Complex & Intelligent Systems (2021) 7:873–890
https://doi.org/10.1007/s40747-020-00252-2

ORIGINAL ARTICLE

Particle swarm optimization algorithm for the optimization of rescue task allocation with uncertain time constraints

Na Geng[1] · Zhiting Chen[1] · Quang A. Nguyen[2] · Dunwei Gong[3]

Received: 5 July 2020 / Accepted: 4 December 2020 / Published online: 5 January 2021

Abstract
This paper focuses on the problem of robot rescue task allocation, in which multiple robots and a global optimal algorithm are employed to plan the rescue task allocation. Accordingly, a modified particle swarm optimization (PSO) algorithm, referred to as task allocation PSO (TAPSO), is proposed. Candidate assignment solutions are represented as particles and evolved using an evolutionary process. The proposed TAPSO method is characterized by a flexible assignment decoding scheme to avoid the generation of unfeasible assignments. The maximum number of successful tasks (survivors) is considered as the fitness evaluation criterion under a scenario where the survivors' survival time is uncertain. To improve the solution, a global best solution update strategy, which updates the global best solution depends on different phases so as to balance the exploration and exploitation, is proposed. TAPSO is tested on different scenarios and compared with other counterpart algorithms to verify its efficiency.

Keywords Robot rescue · Task allocation · PSO · Interval · Constraint

Introduction

The development of the global economy has financially benefited many countries in the world, but also initiated numerous environmental issues. One of its biggest side effects could be the raise of disasters that critically damage human's life, properties, and production [1]. Many disasters have natural causes, for instance: earthquakes, hurricanes, and floods. Many others are human-caused, or non-natural, disasters such as explosions, nuclear incidents, and building collapses. Sometimes, the disasters occurred in a very abrupt and erratic way, sometimes led to massive havoc, whilst their sources and causes were unforeseeable. Therefore, the post-catastrophe support and rescue are equally important as the disaster prediction and warning process. Autonomous and intelligent assistive systems have been prevalent in carrying on supportive missions in many hostile environments after disasters [2–4]. Non-human agents such as robots are increasingly exploited in such situations, especially in working in toxic and hazardous zones. Improving robots' effectiveness and efficiency in rescuing becomes an interesting research area.

The research scopes and ideas in robot rescue are diverse, two of the topics most focused on could be: rescue structure design, and motion model and control [5]. The first one considers the big picture by designing an overall structure for robots to perform a series of rescuing tasks [5–8], for example: walking, driving, or delivering medical equipment and food. The second group focuses on the details of each task and aims at improving the robots' speed, accuracy, and stability [9–11]. Significant studies in the latter one have been published, in which several robot control algorithms were proposed, for instance: simultaneous localization and mapping (SLAM) [12], particle filter (PF) [13], Kalman filter (KF) [13, 14], proportional integral derivative (PID) controller [15, 16], and sliding mode [17]. Those algorithms are considered the backbone of robot rescue performance theory these days.

Although the above methods can obtain valuable results in many simulated situations when the time factor is

✉ Dunwei Gong
dwgong@vip.163.com

1 School of Electrical Engineering and Automation, Jiangsu Normal University, Xuzhou 221116, China

2 School of Computer Science, Coventry University, Coventry CV1 5FB, UK

3 School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221008, China

neglected, they are unable to address some practical issues. In simulated cases, the key questions for designing rescue and controlling robots would be and "How to pass stability through an uneven terrain?", "How can the robots smoothly follow the setting rescue route?", "How to rise the robot system's reliability and survivability effectively by multi-sensor information fusion" When the survival time is tight, for example when some seriously injured people are trapped in a place after a chemical fire accident, there would be more complicated questions such as "What is the rescue route?", "How can the survival time be estimated?", "What is the priority order of tasks?" and "How can the rescue route be altered if some task time values are changed?", these problems can be treated as a task allocation problems. Very limited evidence can be found in the literature that rigorously assesses the time factor in robot rescue. Some of our previous researches show positive outcomes in handling complex rescue routes in real-world situations with an assumption that survival time values are stable [18, 19]. Still, estimating the survival time in destructive locations remains a challenging issue.

