =50, n =200; m =250, n =1000. The experimental results of D-NSGA3-Greedy algorithm are shown in Table 1-3.

Table 1 shows the results of D-NSGA3-Greedy algorithm on four objectives for m =10 and n =40 (where the data precision is kept to 2 decimal places). Due to the limited

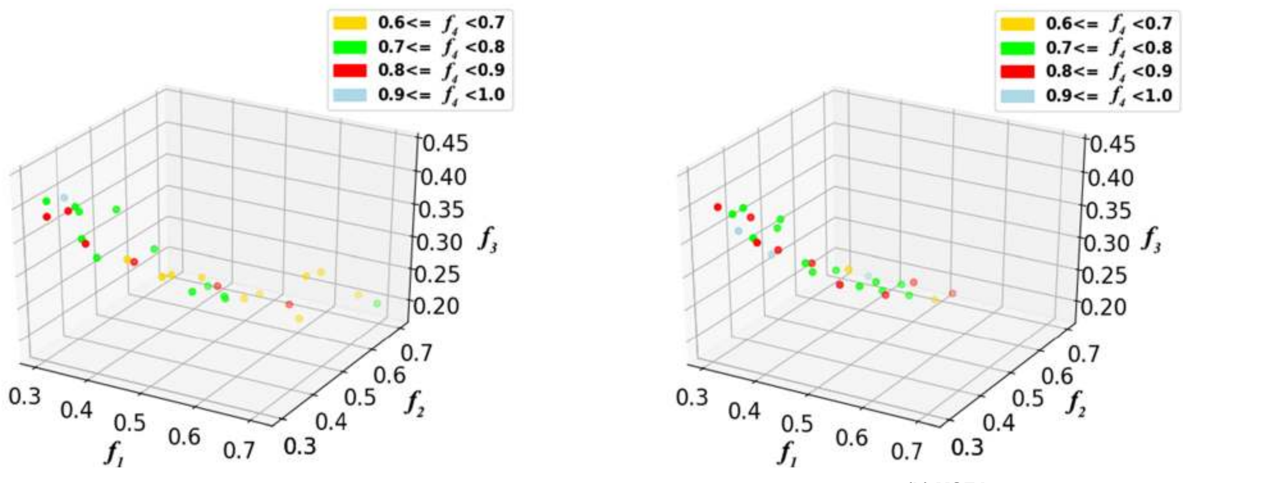

FIGURE 2. Distribution of the results for $m = 10$ and $n = 40$.

TABLE 2. Results of D-NSGA3-Greedy for m=50 and n=200.

f_1	f_2	f_3	f_4
0.31	**0.22**	0.43	0.89
0.31	0.29	0.41	0.85
0.32	0.27	0.41	0.96
0.35	0.26	0.40	0.78
0.37	**0.22**	0.40	0.87
0.37	0.30	0.35	0.90
0.37	0.33	0.37	0.77
0.38	0.34	0.36	0.84
0.39	0.24	0.39	0.92
0.40	0.27	0.35	0.81
0.43	0.29	0.32	0.93
0.43	0.31	0.34	0.74
0.44	0.31	0.31	0.88
0.44	0.38	0.32	0.77
0.47	0.32	0.32	0.81
0.47	0.41	0.28	0.91
0.50	0.43	0.28	**0.73**
0.51	0.41	0.26	0.81
0.52	0.37	0.27	0.93
0.54	0.46	0.25	0.82
0.54	0.47	0.24	0.91
0.56	0.51	0.24	**0.73**
0.59	0.53	0.23	0.87
0.60	0.56	**0.21**	0.81

TABLE 3. Results of D-NSGA3-Greedy for m=250 and n=1000.

f_1	f_2	f_3	f_4
0.34	**0.30**	0.40	0.95
0.37	**0.30**	0.37	0.89
0.39	0.36	0.36	0.89
0.40	0.32	0.36	0.96
0.45	0.37	0.32	**0.85**
0.46	0.40	0.30	0.93
0.48	0.43	0.31	0.86
0.53	0.50	**0.27**	0.96

space, only part of the results are presented. Specifically, for similar data (i.e. if the difference between the two data is within 0.01 on $f_1 \sim f_4$), only one of them is presented in the table, and finally 29 pieces of data are obtained. As shown in Table 1, the D-NSGA3-Greedy algorithm can optimize f_1 to 0.33, which indicates that 67% of the tasks can be assigned and successfully executed through the solutions; D-NSGA3-Greedy algorithm can optimize f_2,f_3,f_4 to 0.25, 0.20 and 0.64, respectively. The spans of f_1, f_2, f_3 and f_4 achieved by the 29 solutions are 0.37, 0.44, 0.22 and 0.29, respectively.

