Task Allocation in Multi-agent Systems Using Many-objective Evolutionary Algorithm NSGA-III

Jing Zhou[1,2](✉), Xiaozhe Zhao[1], Dongdong Zhao[3], and Zhong Lin[2]

[1] Faculty of Management and Economics, Dalian University of Technology, Dalian, China
zj_0562@163.com
[2] Operation Software and Simulation Institute, Dalian Naval Academy, Dalian, China
[3] School of Computer Science and Technology, Wuhan University of Technology, Wuhan, China

Abstract. Task allocation is an important issue in multi-agent systems, and finding the optimal solution of task allocation has been demonstrated to be an NP-hard problem. In many scenarios, agents are equipped with not only communication resources but also computing resources, so that tasks can be allocated and executed more efficiently in a distributed and parallel manner. Presently, many methods have been proposed for distributed task allocation in multi-agent systems. Most of them are either based on complete/full search or local search, and the former usually can find the optimal solutions but requires high computational cost and communication cost; the latter is usually more efficient but could not guarantee the solution quality. Evolutionary algorithm (EA) is a promising optimization algorithm which could be more efficient than the full search algorithms and might have better search ability than the local search algorithms, but it is rarely applied to distributed task allocation in multi-agent systems. In this paper, we propose a distributed task allocation method based on EA. We choose the many-objective EA called NSGA-III to optimize four objectives (i.e., maximizing the number of successfully allocated and executed tasks, maximizing the gain by executing tasks, minimizing the resource cost, and minimizing the time cost) simultaneously. Experimental results show the effectiveness of the proposed method, and compared with the full search strategy, the proposed method could solve task allocation problems with more agents and tasks.

Keywords: Multi-agent system · Task allocation · Evolutionary algorithm

1 Introduction

Along with the rapid development of Internet of Things (IoTs) and wireless communication techniques, multi-agent systems are increasingly employed in industry and military fields. In multi-agent systems, there are usually a number of devices (e.g., robots and unmanned aerial vehicles) that can execute specified tasks automatically and intelligently.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2019
Published by Springer Nature Switzerland AG 2019. All Rights Reserved
X. B. Zhai et al. (Eds.): MLICOM 2019, LNICST 294, pp. 677–692, 2019.
https://doi.org/10.1007/978-3-030-32388-2_56

Task allocation is one of the most important issues in multi-agent systems, and it directly influences the system effectiveness. However, it has been demonstrated that finding the optimal solutions of task allocation in multi-agent systems would be an NP-hard problem [1]. Several objectives need to be optimized in the problem, e.g., maximizing the number of tasks that can be successfully allocated and executed by the agents, maximizing the benefits achieved by executing tasks, minimizing the resources cost during the task execution and minimizing the time cost. In many scenarios, agents are equipped with both communication resources and computing resources, and they can cooperate with a control centre in a distributed/decentralized manner to search for the optimal solutions of task allocation more efficiently.

Presently, many methods have been proposed for distributed task allocation in multi-agent systems, and they can be mainly divided into two classes depending on while they are based on the complete/full search strategy or local search strategy [2]. The methods based on the full search strategy, e.g., [3–7], can usually find the optimal task allocation solutions but they require a large amount of computational cost and communication cost, and it will become unbearable in large-scale systems (e.g., the number of agents or tasks is larger than 200). The methods based on the local search strategy, e.g., [8–11], are usually more efficient and require less communication cost, but they cannot guarantee the quality of obtained solutions, and they would only find few solutions that are local optimal and biased towards one objective. In some scenarios, if multiple objectives should be optimized simultaneously, the utility of these algorithms would decrease significantly. Therefore, it is meaningful to find some new methods for task allocation in multi-agent systems.

Evolutionary Algorithm (EA) is a promising algorithm for optimization [12], and it has been applied to many areas, e.g., Satisfaction Problem [13], Vehicle Routing Problem [14], Dynamic Shortest Path Problem [15], and Optimal Antenna Design Problem [16]. It is demonstrated that EA could be more efficient than the full search strategy and exhibits a better global search ability and could find better solutions than the local search strategy. Moreover, in the situation that multiple objectives are required to be optimized, EA could find more nondominated solutions than the local search strategy, and it enables the capacity of EA to account for diverse requirements. However, it is rarely applied to distributed task allocation in multi-agent systems. Therefore, in this paper, we propose a distributed task allocation method based on EA. We mainly consider four objectives, and we use a many-objective EA called NSGA-III [17] because traditional multi-objective EAs might only perform well on the optimization problems with 2 or 3 objectives. Specifically, the first objective we consider is to maximize the number of tasks that can be successfully allocated and executed by agents; the second objective is to maximize the benefits gained by executing tasks; the third objective is to minimize the resource cost by executing tasks; the last objective is to minimize the maximal time cost required among all agents to finish the tasks. Note that, most of existing works only consider one or two objectives, and these four objectives are rarely optimized simultaneously. To allocate the tasks in the distributed manner, we combine NSGA-III with the Master-Slave model [18], which is a typical distributed computation model.

The rest of this paper is organized as follows. Section 2 presents the preliminary knowledge of this work; Sect. 3 shows the proposed method; Sect. 4 gives the experimental results and discussion; Sect. 5 concludes the whole work.

2 Preliminaries

In this Section, we introduce the formal description of the task allocation problem in multi-agent systems and describe the Master-Slave model.