It is noticed that the changing rate of survival time is not uniform under the condition of emergency rescue but presents as an exponential function with respect to time [20]. Different physical qualities and environments make the survival time uncertain while considering that it only generates minor differences in the survival time value, the value can be treated as an interval without loss of generality. That important evaluation suggested that the change of survival time can be modelled by establishing an interval function of survival time changing over time. With few related studies, and considering that rescue after a disaster is a key issue for security and emergency management, research on the rescue problem with uncertain deadlines is essential.

Based on the above analysis, the rescue task allocation problem—how to rescue survivors within an uncertain and limited survival time—is studied in this paper. Rescue task allocation actually is a nondeterministic polynomial hard (NP-hard) problem, its solving method is much complex, one common-used way to solve it is intelligent optimization algorithms, such as particle swarm optimization (PSO) algorithm, genetic algorithm (GA) and so on.

In light of PSO algorithm is simple to implement, has fewer tuning parameters and fast convergence speed, and can obtain the global optimal or suboptimal solution of the problem as well [19]. Moreover, the PSO algorithm and its variants require fewer evolutionary populations than other swarm intelligence algorithms. Considering the numerous advantages of PSO algorithm, PSO algorithm is employed to solve this problem.

This paper proposes an innovative robot rescue task allocation algorithm that robust in real-world situations when survival time is limited and uncertain. Inheriting the advantages of PSO algorithm, that algorithm establishes a mathematical model for task allocation variable, thus named TAPSO (task allocation PSO). TAPSO tackles this task allocation problem by taking the number of successful survivors (tasks) as an objective function. Moreover, TAPSO improves the particle decode method in the original PSO algorithm to define the rescue route, also upgrades the global best update method by estimating the differences in different evolutionary phases between it and local best solutions.

The organization of the rest of this paper is as follows: The next section reviews some researches on robot rescue task allocation topic. Section 3 defines the particular problems that need to be solved before converting them into a mathematical domain; followed by a demonstration of the TAPSO algorithm in Sect. 4. Section 5 provides the simulation results and analysis, whilst the conclusion and future work are presented in the final section.

Related work

A wide variety of approaches have been reported for solving the problem of rescue task allocation and achieved meaningful results. Traditional methods of addressing task allocation problems include behavior-based algorithms [21], market-based algorithms [22], linear planning methods [23], and intelligent optimization algorithms [24–26]. Technically, the behavior-based algorithm has the advantages of real-time, fault-tolerance, and robustness, but unable to obtain the global optimal solutions. Moreover, market-based algorithms are suitable for solving the task allocation problem in small- and medium-sized heterogeneous distributed collaborative robots, whilst the optimal global solution is not always be computed in reasonable time; unfortunately, it is resource-consuming—once communication is interrupted, its performance drops significantly. Linear planning methods apply matrix operations for presenting robots' information and batch computing for a robot route, while the numbers of robots and tasks are considerable, the computing expense grows exponentially. Further, other hybrid linear planning methods [27–29] can find the optimal solution but are inefficient when the scale of the problem is large. Intelligent algorithms primarily employ GA, PSO, and other algorithms to solve the problem, several empirical reports suggested that this type of algorithms could be an answer for the perplexing issues of low convergence speed and easily converge at a local optimal solution [24–26].

Based on the communication topologies of robots, existing methods of task allocation can be classified into two categories: centralized and distributed. The centralized algorithm has a manager reacting like a server to assign tasks to each robot, and it can achieve the optimal solution; however, the performance of a centralized system deteriorates when the number of tasks is large, the computational load becomes heavy; therefore, a centralized algorithm is suitable for solving such task allocation problems where it is easy to obtain information about the environment and tasks for a relatively small scale. In contrast, the distributed algorithm has no manager; a few robots cannot communicate directly with each other, and the task allocation scheme is determined by the cooperation of robots that can communicate with each other. The distributed method can effectively improve the shortages of the centralized controller with heavy loads and poor fault tolerance; however, the communication cost of the distributed robot system is high, and the algorithm easily falls into the local optimum owing to the lack of global information. The following relevant literature offers supporting evidence.