Table 2 shows the results of D-NSGA3-Greedy algorithm on four objectives for m =50 and n =200, with totally 24 pieces of data. D-NSGA3-Greedy algorithm can optimize f_1 to 0.31, which indicates that 69% of the tasks can be allocated and successfully executed by agents; D-NSGA3-Greedy algorithm can optimize f_2, f_3, f_4 to 0.22, 0.21 and 0.73, respectively. The spans of f_1, f_2, f_3 and f_4 achieved by the 24 solutions are 0.29, 0.34, 0.22 and 0.23, respectively.

Table 3 shows the results of D-NSGA3-Greedy algorithm on four objectives for m =250 and n =1000, with totally 8 pieces of data. D-NSGA3-Greedy algorithm can optimize f_1 to 0.34, which indicates that 66% of tasks can be successfully allocated and successfully executed by agents; D-NSGA3-Greedy algorithm can optimize f_2, f_3, f_4 to 0.30, 0.27 and 0.85, respectively. The spans of f_1, f_2, f_3 and f_4 achieved by the 8 solutions are 0.19, 0.20, 0.13 and 0.11, respectively.

From Table 1-3, it can be learned that the number of solutions obtained decreases as the scale of the problem increases, and so do the value ranges of objectives achieved by the solutions. Therefore, the diversity of solutions gradually decreases with the increase of the problem scale, and the difficulty of solving the problem increases with the increase of the scale of the problem. Overall, the results also demonstrate that the proposed algorithm could effectively solve the large-scale task allocation problem with even $m = 250$ and $n = 1000$, which could be significantly difficult to solve for the full search algorithms.

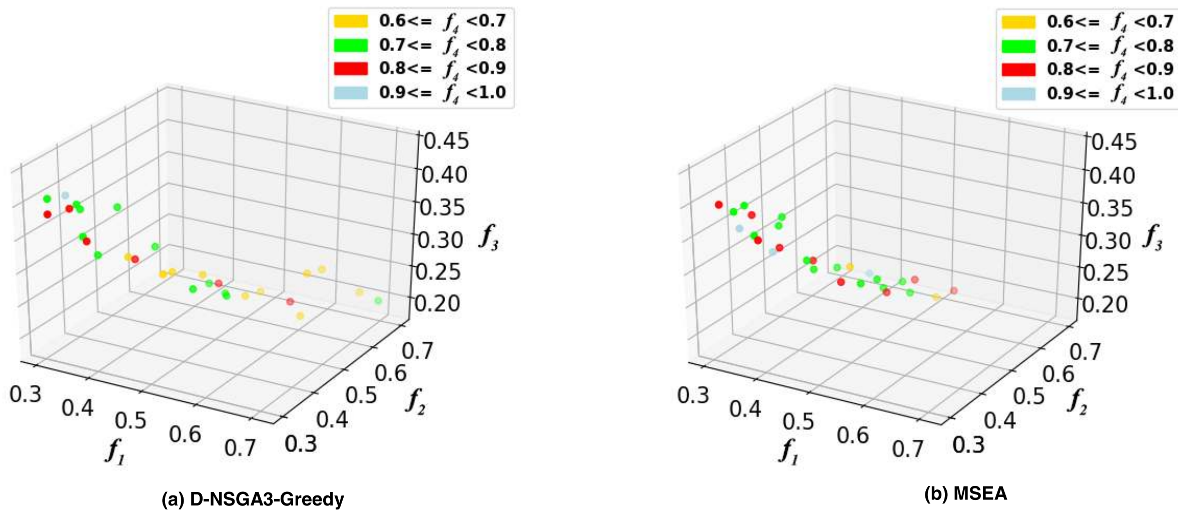

FIGURE 3. Distribution of the results for $m = 50$ and $n = 200$.