2.1 Task Allocation Problem

Task allocation involves different factors and constraints in different scenarios [2, 19, 20], and we extract some common elements to describe the problem we will solve. Specifically, we have:

(1) **Agents:** $A = \{a_1 \ldots a_m\}$, where m is number of agents. The locations of agents are: $L_A = \{l_{a1} \ldots l_{am}\}$, where l_{ai} $(i = 1 \ldots m)$ denotes the location of the agent a_i. The amount of resources equipped by agents is: $R = \{r_1 \ldots r_m\}$, where r_i $(i = 1 \ldots m)$ is the amount of resources equipped by the agent a_i.

(2) **Tasks:** $V = \{v_1 \ldots v_n\}$, where n is the number of tasks. The locations of tasks are: $L_V = \{l_{v1} \ldots l_{vn}\}$, where l_{vi} $(i = 1 \ldots n)$ denotes the location of the task v_i. We suppose each task has an executing time limitation and we define it by the earliest start time and the latest start time of execution, i.e., $T_V = \{[low_1, up_1] \ldots [low_n, up_n]\}$. That means a valid execution of the task v_i should start at the time between low_i and up_i. The time cost by executing the tasks is $Timecost = \{timecost_1 \ldots timecost_n\}$, where $timecost_i$ $(i = 1 \ldots n)$ is the time cost by executing the task v_i. The resource cost by executing tasks is $Rescost = \{rescost_1 \ldots rescost_n\}$, where $rescost_i$ $(i = 1 \ldots n)$ is the resource cost by executing the task v_i. The amount of benefits gained by executing tasks is $Gain = \{gain_1 \ldots gain_n\}$, where $gain_i$ $(i = 1 \ldots n)$ is the amount of benefits achieved by executing the task v_i.

(3) **Task allocation:** $P = \{\pi_1 \ldots \pi_n\}$, where π_i $(i = 1 \ldots n)$ denotes the allocating of task v_i, e.g., $\pi_1 = v_1 \rightarrow a_3$ means the task v_1 is allocated to the agent a_3 for execution, and for simplicity, we directly denote it as $\pi_1 = a_3$. If a task v_j is not allocated to any agents, we denote it as $\pi_j = Null$. Similar to [19], we assume that one task can be finished by one agent individually.

(4) **Agent execution:** $Q = \{q_1 \ldots q_m\}$, where q_i $(i = 1 \ldots m)$ denotes a sequence of tasks will be executed by the agent a_i, e.g., $q_1 = v_1 v_3 v_5$ means q_1 will execute v_1, v_3 and v_5, respectively. Similar to [6, 19, 21], we suppose each agent can only execute one task at a point in time.

(5) **Objectives:** Different applications demand different optimization objectives, and in this work, we choose the following four widely studied objectives:

  (1) Maximizing the number of tasks that can be successfully allocated and executed by agents:

$$f_1 = maximize_{\forall P} \sum\nolimits_{i=1}^{n} \{\pi_i \neq Null\} \tag{1}$$

  where the notation $\{\pi_i \neq Null\}$ returns 1 if the predicate is true; otherwise it returns 0.

(2) Maximizing the benefits gained by executing tasks:

$$f_2 = maximize_{\forall P} \sum_{i=1}^{n} \{\pi_i \neq Null\} \times gain_i \tag{2}$$

(3) Minimizing the resource cost by executing tasks:

$$\begin{aligned} f_3 = minimize_{\forall Q} \sum_{i=1}^{m} \Big(travel\_cost\big(l_{ai}, l_{vq_{i1}}\big) \\ + \sum_{j=2}^{|q_i|} travel\_cost\Big(l_{vq_{i(j-1)}}, l_{vq_{ij}}\Big) + \sum_{j=1}^{|q_i|} rescost_{q_{ij}}\Big) \end{aligned} \tag{3}$$

where the notation $|q_i|$ denotes the length of the sequence q_i, and $travel\_cost\big(l_x, l_y\big)$ is the resource cost by the agent for travelling from location l_x to location l_y.

(4) Minimizing the maximal time cost by executing tasks among all agents:

$$f_4 = minimize_{\forall Q} max_{i=1}^{m} Tcost(q_i, |q_i|) \tag{4}$$

where $Tcost(q_i, |q_i|)$ is the time cost by a_i for executing the $|q_i|$ tasks in q_i, and for j = 2…$|q_i|$:

$$\begin{aligned} Tcost(q_i, j) = \max\Big\{Tcost(q_i, j-1) + travel\_time\Big(l_{vq_{i(j-1)}}, l_{vq_{ij}}\Big), low_{q_{ij}}\Big\} \\ + timecost_{q_{ij}} \end{aligned}$$

and $Tcost(q_i, 1) = travel\_time\big(l_{ai}, l_{vq_{i1}}\big) + timecost_{q_{i1}}$ and $travel\_time\big(l_x, l_y\big)$ is the time cost by the agent for travelling from location l_x to l_y. We suppose that if an agent arrives the location of a task before its earliest start time, it will wait at that location.