In [30], Whitbrook et al. modified the performance impact (PI) task-allocation algorithm with ε-greedy and softmax auction selection methods to explore assignments with less rescue time; furthermore, in [31] the researchers presented a new algorithm based on the work in [30] to solve the problem of being trapped into the local minima and a static structure. To maximize the number of task allocations in a multi-robot system under strict time constraints, Turner et al. [32] proposed an effective algorithm to improve the solutions' performance. However, they primarily focus on the application of market-based algorithms rather than generating global optimal solutions. Robot rescue task allocation is an optimization problem in reality, an optimal allocation strategy can help rescue effectively.

Different from market-based algorithm, PSO, an centralized algorithm, can offer the global optimal solution, and it has been implemented to the field of task allocation, for example, Yu [33] presented an improved particle swarm optimization (IPSO) algorithm to improve the efficiency of resource scheduling, and this IPSO algorithm can overcome the problem of premature; Nethravathis et al. [34] proposed a permutation optimization strategy based on PSO algorithm to solve the resource sharing among device-to-device communication problem; Lin et al. [35] illustrated a new group method to divide rescue tasks into groups according to their distances, and employed an improved PSO algorithm to assign the grouped tasks to robots, results indicated that the proposed method can increase the success rate of rescue; Singh et al. [36] presented a novel PSO algorithm for solving multi-objective flexible job-shop scheduling problem with the goal of finding approximations of the optimal solutions, and its results verified its effectiveness. From the above studies, PSO-based algorithms are effective to gain the optimal solutions, However, they are centralized algorithms, the global information is needed to present the optimal solution.

At present, Unmanned Aerial Vehicle (UAV) detection technology is a relatively mature method [37], accordingly, in this paper, the environmental information after a disaster is assumed to be known in advance. Because of the noise and other communication interference issues, the information is uncertain, and the interval is reasonable as a representation of the data. Further, the data obtained from the environment after a disaster are the survival time of each survivor, which are critical constraints for the robots to perform the rescue mission, as a result, the survival time is treated as an uncertain constraint. Hence, the problem of rescue task allocation with uncertain constraints is considered in this paper.

Description and modeling

Problem description

In this paper, uncertainty refers to the situation where the tasks' survival time can only be estimated from the detected data, which we consider an interval based on the experience—a constraint when the robots provide emergency support to the survivors.

The ultimate target of the rescue task allocation is maximizing the number of survivors rescued (successful tasks). This paper focuses on solving the problem of uncertainty survival time constraints. For simplicity, the following assumptions are made:

1. Robots: all robots are identical. Their batteries are sufficient and their communication is maintained during the rescue process. A robot uses an equal amount of time on completing a rescue mission, once it reaches a survivor's location.
2. Survivors: their survivor times are different but their locations are known in advance and remain unchanged during the rescue process. Each survivor can only be rescued by one robot.
3. Rescue route: a robot takes a rescue route in which it can rescue several survivors before returning to its initial position. The rescue routes of the two robots are not overlapped.
4. If a robot can reach a survivor before his survival time ends, a successful task is recorded. Otherwise, it is a failed task.

Accordingly, the task allocation problem in this study can be described as follows: After a disaster, the location and survival time of each survivor is acknowledged, each robot needs to construct a route to accomplish the rescue process. Under the time constraints (limited survival time), the robot should maximize the number of overall successful tasks, so that the successfully rescued survivors are maximum.

Problem modeling

Given that N tasks are allocated to M robots, the related notations are used in this research, which are listed in Table 1.

To formulate the problem mathematically, a set of M rescued routes are represented by $\text{rs}_1\ \text{rs}_2\ \cdots\ \text{rs}_i\ \cdots\ \text{rs}_M$, where $\text{rs}_i = (\text{rs}_{i1}, \text{rs}_{i2}, \cdots, \text{rs}_{ij}, \cdots, \text{rs}_{i\lceil N/M \rceil})$, and the route formed by all robots' rescue routes is $\text{RS} = [\text{rs}_1\ \text{rs}_2\ \cdots\ \text{rs}_i\ \cdots\ \text{rs}_M]^T$.