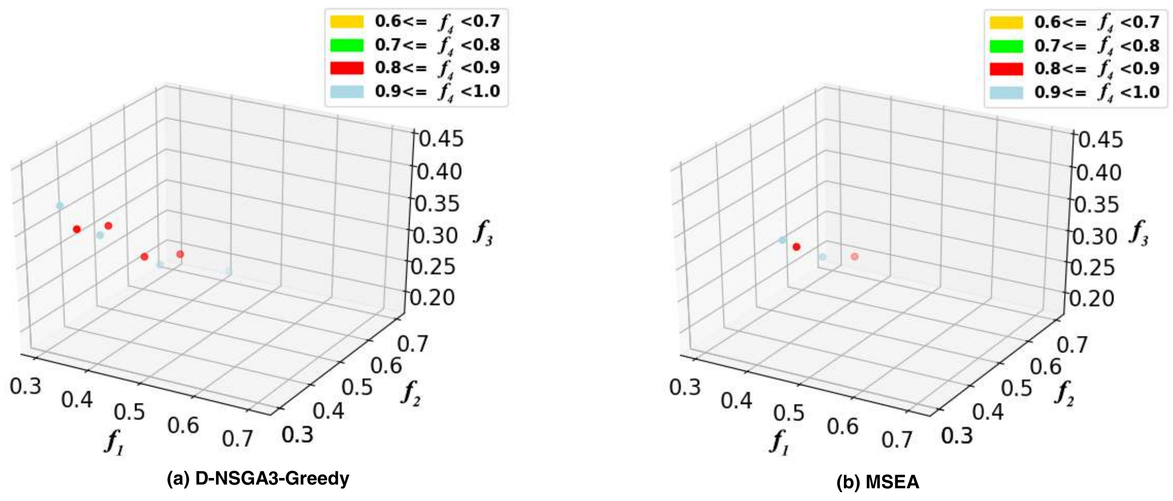

FIGURE 4. Distribution of the results for $m = 250$ and $n = 1000$.

B. ALGORITHM COMPARISON

In this section, we compare D-NSGA3-Greedy with MSEA proposed in [26] in terms of the quality and diversity of solutions, and we also investigate the efficiency of the proposed method with/without using the Master-Slave model. Table 4-6 and figure 1-3 show the comparison results of D-NSGA3-Greedy and MSEA for ($m = 10$, $n = 40$), ($m = 50$, $n = 200$) and ($m = 250$, $n = 1000$). When ($m = 10$, $n = 40$), 7 solutions found by D-NSGA3-Greedy algorithm are better than the corresponding solutions found by MSEA and 3 solutions are worse than that of MSEA. When ($m = 50$, $n = 200$), 2 solutions of D-NSGA3-Greedy algorithm are better than that of MSEA and no solution is worse than that of MSEA. When ($m = 250$, $n = 1000$), 1 solution of D-NSGA3-Greedy algorithm is better than that of MSEA and no solution is worse than that of MSEA. Under the three parameter settings, D-NSGA3-Greedy algorithm finds 29, 24, 8 solutions, respectively, while MSEA finds 27, 13, 4 solutions, respectively. As shown in figure 2-4, the solutions found by D-NSGA3-Greedy distribute more uniformly in the objective domain than the solutions found by MSEA. It demonstrates that D-NSGA3-Greedy algorithm could find more diverse solutions than MSEA, and D-NSGA3-Greedy algorithm shows stronger optimization capability than MSEA.

Table 7 shows the time cost by running the proposed method with/without using the distributed Master-Slave model (denoted as D-NSGA3-Greedy and NSGA3-Greedy, respectively). Each algorithm is executed 30 times, and average time cost is recorded. The parameter D is set to 250, and other parameters are set as default. The data are accurate to 2 decimal places. When $m = 10$ and $n = 40$, using the Master-Slave model decreases the time cost by CSS from 320.94 seconds to 261.27 seconds. The 10 agents share amount of computations costing 66.14 seconds for evaluation in total, and each agent spends 6.61 seconds on average.

TABLE 4. Comparison Results of D-NSGA3-Greedy and MSEA for m=10 and n=40.