(6) **Constraints:** There are usually several constraints in the task allocation problem, and we mainly consider the time constraint, the resource constraint and the function constraint

(1) *Time constraint*: An agent a_i $(1 \leq i \leq m)$ can execute a task v_j successfully only if a_i arrives the location of the task and start the execution in the time interval [low_j, up_j], and for the execution sequence q_i of a_i we have:
For j = 2…$|q_i|$:

$$\begin{aligned} g_{ij}^{T} &= \text{arrival\_time}(q_i, j) \\ &= Tcost(q_i, j-1) + travel\_time\Big(l_{vq_{i(j-1)}}, l_{vq_{ij}}\Big) \leq up_{q_{ij}} \end{aligned} \tag{5}$$

and $g_{i1}^{T} = \text{arrival\_time}(q_i, 1) = travel\_time\big(l_{ai}, l_{vq_{i1}}\big) \leq up_{q_{i1}}$.

(2) *Resource constraint*: An agent a_i $(1 \leq i \leq m)$ can execute a task successfully only if the current resources loading on a_i are sufficient for travelling to the task and executing it, and for the execution sequence q_i of a_i we have:
For j = 1…$|q_i|$:

$$\begin{aligned} g_{ij}^{R} &= \text{resource\_cost}(q_i, j) \\ &= travel\_cost\left(l_{ai}, l_{vq_{i1}}\right) + \textstyle\sum_{k=2}^{j} travel\_cost\left(l_{vq_{i(k-1)}}, l_{vq_{ik}}\right) \\ &+ \textstyle\sum_{k=1}^{j} rescost_{q_{ik}} \le r_i \end{aligned} \quad (6)$$

(3) *Function constraint*: In real-world applications, different kinds of agents may have different functions/abilities, and they can only execute the tasks fitting their functions. Specifically, we have:

(a) At the view of agents: for $i = 1 \ldots m$,

$$g_i^{FA} = \{v_{i_1}, \ldots, v_{i_b}\} :\rightarrow a_i \quad (7)$$

This constraint describes that, according to the function of the agent a_i, it is able to execute the b tasks $v_{i_1}, \ldots, v_{i_b}$.

(b) At the view of tasks: for $j = 1 \ldots n$,

$$g_j^{FT} = v_j :\rightarrow \{a_{j_1}, \ldots, a_{j_c}\} \quad (8)$$

This constraint describes that, according to the requirements of the task v_j on the agent functions, it can be allocated to any agent in $\{a_{j_1}, \ldots, a_{j_c}\}$ for execution.

2.2 Master-Slave Model

In this paper, we use a widely used distributed model for EA called Master-Slave [18] for task allocation. An example of the Master-Slave model is shown in Fig. 1. It contains one master and multiple slaves. The master has high computational and communication abilities and is responsible for the evolving process of EA, e.g., it conducts the population initialization, crossover operation, mutation operation, selection operation and iteration. The slaves are mainly responsible for evaluating individuals.

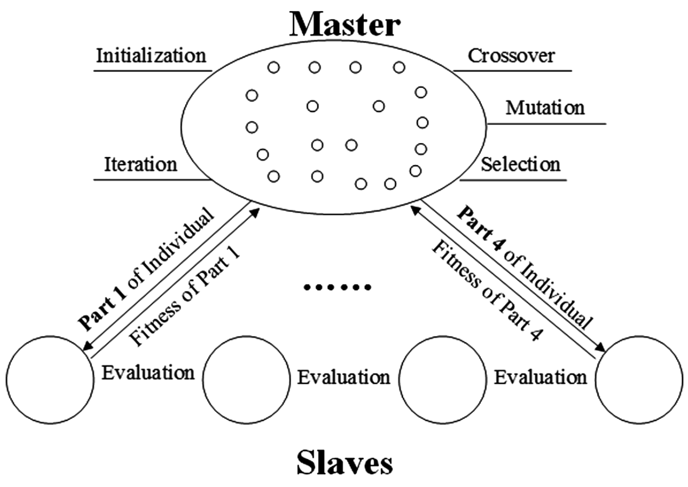

Fig. 1. The Master-Slave model for EA

The Master-Slave model is quite suitable for the scenarios that contain a control centre and a number of agents, where the control centre has high computational and communication abilities and is responsible for publishing tasks, sending orders to agents and allocating tasks to agents, and thus it corresponds to the master node. The agents are responsible for performing tasks and collecting the dynamic environment around the tasks. The effectiveness of a task allocation should be evaluated by agents because some data (e.g., the time cost, resource cost and current location) about the tasks could only be captured by agents and these data might change with the environment in some situation. In this paper, we assume that an agent could evaluate the part of solutions involves the tasks allocated to it, so each individual should be sent to multiple agents for a complete evaluation.

3 Distributed Task Allocation Based on NSGA-III

We present the distributed task allocation method based on the many-objective EA NSGA-III in this section.

```
Algorithm 1: NSGA-III [17]
Input: population size $N$, maximal iteration times $MaxIterN$, and the set of
       reference points $Ref$
Output: pareto set
1:   $i \leftarrow 1$
2:   Randomly initialize the popution $P_i$ with $N$ individuals, and evaluate the
     individuals
3:   While $i \leq MaxIterN$ do
4:     Perform the crossover, mutation operations on $P_i$ to obtain offspring $Q_i$
5:     Evaluate all the individuals in $Q_i$
6:     Conduct nondominated sorting on $P_i \cup Q_i$ based on the objective values,
       and obtain the nondominated layers $F_1 \ldots F_H$
7:     Find the $h$ s.t., $|F_1 \cup \ldots \cup F_{h-1}| < N \leq |F_1 \cup \ldots \cup F_h|$
8:     If $|F_1 + \cdots + F_h| = N$ then
9:       $P_{i+1} \downarrow F_1 \cup \ldots \cup F_h$
10:    Else
11:      $P_{i+1} \downarrow F_1 \cup \ldots \cup F_{h-1}$
12:      Select $N - |F_1 \cup \ldots \cup F_{h-1}|$ individuals from $F_h$ based on $Ref$, and add
         them into $P_{i+1}$
13:    End if
14:    If the population is convergent then
15:      Return $F_1$ as the pareto set
16:    End if
17:    $i \leftarrow i+1$
18:  End while
19:  Return $F_1$
```