As depicted in Fig. 1, there are 13 survivors and 3 robots in an environment after a disaster, and their rescue route is represented as. $\text{RS} = \begin{bmatrix} \text{rs}_1 \\ \text{rs}_2 \\ \text{rs}_3 \end{bmatrix} = \begin{bmatrix} 2, 6, 10, 0, 0 \\ 7, 4, 12, 3, 9 \\ 1, 11, 5, 8, 13 \end{bmatrix}$, where the first robot executes the rescue in the sequential order of $T_2 \to T_6 \to T_{10}$, the second robot's sequential order is $T_7 \to T_4 \to T_{12} \to T_3 \to T_9$, and the last route for the third robot is $T_1 \to T_{11} \to T_5 \to T_8 \to T_{13}$, where 0 in RS is to complete the matrix. The routes for all the three robots are depicted in Fig. 1, where the arrows illustrate the direction that robots are heading.

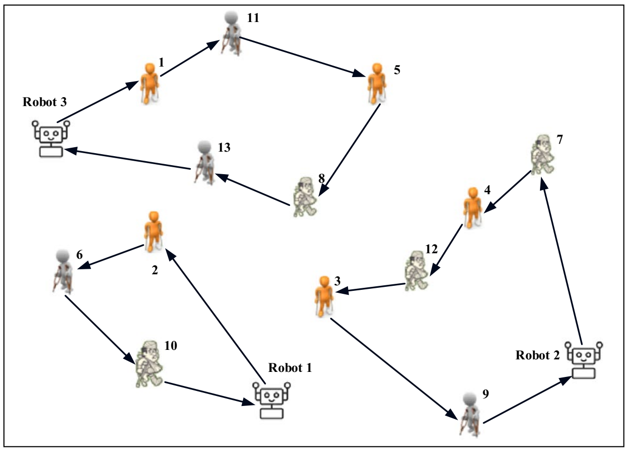

Fig. 1 Rescue environment with 13 survivors and 3 robots

Accordingly, the survivor's initial survival time is denoted as $\sigma_{0j} = [\sigma_j - \alpha, \sigma_j + \alpha]$. Survival time changes over

Table 1 The notations in this paper

Notation	Meaning
M	Number of robots, with each being indexed by $i = 1,2,\ldots,M$
N	Number of survivors (tasks), with each being indexed by $j = 1,2,\ldots,N$
RS	Rescue sequence with the matrix of $M * \lfloor N/M \rfloor$
S_R	Reference rescue sequence with the matrix of $1 * N$
R_i	i-th robot
rs_i	Rescue sequence of the i-th robot
T_j	j-th task
rs_{ij}	j-th task of the i-th robot
σ_{ij}	Deadline of T_j when the i-th robot reaches to it
Δ	Threshold of survival time
σ_{ij}^-	Lower bound of survival time when the i-th robot reaches to T_j
σ_{ij}^+	Upper bound of survival time when the i-th robot reaches to T_j
σ_j	Survival time of the j-th task
α	A constant setting artificially
σ_{j0}	Initial survival time of the j-th task detected by UAVs in advance
t_{ij}	The elapsed time for the i-th robot reaching rescue T_j
y_{ij}	Successful rescued number of tasks by the i-th robot, if T_j is rescued by the i-th robot, y_{ij} is 1; otherwise, 0
SN_i	The actual number of R_i assigned
F	Total number of successful tasks rescued by all robots
F^-	Lower bound of F
F^+	Upper bound of F
X_i	Location of the i-th particle
x_{ij}	Location of the j-th dimension of the i-th particle
V_i	Velocity of the i-th particle
ν_{ij}	Velocity of the j-th dimension of the i-th particle
l_{best}	Local best solution
g_{best}	Global best solution for the particle swarm