D-NSGA3-Greedy				MSEA [26]			
f_1	f_2	f_3	f_4	f_1	f_2	f_3	f_4
0.33	0.28	0.42	0.78	--	--	--	--
0.33	0.33	0.41	0.93	**0.33**	**0.29**	**0.41**	**0.88**
0.35	0.25	0.41	0.88	--	--	--	--
0.35	0.31	0.40	0.88	**0.35**	**0.30**	**0.40**	**0.78**
0.35	0.33	0.40	0.72	0.35	0.33	0.40	0.72
0.35	0.34	0.39	0.77	0.35	0.35	0.38	0.88
0.38	0.40	0.38	0.72	0.38	0.27	0.39	0.93
				0.38	0.38	0.36	0.79
				0.38	**0.39**	**0.37**	**0.72**
0.40	**0.27**	**0.38**	**0.79**	0.40	0.28	0.38	0.79
0.40	**0.28**	**0.37**	**0.87**	0.40	0.29	0.37	0.88
0.40	**0.31**	**0.34**	**0.74**	0.40	0.33	0.34	0.93
				0.40	0.35	0.34	0.88
0.43	**0.35**	**0.33**	**0.66**	0.43	0.40	0.31	0.89
0.43	0.37	0.32	0.82				
0.43	0.43	0.32	0.70				
--	--	--	--	0.45	0.35	0.33	0.79
				0.45	0.37	0.31	0.73
				0.45	0.44	0.29	0.79
0.48	**0.40**	**0.30**	**0.64**	0.48	0.43	0.30	0.68
--	**--**	**--**	**--**	0.48	0.49	0.27	0.93
0.50	0.34	0.32	0.68	0.50	0.37	0.30	0.88
0.50	0.46	0.28	0.64	--	--	--	--
0.50	0.48	0.26	0.73				
0.50	0.51	0.25	0.89				
0.53	**0.38**	**0.29**	**0.79**	0.53	0.38	0.30	0.79
--	--	--	--	0.53	0.43	0.29	0.70
				0.53	0.45	0.27	0.79
				0.53	0.46	0.26	0.89
0.53	**0.48**	**0.25**	**0.79**	0.53	0.51	0.26	0.79
				0.53	0.55	0.25	0.89
0.55	0.45	0.26	0.70	0.55	0.50	0.25	0.73
0.55	0.51	0.24	0.66	--	--	--	--
0.55	0.56	0.23	0.68				
--	--	--	--	0.57	0.55	0.23	0.68
				0.57	0.61	0.22	0.89
0.60	0.57	0.22	0.89	--	--	--	--
0.63	0.55	0.21	0.68				
0.63	0.57	0.27	0.64				
0.63	0.62	0.26	0.64				
0.68	0.66	0.22	0.65				
0.70	0.69	0.20	0.73				

When $m = 50$ and $n = 200$, using the Master-Slave model decreases the time cost by CCS from 2225.61 seconds to 312.81seconds. The 50 agents share 1951.84 seconds of computations for evaluation from CCS, and each agent spends 39.04 seconds on average. When $m = 250$ and $n = 1000$, using the Master-Slave model decreases the time cost by CCS from 37379.89 seconds to 495.85 seconds. The 250 agents share 37032.17 seconds of computations for evaluation from CCS, and each agent spends 148.13 seconds on average. It demonstrates that the distributed model has higher efficiency in larger task allocation problems because the agents will share more computations from CCS by executing the evaluation and this is identical to the analysis in Section III.D. Moreover, for large-scale task allocation problems ($m = 250$, $n = 1000$), the distributed model enhances the computation efficiency by about two orders of magnitude.

TABLE 5. Comparison Results of D-NSGA3-Greedy and MSEA for m=50 and n=200.