3.1 Framework

NSGA-III [17] is improved from NSGA-II [22], which is a prominent multi-objective EA and has good performance on the optimization problems with 2 or 3 objectives. NSGA-III is proposed for solving the problems with more than 3 objectives. The procedure of the NSGA-III is shown in Algorithm 1. Specifically, (1) NSGA-III randomly initializes the population with N individuals, and evaluates them; (2) NSGA-III performs the crossover and mutation operation to generate offspring; (3) NSGA-III evaluates all individuals in both parent and offspring populations; (4) NSGA-III conducts nondominated sorting on the parent and offspring populations, and obtains H layers $F_1 \ldots F_H$ of nondominated fronts; (5) NSGA-III finds out the former h layers of nondominated fronts subject to $|F_1 \cup \ldots \cup F_{h-1}| < N \leq |F_1 \cup \ldots \cup F_h|$. (6) NSGA-III constructs the next generation of population: if $|F_1 \cup \ldots \cup F_h| = N$, then selects the individuals in the former h layers as the next population; otherwise, selects $N - |F_1 \cup \ldots \cup F_{h-1}|$ individuals using the reference points from the layer F_h and combines these individuals with the individuals in the former $h - 1$ layers to obtain the next population; (7) NSGA-III carries out the steps 2–6 until the population is convergent or the terminating condition (reach the maximal iteration times) is satisfied.

To find the optimal task allocation in multi-agent systems, we combine the NSGA-III algorithm with the Master-Slave model. As the control centre usually has high computational and communication abilities, it will play the role of master. The agents usually have limited computational and communication abilities, so they will play the role of slaves. In real-world applications, the cost of executing a task could only be captured by the agent that monitors the task, and the cost might change over the time. Moreover, sometimes, the environment might also change, e.g., some agents are broken, some new agents are available, or some new tasks appear. Therefore, it is more reasonable to distribute the evaluation of individuals to agents. Furthermore, we suppose that each agent can only evaluate the part of individuals that contains tasks which it can monitor and execute. This assumption is more practical, and the agents will spend less communication cost and computation cost.

Specifically, at the control centre C (Master) side:

1. C randomly initializes the population P_1, and makes the individuals distributed uniformly.
2. C divides each individual in P_i into m parts, and sends each part of the individuals to the corresponding agent for evaluation.
3. C collects the evaluation results from agents.
4. C performs the crossover, mutation operations on P_i to obtain offspring Q_i.
5. C sends each part of the individuals in Q_i to the corresponding agent for evaluation.
6. C collects the evaluation results from agents.
7. C conducts nondominated sorting on $P_i \cup Q_i$, computes nondominated layers, performs the selection operation based on reference points, and obtains the next generation of population (i.e., P_{i+1}).
8. C iteratively conducts steps 4–7 until the population is convergent or reaches the maximal iteration times.

At the agent (slave) side, each agent will receive a part of individuals sent from C, and it will evaluate the part and send back the evaluation results, i.e., the objective values. If the environment changes, the agent is also responsible for collecting the changed information and report to the control centre, and the control centre might request the re-evaluation of the population.

According to the procedure conducted by the control centre, the communication complexity at each iteration is $O(m \times |P|)$ because each individual will be divided into m parts and these m parts will be sent to m agents, respectively. Note that agents are usually much less than the tasks. For the worst case, i.e., the evolutionary process ends with the maximal iteration times, the total communication complexity of the proposed method is $O(m \times |P| \times maxIterN)$.

3.2 Chromosome Encoding, Crossover and Mutation

In EA, the individuals are represented as chromosomes, and the encoding of chromosomes influence the effectiveness of EA directly. In our work, to comprehensively contain the factors that affect the objective values, we encode a chromosome as: $x = x_1 \ldots x_m$, where x_i is a sequence of the indexes of tasks. Specifically, assume a_i is able to execute tasks $v_{i_1}, \ldots, v_{i_b}$ according to the function constraint, then x_i is a permutation of $i_1 \ldots i_b$, and it denotes the order that a_i would execute the tasks $v_{i_1}, \ldots, v_{i_b}$. For example, if there are 6 tasks and 2 agents, x = 1 3 6 2 4 5 and both a_1 and a_2 are able to execute 3 tasks, then x represents that a_1 would execute the tasks v_1, v_3 and v_6, respectively and a_2 would execute the tasks v_2, v_4 and v_5, respectively (Note that whether the agents can really execute these tasks successfully depends on the constraints are satisfied or not). By this encoding, all of the chromosomes have the same length when the function constraint is given, and the function constraint is always satisfied during the evolution. The initialization of individuals/chromosomes in the step 2 of Algorithm 1 could be quite easy. For example, we can initialize x_i of each individual as a random permutation of $i_1 \ldots i_b$. The time constraint and resource constraint will be blended in objective evaluation.