time [20], when time t_{ij} elapses, the survival time of task T_j when the i-th robot R_i reaching it, σ_{ij}, can be represented as follows:

$$\sigma_{ij} = \left[\sigma_{ij}^-, \sigma_{ij}^+\right] = \left[(\sigma_i - \alpha)e^{-\gamma t_{ij}}, (\sigma_i + \alpha)e^{-\gamma t_{ij}}\right] \tag{1}$$

where γ is a real number between 0 and 1, based on the result in reference [20]; in this paper γ= 0.037,$t_{ij} = \text{dis}_{ij}/v_r$, dis_{ij} is the total distance that robot R_i goes from its initial position to task T_j.

Each robot performs its respective tasks, and the merit is estimated by the number of successful tasks. Accordingly, each robot needs to construct an optimal route to gain the maximal number of successful tasks (successfully rescued survivors) under the time constraints.

Therefore, the objective in this paper is to maximize the number of successful tasks, under the previous assumptions and notations, the mathematical model is constructed as follows:

$$\text{Max}[F^-(y_{ij}), F^+(y_{ij})] \tag{2}$$

s.t.

$$\sum_{i=1}^{M} y_{0i} = 1, \quad i = 1, 2, \dots, N \tag{3}$$

$$\begin{aligned} &\sum_{j=1}^{M} y_{ij}^+ \le 1, \ i = 1, 2, \dots, N \\ &\sum_{j=1}^{M} y_{ij}^- \le 1, \ i = 1, 2, \dots, N \end{aligned} \tag{4}$$

$$\bigcap \text{rs}_i = \emptyset, \ i = 1, 2, \dots, M \tag{5}$$

$$y_{ij}^+, y_{ij}^- \in \{\ 0,\ 1\ \},\ i,j = 1, 2, \dots, N, i \ne j \tag{6}$$

$$\text{SN}_i \le N, \ i = 1, 2, \dots, M \tag{7}$$

where

$$\begin{aligned} F^-(y_{ij}^-) &= \sum_{i=1}^{M}\sum_{j=1}^{SN_i} y_{ij}^- \\ F^+(y_{ij}^+) &= \sum_{i=1}^{M}\sum_{j=1}^{SN_i} y_{ij}^+ \end{aligned}, \quad \begin{aligned} y_{ij}^- &= \text{sgn}(\max(0, \sigma_{ij}^- - \Delta)) \\ y_{ij}^+ &= \text{sgn}(\max(0, \sigma_{ij}^+ - \Delta)) \end{aligned}, \text{ and } \text{sgn}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}.$$

,

.

Equation (2) maximizes the number of successful tasks. Constraint (3) restricts each robot starting from the initial position, which is a necessity-node for each robot. Constraint (4) ensures that each survivor can be rescued no more than once. Constraint (5) illustrates that the rescue route of any robot cannot be in conflict with another. Constraint (6) limits the rescue result to 0 or 1, where 0 represents failure and 1 is success. Constraint (7) guarantees that the rescued task number is smaller than the original task number.

The mathematical model of the problem is a model with an interval function. In the next section, PSO is employed to optimize this model to generate an optimal or suboptimal rescue assignment to satisfy (2) ~ (8). Significantly, there may exist more than one rescue routes with the same value of the objective function.

The proposed algorithm

The proposed algorithm, TAPSO, is presented in Algorithm 1, where the maximal iteration number is regarded as the stopping criterion. The proposed algorithm consists of three stages. In the first stage, the particles are generated randomly, and a decoding method is presented to estimate the particles, referring to lines 1–4. In the second stage, l_{best} and g_{best} are updated to balance the exploitation and exploration, as listed in lines 5–9. Finally, the particles are evolved by updating the formulae of velocity and position in lines 10–13. The algorithm will be elaborated in the following sections.

Furthermore, several improvements are made, primarily with the particle decode method and g_{best} update strategy. The highlighted lines are the work will be performed in this paper.