D-NSGA3-Greedy				MSEA			
f_1	f_2	f_3	f_4	f_1	f_2	f_3	f_4
0.31	0.22	0.43	0.89	--	--	--	--
0.31	0.29	0.41	0.85				
0.32	0.27	0.41	0.96				
0.35	0.26	0.40	0.78				
0.37	0.22	0.40	0.87	0.37	0.32	0.37	0.88
0.37	0.30	0.35	0.90				
0.37	0.33	0.37	0.77				
0.38	0.34	0.36	0.84	0.38	0.30	0.37	0.93
0.39	0.24	0.39	0.92	--	--	--	--
0.40	0.27	0.35	0.81				
--	--	--	--	0.41	0.37	0.33	0.93
				0.41	0.38	0.34	0.86
0.43	0.29	0.32	0.93	0.43	0.30	0.34	0.88
0.43	0.31	0.34	0.74	--	--	--	--
0.44	**0.31**	**0.31**	**0.88**	0.44	0.31	0.33	0.94
0.44	0.38	0.32	0.77	--	--	--	--
--	--	--	--	0.46	0.35	0.32	0.81
				0.46	0.38	0.31	0.93
				0.46	0.40	0.30	0.81
0.47	0.32	0.32	0.81	0.47	0.36	0.30	0.87
0.47	0.41	0.28	0.91				
--	--	--	--	0.49	0.42	0.28	0.88
0.50	0.43	0.28	0.73	--	--	--	--
0.51	0.41	0.26	0.81				
0.52	**0.37**	**0.27**	**0.93**	0.52	0.40	0.27	0.93
--	--	--	--	0.52	0.41	0.27	0.81
0.54	0.46	0.25	0.82	--	--	--	--
0.54	0.47	0.24	0.91				
0.56	0.51	0.24	0.73				
0.59	0.53	0.23	0.87				
0.60	0.56	0.21	0.81				

TABLE 6. Comparison Results of D-NSGA3-Greedy and MSEA for m=250 and n=1000.

D-NSGA3-Greedy				MSEA			
f_1	f_2	f_3	f_4	f_1	f_2	f_3	f_4
0.34	0.30	0.40	0.95	--	--	--	--
0.37	0.30	0.37	0.89				
0.39	0.36	0.36	0.89				
0.40	0.32	0.36	0.96				
--	--	--	--	0.41	0.37	0.34	0.92
--	--	--	--	0.43	0.38	0.33	0.87
0.45	0.37	0.32	0.85	--	--	--	--
0.46	**0.40**	**0.30**	**0.93**	0.46	0.41	0.31	0.93
0.48	0.43	0.31	0.86	--	--	--	--
--	--	--	--	0.49	0.46	0.30	0.89
0.53	0.50	0.27	0.96	--	--	--	--

V. DISCUSSION

In this paper, we propose a distributed task allocation method based on the NSGA3 algorithm and a greedy algorithm. The main difficulty of this work is:

(1) How to handle the large amount of constraints related to time, resources, locations and functions while optimizing four objectives: for this issue, we introduce a special encoding of chromosomes which consist of ordered sequences of tasks that are able to be executed by each agent. Function constraints are handled by only encoding the tasks that are able to be executed by an

TABLE 7. Time cost by D-NSGA3-Greedy and NSGA3-Greedy.

	D-NSGA3-Greedy (s)		NSGA3-Greedy
	CCS	Agent (avg.)	(s)
(m = 10, n = 40)	**261.27**	6.61	320.79
(m = 50, n = 200)	**312.81**	39.04	2225.61
(m = 250, n = 1000)	**495.85**	148.13	37379.89

agent to the sequence of that agent. Time, resources and locations constraints are embedded into the evaluation function. When calculating the four objective values, only the tasks that can be successfully executed by agents while satisfying all constraints are taken into account. By this way, we can handle the constraints efficiently while optimizing the four objectives.

(2) How to design a greedy algorithm dedicated to NSGA3 such that it can effectively improve the local optimizing ability of NSGA3: for this issue, we design a greedy algorithm based on the chromosome structure used in NSGA3. The greedy algorithm takes the chromosome of individuals of NSGA3 as input, and it locally explores the exchanges of tasks that greedily improve the solution quality on the chromosome. To combine the greedy algorithm with NSGA3, we use a parameter D to control the frequency of calling the greedy strategy during the population evolution of NSGA3. When D is smaller, the greedy strategy will be called more frequently and better solutions would be found, but more computation and communication cost is required. In a specific scenario, we can choose proper D according to the requirement on solutions and available computation and communication resources.

Although experimental results have demonstrated the effectiveness of the proposed method, but it still has some room for improvement. For example, the agents in the proposed method are only responsible for evaluating the task allocation solutions and their computational and communication resources would not be sufficiently used in some scenarios where agents are equipped with high-performance devices. Moreover, dynamic environment change has not been fully considered in this work, e.g. new agents or tasks come over time, and some agents or tasks would be destroyed due to accidents. We will improve the proposed method to solve the two issues in future work.