Based on the chromosome structure, we can use classic crossover and mutation strategies directly. For crossover, a split point s is first randomly chosen, and two individuals x, y exchange their parts according to s. After crossover, two new individuals x' and y' will be generated, and we have $x' = y_1 \ldots y_s x_{s+1} \ldots x_m$ and $y' = x_1 \ldots x_s y_{s+1} \ldots y_m$. For mutation, we set a parameter mr to control the mutation probability, if x_i is chosen to mutate according the probability, then we will randomly generate a permutation of $i_1 \ldots i_b$ to replace the current value of x_i. The selection operation remains the same with the original NSGA-III, please refer to [17] for details.

3.3 Evaluation

The evaluation process determines the search direction of EA, so it is important to design it carefully. In our work, we transform the four objectives in Sect. 2 into the "minimize" form and blend the constraints in objectives for evaluation. Specifically, when evaluating an individual x, we have:

(1) The first objective value of x could be calculated by:

$$f_1(x) = 1.0 - \frac{\sum_{i=1}^{m} e1(x_i)}{n} \tag{9}$$

where $x_i = i_1 \ldots i_b$ and $e1(x_i)$ is calculated by:

$$e1(x_i) = \sum\nolimits_{w=1}^{b} h(i_w)$$

where $h(i_w) = \begin{cases} 0 & \begin{array}{l} \textit{if } v_{i_w} \textit{ has been allocated}, \\ \textit{or arrival\_time}\left(x_i^E + i_w, \left|x_i^E\right| + 1\right) > up_{i_w}, \\ \textit{or resource\_cost}\left(x_i^E + i_w, \left|x_i^E\right| + 1\right) > r_i \end{array} \\ 1 & \textit{otherwise} \end{cases}$ and it judges whether v_{i_w} can be added to the current execution sequence of a_i, x_i^E (it is the empty set at the beginning) is the current execution sequence of a_i, and $x_i^E + i_w$ denotes adding i_w to x_i^E.

(2) The second objective value of x could be calculated by:

$$f_2(x) = 1.0 - \frac{\sum_{i=1}^{m} e2(x_i)}{\sum_{i=1}^{n} gain_i} \tag{10}$$

where $e2(x_i) = \sum_{w=1}^{b} h(i_w) \times gain_{i_w}$.

(3) The third objective value of x could be calculated by:

$$f_3(x) = \sum\nolimits_{i=1}^{m} \frac{e3(x_i)}{m \times max\_t} \tag{11}$$

where max_r is the maximal amount of the resources equipped by agents, $e3(x_i) = \sum_{w=1}^{k} h(i_w) \times (travel\_cost(l_{last}, l_{vi_w}) + rescost_{i_w})$, and l_{last} is the position of the last task allocated to a_i (l_{last} is the position of a_i when w = 1).

(4) The fourth objective value of x could be calculated by:

$$f_4(x) = max_{i=1}^{m} \frac{e4(x_i)}{m \times max\_t} \tag{12}$$

where max_t is the maximal time that a task could be finished, and $e4(x_i) = \sum_{w=1}^{k} h(i_w) \times (travel\_time(l_{last}, l_{vi_w}) + timecost_{i_w})$.

It is worth to mention that, not all of the tasks encoded in a chromosome could be successfully allocated to agents, and the order that agents execute tasks is implied in chromosomes. The time constraint and resource constraint are blended in by the function $h()$. By the above transformation, we can easily evaluate an individual by inputting it into the four functions in (9)–(12).

4 Experiments and Discussion

In this section, we present experimental results to show the effectiveness of the proposed method, and we compare it with a typical full search strategy.

We first randomly generate the data about agents and tasks. Specifically, we vary the number of agents (i.e., m) from 10 to 250 and the number of tasks (i.e., n) from 10 to 1000 (n is 4 times to m by default). All agents are initially equipped with the same amount of resources, i.e., $r_i = 500$ for $i = 1 \ldots m$. They are set to be able to execute 5 tasks (i.e. $|q_i| = 5$), and these tasks are randomly distributed to agents to construct the function constraints. We denote the positions of agents and tasks by 2-dimensional points (x, y), and the range of x and y is [0, 100]. We randomly generate the positions of all agents and tasks. The earliest start time low_i ($i = 1 \ldots n$) of the task v_i is randomly generated in [0, 100], and the latest start time up_i is randomly generated in $[low_i, 150]$. To ensure an agent could execute multiple tasks, the time cost by executing a task is randomly generated in [1, 10], and the resource cost is randomly generated in [0, 100]. The benefits gained by executing a task is randomly generated as a double value in [0, 1]. For simplicity, we assume that both the time cost and resource cost by travelling are positively proportional to the distance.

After generating the test data, we implement our method on the source code of NSGA-III provided by prof. Chiang [23]. We set the maximal iteration times of NSGA-III as 1000 by default. The crossover rate is set to be 1.0, which means crossover operation is certainly performed once two parents are chosen. The mutation rate is set to be 1/(the length of chromosome), which means x_i ($i = 1 \ldots m$) will be mutated with this probability once the individual x is chosen.

4.1 Varying Problem Scale

To investigate the performance of the proposed method on different scales of problems, we choose three parameter settings: (1) $m = 10$, $n = 40$; (2) $m = 50$, $n = 200$; (3) $m = 250$, $n = 1000$. After generating the agents and tasks, we use the proposed method to solve the three problems, and the results are shown in Tables 1, 2 and 3.