VI. RELATED WORK

At present, some algorithms have been proposed for task allocation in multi-agent systems. In most of the early methods, a center node with high performance is adopted to perform all the computations and searching. Finally, the center node finds the solutions for task allocation and sends them to all agents [8], [29]. However, several drawbacks exist in these methods, such as the single point of failure and poor fault tolerance. In recent years, a large number of distributed task allocation methods have been proposed, which can be roughly divided into the following types: distributed full search algorithms [11]–[15], distributed local search algorithms [16]–[18], algorithms based on auction mechanism [19], distributed particle swarm optimization [20], and distributed ant colony algorithm [21]. Distributed full search algorithms [14] use distributed depth-first search or breadth-first search to find the optimal solution for task allocation. These methods can ensure the solution is globally optimal, but they require high communication cost and computation cost [15]. These methods are difficult to be applied in the situation that agents are equipped with low computation or communication resources. Distributed local search algorithms [17] greatly reduce the cost of communication and computation. Each agent searches locally for the most suitable task allocation scheme, and then realizes the global unification and coordination of task allocation through communication and thus to avoid conflict [18]. But it is usually difficult for such methods to ensure the diversity and quality of solutions. The algorithms based on the auction mechanism can be regarded as a special case of distributed local search algorithms, which simulate the auction mechanism from economic market [19]. Specifically, each agent bids on tasks based on their resources and capabilities and performs the tasks won from auction. Distributed ant colony algorithm/particle swarm optimization takes the agents/robotics as ants/particles, and simulates the group behavior of ants/particles when they are foraging for food to get the optimal task allocation solutions [20], [21]. The two methods utilize intelligent heuristic optimization strategies to search optimal solutions. However, it is difficult for them to deal with the large number of constraints. Besides, they are easy to trap in local optimum, and could not even get solutions that satisfy the constraint conditions. Moreover, few of existing works consider optimizing many (more than 3) objectives simultaneously together with the constraints in task allocation problems in multi-agent systems.

Overall, compared with existing distributed full search algorithms for task allocation in multi-agent systems, the proposed method has lower computational and communication cost because it uses D-NSGA3 and the greedy strategy for heuristic search instead of enumerating every possible solutions. Compared with existing distributed local search algorithms and auction mechanism-based algorithms for task allocation in multi-agent systems, the proposed method could achieve solutions with better diversity and quality because D-NSGA3 employs the non-dominated sorting on four objectives (instead of a simple sorting on the weighted sum of objectives in local search algorithms) and uses a selection operator based on a set of reference points, which is specially designed for maintaining solution diversity. Compared with existing distributed ant colony algorithms and particle swarm optimization algorithms for task allocation in multi-agent systems, the proposed method comprehensively considers a large amount of constraints related to time, resources, locations and agent functions usually involved in large-scaled multi-agent systems, and this makes the proposed method produces more reasonable and practical task allocation solutions. Moreover, the proposed algorithm considers the optimization of four widely-used objectives simultaneously, which is rarely considered in existing methods.

VII. CONCLUSION

This paper proposes a distributed task allocation algorithm called D-NSGA3-Greedy for multi-agent systems based on a many-objective evolutionary algorithm, the Master-Slave model and a greedy algorithm. D-NSGA3-Greedy is used to optimize the four widely-studied objectives, i.e., maximizing the number of successfully executed tasks, maximizing the benefits of task execution, minimizing the resource consumption, and minimizing the time cost, simultaneously. D-NSGA3-Greedy exploits the population diversity of many-objective evolutionary algorithms to improve the horizontal searching ability, and it combines the greedy algorithm to improve the vertical local optimizing ability. The experimental results show that D-NSGA3-Greedy can effectively solve large-scale task allocation problems. And compared with MSEA, D-NSGA3-Greedy can obtain better and more diverse non-dominated solutions.

REFERENCES

[1] Y. Sun, Z. Ji, Q. Qi, and H. Ma, "Bipartite consensus of multi-agent systems with intermittent interaction," *IEEE Access*, vol. 7, pp. 130300–130311, 2019, doi: 10.1109/access.2019.2940541.

[2] R. Bahnemann, D. Schindler, M. Kamel, R. Siegwart, and J. Nieto, "A decentralized multi-agent unmanned aerial system to search, pick up, and relocate objects," in *Proc. IEEE Int. Symp. Safety, Secur. Rescue Robot. (SSRR)*, Oct. 2017, pp. 123–128

[3] I. Arel, C. Liu, T. Urbanik, and A. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intell. Transp. Syst.*, vol. 4, no. 2, p. 128, 2010.

[4] L.-J. Wan, P. Y. Yao, and P. Sun, "Distributed task allocation method of manned/unmanned combat agents," *Syst. Eng. Electron.*, vol. 35, no. 2, pp. 310–316, 2013.

[5] Y. Jiang, Y. Zhou, and W. Wang, "Task allocation for undependable multiagent systems in social networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 8, pp. 1671–1681, Aug. 2013.

[6] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Trans. Ind. Inf.*, vol. 14, no. 10, pp. 4712–4721, Oct. 2018.

[7] W. Zhao, X. M. Bai, J. Ding, Z. Fang, and Z. H. Li, "A new fault diagnosis approach of power grid based on cooperative expert system and multi-agent technology," *Zhongguo Dianji Gongcheng Xuebao (Proc. Chin. Soc. Electr. Eng.)*, vol. 26, no. 20, pp. 1–8, 2006.

[8] S. D. Ramchurn, A. Farinelli, K. S. Macarthur, and N. R. Jennings, "Decentralized coordination in RoboCup rescue," *Comput. J.*, vol. 53, no. 9, pp. 1447–1461, Nov. 2010.

[9] S. D. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, and N. R. Jennings, "Coalition formation with spatial and temporal constraints," in *Proc. 9th Int. Conf. Auto. Agents Multiagent Syst.*, vol. 3, 2010, pp. 1181–1188.

[10] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, "Coalition structure generation with worst case guarantees," *Artif. Intell.*, vol. 111, nos. 1–2, pp. 209–238, Jul. 1999.

[11] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "Adopt: Asynchronous distributed constraint optimization with quality guarantees," *Artif. Intell.*, vol. 161, nos. 1–2, pp. 149–180, Jan. 2005.

[12] A. Petcu and B. Faltings, "DPOP: A scalable method for multiagent constraint optimization," *Proc. 19th Int. Joint Conf. Artif. Intell. (IJCAI)*, Acapulco, Mexico, 2005, pp. 266–271.

[13] R. Mailler and V. Lesser, "Solving distributed constraint optimization problems using cooperative mediation," *Proc. 3rd Int. Joint Conf. Auto. Agents Multiagent Syst. (AAMAS)*, 2004, pp. 438–445.

[14] B. Xie, J. Chen, and L. Shen, "Cooperation algorithms in multi-agent systems for dynamic task allocation: A brief overview," in *Proc. 37th Chin. Control Conf. (CCC)*, Jul. 2018, pp. 6776–6781.

[15] K. S. Macarthur, R. Stranders, S. D. Ramchurn, and N. Jennings, "A distributed anytime algorithm for dynamic task allocation in multi-agent systems," in *Proc. 25th AAAI Conf. Artif. Intell.*, 2011, pp. 701–706.

[16] S. Fitzpatrick, and L. Meetrens. *Distributed Sensor Networks: A Multi-agent Perspective*. Norwell, MA, USA: Kluwer, 2003.

[17] A. C. Chapman, R. A. Micillo, R. Kota, and N. R. Jenning, "Decentralised dynamic task allocation: A practical game-theoretic approach," in *Proc. 8th Int. Conf. Auto. Agents Multiagent Syst. (AAMAS)*, 2009, pp. 1–8.

[18] H. Yedidsion, R. Zivan, and A. Farinelli, "Applying max-sum to teams of mobile sensing agents," *Eng. Appl. Artif. Intell.*, vol. 71, pp. 87–99, May 2018.

[19] L. Vig and J. A. Adams, "Market-based multi-robot coalition formation," *Distrib. Auto. Robotic Syst.*, vol. 7, pp. 227–236, Jun. 2007.

[20] N. Nedjah, R. M. D. Mendonça, and L. D. M. Mourelle, "PSO-based distributed algorithm for dynamic task allocation in a robotic swarm," *Procedia Comput. Sci.*, vol. 51, pp. 326–335, Jan. 2015.