For the case $m = 10$ and $n = 40$, if two groups of results are close to each other (the differences between their values for the first three objectives are less than 0.02), we only present one of them. Finally, we obtain 24 groups of nondominated results, and the results are correct to three decimal places. It is shown that the minimal values of f_1, f_2, f_3, f_4 are 0.350, 0.316, 0.226, 0.703, respectively, and the proposed method can finish at most 65% of the 40 tasks.

For the case $m = 50$ and $n = 200$, we filter the results in the same way, and finally we obtain 13 groups of nondominated results. It is shown that the minimal values of f_1, f_2, f_3, f_4 are 0.465, 0.417, 0.259, 0.776, respectively, and the proposed method can finish at most 54.5% of the 200 tasks.

For the case $m = 250$ and $n = 1000$, we also filter the results but use the difference threshold 0.01, and finally we only obtain 7 groups of nondominated results. It is shown that the minimal values of f_1, f_2, f_3, f_4 are 0.532, 0.482, 0.254, 0.850, respectively, and the proposed method can finish at most 46.8% of the 1000 tasks.

Table 1. The results of the proposed method when m = 10 and n = 40

f_1	f_2	f_3	f_4
0.350	0.330	0.386	0.775
0.375	0.377	0.364	0.897
0.400	0.316	0.356	0.775
0.400	0.387	0.352	0.771
0.425	0.333	0.351	0.775
0.425	0.400	0.341	0.703
0.450	0.407	0.330	0.703
0.450	0.447	0.321	0.736
0.475	0.354	0.319	0.775
0.475	0.400	0.306	0.897
0.500	0.403	0.309	0.703
0.500	0.437	0.297	0.736
0.500	0.501	0.282	0.897
0.500	0.521	0.276	0.771
0.525	0.461	0.280	0.707
0.525	0.496	0.282	0.703
0.525	0.502	0.262	0.897
0.550	0.506	0.265	0.736
0.550	0.532	0.260	0.897
0.550	0.558	0.248	0.746
0.575	0.546	0.256	0.707
0.575	0.568	0.241	0.897
0.575	0.588	0.234	0.746
0.600	0.624	0.226	0.746

Table 2. The results of the proposed method when m = 50 and n = 200

f_1	f_2	f_3	f_4
0.465	0.417	0.358	0.847
0.475	0.453	0.338	0.837
0.490	0.493	0.320	0.804
0.500	0.457	0.317	0.956
0.505	0.427	0.331	0.956
0.525	0.451	0.321	0.847
0.530	0.484	0.303	0.956
0.530	0.520	0.301	0.778
0.550	0.487	0.298	0.956
0.555	0.563	0.279	0.776
0.565	0.516	0.277	0.956
0.575	0.541	0.267	0.847
0.585	0.593	0.259	0.776

By comparing the three experiments, we find that the solutions found by the proposed method have worse minimal objective values on f_1, f_2, f_4 when m and n increase, and the three objectives might be more difficult to optimize in this case. But the optimization of f_3 seems have no obvious relationship with the increase of m and n. Moreover, we find that the diversity of solutions found by the proposed method decreases with the increase of m and n as less results have been filtered out.

Table 3. The results of the proposed method when m = 250 and n = 1000

f_1	f_2	f_3	f_4
0.532	0.482	0.293	0.931
0.544	0.501	0.286	0.882
0.555	0.506	0.277	0.892
0.561	0.518	0.275	0.850
0.573	0.529	0.266	0.860
0.578	0.540	0.259	0.931
0.591	0.556	0.254	0.850

Table 4. Results for $n/m = 1$

f_1	f_2	f_3	f_4
0.100	0.019	0.101	0.864
0.100	0.019	0.109	0.784
0.100	0.019	0.115	0.836
0.100	0.019	0.124	0.755
0.200	0.046	0.077	0.784
0.200	0.046	0.083	0.644

Table 5. Results for $n/m = 3$

f_1	f_2	f_3	f_4
0.133	0.163	0.367	0.901
0.167	0.143	0.361	0.901
0.167	0.211	0.341	0.901
0.200	0.157	0.363	0.861
0.200	0.197	0.332	0.764
0.200	0.265	0.312	0.856
0.233	0.156	0.342	0.901
0.233	0.246	0.313	0.675
0.233	0.311	0.288	0.856
0.267	0.180	0.338	0.861
0.267	0.210	0.313	0.732
0.267	0.249	0.308	0.675
0.267	0.290	0.282	0.764
0.267	0.359	0.278	0.856
0.300	0.191	0.333	0.861
0.300	0.259	0.293	0.675
0.300	0.304	0.271	0.675
0.333	0.280	0.275	0.901
0.333	0.318	0.255	0.901
0.367	0.233	0.290	0.901
0.367	0.278	0.268	0.697
0.367	0.362	0.240	0.901
0.367	0.400	0.226	0.675
0.367	0.442	0.220	0.675
0.400	0.288	0.261	0.697
0.400	0.338	0.244	0.697
0.400	0.444	0.217	0.675
0.433	0.301	0.245	0.901
0.433	0.351	0.247	0.656
0.433	0.423	0.203	0.901
0.467	0.361	0.242	0.656