[21] W. Xiang and H. Lee, "Ant colony intelligence in multi-agent dynamic manufacturing scheduling," *Eng. Appl. Artif. Intell.*, vol. 21, no. 1, pp. 73–85, Feb. 2008.

[22] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary Computation 1: Basic Algorithms and Operators*. Boca Raton, FL, USA: CRC Press, 2018.

[23] Y.-H. Jia, W.-N. Chen, T. Gu, H. Zhang, H.-Q. Yuan, S. Kwong, and J. Zhang, "Distributed cooperative co-evolution with adaptive computing resource allocation for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 23, no. 2, pp. 188–202, Apr. 2019.

[24] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang, and J.-J. Li, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art," *Appl. Soft Comput.*, vol. 34, pp. 286–300, Sep. 2015.

[25] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, Aug. 2014.

[26] J. Zhou, X. Zhao, D. Zhao, and Z. Lin, "Distributed task allocation in multi-agent systems using many-objective evolutionary algorithm NSGA-III," in *Proc. 4th EAI Int. Conf. Mach. Learn. Intell. Commun.*, to be published.

[27] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe, "Allocating tasks in extreme teams," in *Proc. 4th Int. Joint Conf. Auto. Agents Multiagent Syst. (AAMAS)*. New York, NY, USA: ACM, 2005, pp. 727–734

[28] T.-C. Chiang. *nsga3cpp: A C++ Implementation of NSGA-III*. Accessed: Feb. 28, 2019. [Online]. Available: https://web.ntnu.edu.tw/~tcchiang/publications/nsga3cpp/nsga3cpp.htm?tdsourcetag=s_pcqq_aiomsg

[29] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artif. Intell.*, vol. 101, nos. 1–2, pp. 165–200, May 1998.

JING ZHOU was born in Qianjiang, Hubei, China, in 1981. He received the M.S. degree in intelligent systems and pattern recognition from the Huazhong University of Science and Technology, Wuhan, China, in 2009. He is currently pursuing the Ph.D. degree in system engineering with the Faculty of Management and Economics, Dalian University of Technology, China.

He has served as an Assistant Professor and a Lecturer. Since 2012, he has been a Research Assistant with the Institute of Software and Simulation, Dalian Navy Academy, China. He has published more than 20 articles and several inventions patents. His research interests include mathematical modeling and simulation, and machine learning.

XIAOZHE ZHAO was born in Dalian, Liaoning, China, in 1963. He received the B.S. degree in computer science and engineering from the Dalian Institute of Technology, China, in 1984, and the M.S. and Ph.D. degrees in systems engineering, in 1987 and 1992, respectively.

He has served as an Assistant Professor, a Lecturer, an Associate Professor, a Professor, and the Director of the Institute. He is an expert in control and information system engineering. He is currently a Doctoral Tutor with the Dalian University of Technology, China. He published more than 100 academic papers and published five books.

Dr. Zhao has received three second prizes and one third prize for National science and technology progress, and four first prizes and seven second prizes for science and technology progress at the provincial and ministerial levels.

XIAOPAN ZHANG received the B.S. degree in communication engineering, in 2001, and the M.S. and Ph.D. degrees in system engineering, in 2007, from the Huazhong University of Science and Technology, China.

From 2007 to 2010, he was a Lecturer with the Resources and Environmental Engineering School, Wuhan University of Technology, where he has been an Associate Professor with the Resources and Environmental Engineering School, since 2010. His research interests include operational research and GIS.

DONGDONG ZHAO (Member, IEEE) received the B.S. degree in computer science from the China University of Geosciences, in 2011, and the Ph.D. degree in computer application technology from the University of Science and Technology of China, in 2016. He is currently a Lecturer with the School of Computer Science and Technology, Wuhan University of Technology. His research interests include information security, privacy protection, and biometrics.

HUANHUAN LI received the B.S. and M.S. degrees in computer science from the China University of Geosciences, in 2011 and 2014, respectively. She is currently pursuing the Ph.D. degree with the School of Mechanical Engineering and Electronic Information, China University of Geosciences. Her research interests include evolutionary algorithm, artificial intelligence, and knowledge management.