Table 6. Results for $n/m = 5$

f_1	f_2	f_3	f_4
0.560	0.608	0.358	0.768
0.580	0.608	0.323	0.877
0.600	0.604	0.330	0.876
0.600	0.616	0.309	0.768
0.600	0.645	0.301	0.768
0.620	0.623	0.323	0.756
0.620	0.627	0.298	0.768
0.620	0.651	0.280	0.756
0.620	0.661	0.260	0.877
0.640	0.640	0.260	0.756
0.640	0.665	0.240	0.877
0.660	0.623	0.269	0.876
0.660	0.672	0.260	0.748
0.660	0.692	0.221	0.877
0.680	0.683	0.214	0.675
0.680	0.718	0.203	0.675
0.700	0.674	0.235	0.675
0.720	0.685	0.208	0.756
0.740	0.718	0.180	0.675
0.760	0.734	0.175	0.675

4.2 Varying *n/m*

In this section, we conduct three experiments to show the influence of the ratio *n/m* on the performance of the proposed method. In these experiments, *m* (i.e., the number of agents) is set to 10.

In the first experiment, we set *n* as 10 and *n*/*m* is 1, and the results are shown in Table 4. We filter the results using the difference threshold 0.005, and only 6 groups of data are obtained. The minimal values of f_1, f_2, f_3, f_4 are 0.1, 0.019, 0.077, and 0.644, respectively. The agents can finish at most 90% of the 10 tasks.

In the second experiment, we set *n* as 30 and *n*/*m* is 3, and the results are shown in Table 5. We filter the results using the difference threshold 0.02, and 31 groups of data are obtained. The minimal values of f_1, f_2, f_3, f_4 are 0.133, 0.143, 0.203, and 0.655, respectively. The agents can finish at most 86.7% of the 30 tasks.

In the third experiment, we set *n* as 50 and *n*/*m* is 5, and the results are shown in Table 6. We filter the results using the difference threshold 0.02, and 20 groups of data are obtained. The minimal values of f_1, f_2, f_3, f_4 are 0.560, 0.604, 0.175, and 0.675, respectively. The agents can finish at most 44% of the 50 tasks.

By comparing the results from the three experiments, we find that the objectives f_1 and f_2 seem more difficult to optimize when *n/m* increases, but no obvious rules about the optimization of f_3 and f_4 has been found.

4.3 Comparison

In this section, we compare the proposed method with the full search strategy, which simply enumerates every possible task allocation solution and records the best ones. As there are so many optimal solutions that we cannot present them one by one in this paper, so we only present those results that are better and closest to the results of the proposed method. We choose 5 parameter settings, i.e., (1) $m = 5$, $n = 5$; (2) $m = 5$, $n = 10$; (3) $m = 5$, $n = 20$; (4) $m = 10$, $n = 10$; (5) $m = 10$, $n = 20$. The results are shown in Tables 7, 8 and 9, and we do not present the comparison results for ($m = 5$, $n = 20$) and ($m = 10$, $n = 20$) because the full search strategy did not finish execution in 1 h (while the proposed method finished in 5 min). The results of the proposed method are filtered using the difference threshold 0.02 for $m = 5$ and 0.005 for $m = 10$.

Table 7. The results of the proposed method and the full search strategy when $m = 5$ and $n = 5$

Proposed method				Full search			
f_1	f_2	f_3	f_4	f_1	f_2	f_3	f_4
0.000	0.000	0.180	0.656	0.000	0.000	**0.159**	0.656

Table 8. The results of the proposed method and the full search strategy when $m = 5$ and $n = 10$

Proposed method				Full search			
f_1	f_2	f_3	f_4	f_1	f_2	f_3	f_4
0.300	0.355	0.243	0.753	0.300	0.355	**0.231**	0.753
0.300	0.355	0.265	0.715	0.300	0.355	0.265	0.715
0.400	0.415	0.227	0.638	0.400	0.415	0.227	0.638

Table 9. The results of the proposed method and full search strategy when $m = 10$ and $n = 10$

Proposed method				Full search			
f_1	f_2	f_3	f_4	f_1	f_2	f_3	f_4
0.100	0.019	0.101	0.864	0.100	0.019	**0.085**	**0.644**
0.100	0.019	0.109	0.784	0.100	0.019	**0.085**	**0.644**
0.100	0.019	0.115	0.836	0.100	0.019	**0.085**	**0.644**
0.100	0.019	0.124	0.755	0.100	0.019	**0.085**	**0.644**
0.200	0.046	0.077	0.784	0.200	0.046	**0.064**	**0.644**
0.200	0.046	0.083	0.644	0.200	0.046	**0.064**	0.644

By comparing the results of the proposed method with the full search strategy, we find that the proposed method could find solutions that are close to the global optimal solutions found by the full search strategy. For $m = 5$ and $n = 5$, the difference between the solution of the proposed method and the corresponding optimal solution found by the full search strategy is 0.021. For $m = 5$ and $n = 10$, the average difference between the solutions of the proposed method and the corresponding optimal solutions found by the full search strategy is 0.004, the maximal difference is 0.012, and the minimal difference is 0. For $m = 10$ and $n = 10$, the average difference between the solutions of the proposed method and the corresponding optimal solutions found by the full search strategy is 0.139, the maximal difference is 0.221, and the minimal difference is 0.019. Though, the full search strategy can find the best results but it can only solve small problems, and it could not be used when n or m is not less than 20. On the contrast, the proposed method could solve the problems even with $m \geq 250$ and $n \geq 1000$, so it would have better utility on solving large problems.

5 Conclusions

In this paper, we propose a distributed method for task allocation based on NSGA-III. Specifically, NSGA-III is combined with the Master-Slave model, which is a classical distributed model for EA. The control centre in multi-agent systems plays the master rule, and the agents play the slave rule. Experimental results show that, the proposed method could be used for searching optimal task allocation solutions even when the number of agents (i.e., m) is not less than 250 and the number of tasks (i.e., n) is not less than 1000, while the full search strategy is only effective when $m < 20$ or $n < 20$. It demonstrates that the proposed method would have better utility on solving large problems.

In future work, we attempt to combine NSGA-III with other distributed models, e.g., the Island-cellular hybrid model. We will also try to solve the task allocation problems with ordering constraints.

Acknowledgments. This work was partially supported by the National Natural Science Foundation of China (Grant No. 71701208).

References

1. Korsah, G.A., Stentz, A., Dias, M.B.: A comprehensive taxonomy for multi-robot task allocation. Int. J. Robot. Res. **32**(12), 1495–1512 (2013)
2. Nunes, E., Manner, M., Mitiche, H., Gini, M.: A taxonomy for task allocation problems with temporal and ordering constraints. Robot. Auton. Syst. **90**, 55–70 (2017)
3. JayModi, P., Shen, W., Tambe, M., Yokoo, M.: ADOPT: asynchronous distributed constraint optimization with quality guarantees. Artif. Intell. **161**(1), 149–180 (2005)
4. Petcu, A., Faltings, B.: DPOP: a scalable method for multiagent constraint optimization. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), Acapulco, Mexico, pp. 266–271 (2005)
5. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), pp. 438–445 (2004)
6. Macarthur, K.S., Stranders, R., Ramchurn, S., Jennings, N.: A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In: Twenty-Fifth AAAI Conference on Artificial Intelligence, pp. 701–706 (2011)
7. Xie, B., Chen, J., Shen, L.: Cooperation algorithms in multi-agent systems for dynamic task allocation: a brief overview. In: 2018 37th Chinese Control Conference (CCC), pp. 6776–6781 (2018)
8. Fitzpatrick, S., Meetrens, L.: Distributed Sensor Networks: A Multi-agent Perspective. Kluwer Academic, Dordrecht (2003)
9. Chapman, A.C., Micillo, R.A., Kota, R., Jenning, N.R.: Decentralised dynamic task allocation: a practical game-theoretic approach. In: Proceedings of 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), pp. 1–8 (2009)
10. Fitzpatrick, S., Meertens, L.: Distributed coordination through anarchic optimization. In: Lesser, V., Ortiz, C.L., Tambe, M. (eds.) Distributed Sensor Networks: A Multiagent Perspective. MASA, vol. 9, pp. 257–295. Springer, Boston (2003). https://doi.org/10.1007/978-1-4615-0363-7_11
11. Yedidsion, H., Zivan, R., Farinelli, A.: Applying max-sum to teams of mobile sensing agents. Eng. Appl. Artif. Intell. **71**, 87–99 (2018)
12. Zhang, Q., Li, H.: MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans. Evol. Comput. **11**(6), 712–731 (2007)
13. Gottlieb, J., Marchiori, E., Rossi, C.: Evolutionary algorithms for the satisfiability problem. Evol. Comput. **10**(1), 35–50 (2002)

14. Yi, R., Luo, W., Bu, C., Lin, X.: A hybrid genetic algorithm for vehicle routing problems with dynamic requests. In: Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2017), Honolulu, Hawaii, USA, pp. 1–8 (2017)
15. Zhu, X., Luo, W., Zhu, T.: An improved genetic algorithm for dynamic shortest path problems. In: Proceedings of the 2014 IEEE Congress on Evolutionary Computation, Beijing, China, 6–11 July, pp. 2093–2100 (2014)
16. Hu, C., Zeng, S., Jiang, Y., Sun, Y., Jiang, Y., Gao, S.: A robust technique without additional computational cost in evolutionary antenna optimization. IEEE Trans. Antennas Propag. **67**(04), 1–10 (2019)
17. Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: solving problems with box constraints. IEEE Trans. Evol. Comput. **18**(4), 577–601 (2014)
18. Gong, Y., Chen, W., Zhan, Z., et al.: Distributed evolutionary algorithms and their models: a survey of the state-of-the-art. Appl. Soft Comput. **34**, 286–300 (2015)
19. Scerri, P., Farinelli, A., Okamoto, S., Tambe, M.: Allocating tasks in extreme teams. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 727–734 (2005)
20. Gini, M.: Multi-robot allocation of tasks with temporal and ordering constraints. In: Thirty-First AAAI Conference on Artificial Intelligence, pp. 4863–4869 (2017)
21. Ramchurn, S.D., Polukarov, M., Farinelli, A., Truong, C., Jennings, N.R.: Coalition formation with spatial and temporal constraints. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, pp. 1181–1188 (2010)
22. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Schoenauer, M., et al. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 849–858. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45356-3_83
23. Chiang, T.-C.: nsga3cpp: a C++ implementation of NSGA-III. https://web.ntnu.edu.tw/~tcchiang/publications/nsga3cpp/nsga3cpp.htm?tdsourcetag=s_pcqq_aiomsg. Accessed 28 Feb 2019