The main difference between the canonical PSO and TAPSO is the interval fitness, where the particles' fitness values are intervals; thus, comparing with exact values, it is more difficult to distinguish which one is better. Moreover, it has an impact on the result of task allocation, so the comparison method between two intervals is essential and difficult. Once the superior particle is chosen, PSO can proceed based on the steps in Algorithm 1.

Algorithm 1 The framework of the proposed algorithm

```
Input: the parameters used in the proposed algorithm
Output: the optimal solution gbest
1: Initialize the particles, and determine lbest and gbest
2: While the stopping criterion is not met, do
3:     For each particle
4:         Decode and calculate its fitness
5:         Compare the current particle with lbest
6:         If the fitness value is better than lbest in history
7:             Update the lbest as the current particle
8:         End if
9:     End for
10:    Update the gbest
11:    For each particle
12:        Update particle's velocity and position
13:    End for
14: End while
15: Output gbest
```

Canonical PSO

PSO algorithm is inspired by a flock of birds seeking food. It treats each solution of the optimization problem as a bird that flies at a certain velocity in the search space, and its velocity is adjusted dynamically [38–40]. The bird is abstracted as a particle without weight and volume, and the location of the i-th particle in all the n dimensions is represented as $X_i = (x_{i1}, x_{i2}, \cdots, x_{in})$. Its velocity is represented as $V_i = (v_{i1}, v_{i2}, \cdots, v_{in})$, which is the distance to be traveled by the particle from its current position. Each particle owns a fitness value determined by the objective function that needs to be optimized, and a record of one particle's best location so far is stored, l_{best}, denoted as $P_i = (p_{i1}, p_{i2}, \cdots, p_{in})$. All particles also know their global best location, g_{best}, denoted as $P_g = (p_{g1}, p_{g2}, \cdots, p_{gn})$. The particles determine their further movements based on the experiences of their companions and themselves. Taking the canonical PSO (CPSO) algorithm [40] with inertia weight as an example, the updated formulae of a particle's velocity and location are as follows:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1 (p_{ij}(t) - x_{ij}(t)) + c_2 r_2 (p_{gj}(t) - x_{ij}(t)) \tag{8}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{9}$$

where w is the inertia weight, a user-specified parameter. A large inertia weight influences the particles toward global exploration by searching new areas, whereas a small inertia weight influences the particles toward detailed exploitation in the current search areas. c_1 and c_2 are positive constants called acceleration coefficients. Suitable inertia weight and acceleration coefficients can provide a balance between exploration and exploitation. r_1 and r_2 are random numbers within the range of [0,1].

Formula (8) is used to update the particle's velocity based on its previous velocity and the distances between its current position and l_{best} along with g_{best}. The particle then flies toward a new position based on Formula (9).

To be applicable to the problem to be solved, a variant PSO, TAPSO, is employed to gain the optimal assignment of rescue task allocation. The details of TAPSO will be elaborated in the following subsections.

Decode method

In TAPSO, one particle represents one solution, which can be decoded as the assigned tasks of all robots, denoted as matrix RS.

As set forth, the rescue route is combined by a series of integers; hence, the particles are coded in integers, and the dimension of the particles is set to N.

Caution should be exercised in this decode process to avoid an infeasible solution. Accordingly, a reference rescue sequence S_R is set in this paper, denoted as $S_R = (s_{R1}, s_{R2}, s_{R3}, \cdots, s_{Rj}, \cdots, s_{RN})$,$S_R$ is a sequence of all N tasks without overlap. Assume a particle can be represented as $X_i = (x_{i1}, x_{i2}, \cdots, x_{ij}, \cdots, x_{iN})$, $1 \le i \le N_p$, where N_p is the number of the particles in the swarm, and the dimension of each robot's rescue sequence is set to $\lceil N/M \rceil$ so as to assign the tasks averagely to each robot. After decode, we can gain the rescue sequence of the i-th particle,RS_i, denoted as follows:

$$\text{RS}_i = \begin{bmatrix} \text{rs}_1^i \\ \text{rs}_2^i \\ \cdots \\ \text{rs}_M^i \end{bmatrix} = \begin{bmatrix} \text{rs}_{11}^i, & \text{rs}_{12}^i, \cdots, & \text{rs}_{1j}^i, \cdots, & \text{rs}_{1\lceil N/M \rceil}^i \\ \text{rs}_{21}^i, & \text{rs}_{22}^i, \cdots, & \text{rs}_{2j}^i, \cdots, & \text{rs}_{2\lceil N/M \rceil}^i \\ & & & \cdots \\ \text{rs}_{M1}^i, & \text{rs}_{M2}^i, \cdots, & \text{rs}_{Mj}^i, \cdots, & \text{rs}_{M\lceil N/M \rceil}^i \end{bmatrix}$$

Taking Fig. 1 for example, there are 3 robots and 13 tasks, so, N=13, M=3,$\lceil N/M \rceil$=5. Therefore, assume the i-th particle can be represented as $X_i = (x_{i1}, x_{i2}, \cdots, x_{ij}, \cdots, x_{iN})$ $= (5, 13, 22, 16, 77, 9, 22, 45, 68, 4, 3, 12, 83)$, correspondingly, assume its rescue sequence is:

$$\text{RS}_i = \begin{bmatrix} rs_1^i \\ rs_2^i \\ rs_3^i \end{bmatrix} = \begin{bmatrix} 2, & 6, & 10, & 0, & 0 \\ 7, & 4, & 12, & 3, & 9 \\ 1, & 11, & 5, & 8, & 13 \end{bmatrix}.$$

Then two questions arise: How to decode a particle into a solution of different size? and How to guarantee the rescue sequence contains all tasks and have no overlap tasks among all robots?

Once overlap occurs, which means one task is rescued more than once, and it will create an unfeasible solution. Accordingly, the following decode method is designed to avoid this situation and satisfy the above requirements, which is as illustrated in Algorithm 2.

First, one dimension in particle *i*, denoted as x_{ij}, is selected to perform the mod and add operations; therefore, the obtained result should be an integer number between 1 and *N*. Next, the task corresponding to the gained integer number is selected and deleted from the reference rescue sequence S_R, listed in lines 2–4. Thus, we can obtain a sequence of integers by repeating the previous operations *N* times.

The number of elements in S_R is *N,* they are all *N* tasks without overlap, the dimension of the particle is *N* as well. When performing the above decode steps one time, one task can be selected and then deleted from the S_R, the delete operation guarantee that the decode sequence has no overlap. When performing the above operations *N* time, all survivors can be selected and deleted from the S_R, as a result, the number of the tasks in S_R decreases one compared with the previous operation. Therefore, when repeat *N* times, it can guarantee that all *N* tasks can be selected without overlap, there is hence the decode method can avoid the occurrence of unfeasible solutions, which can meet the requirement in this paper.

From the above operations, a sequence of *N* integers is gained, further, tasks need to assign to each robot, to assign averagely, the maximum number of tasks assigned to each robot is set to $\lceil N/M \rceil$. Firstly, the distances between the first task and all *M* robots are calculated, the task is then assigned to the robot with the shortest distance. Next, for the second task, we calculate the distance between it and the remaining (*M*-1) robots except for the selected one, the robot with the shortest distance is assigned to the task. Repeating the previous procedure *M* times, we can obtain a rescue sequence. For the remaining tasks, the same operations and a new round are performed, and then we can obtain the rescue routes, as listed in lines 8–13. Figure 2 depicts the assignment procedure for each robot. Overall, the rescue sequence for each robot is gained.

Algorithm 2 Decode method

```
Input: particles i; task number, N; robot number, M, S_R; all tasks' position TP, all robots' position RP
Output: the rescue sequence of all robots
1: For particle i, $X_i$
2:     For dimension j in $X_i$
3:         $sx_{ij}=S_R(\text{mod}(x_{ij}, N-j+1)+1)$
4:         remove $sx_{ij}$ from $S_R$
5:         judge whether $S_R=\varnothing$
6:         If $S_R=\varnothing$, then $S_R$ is set to its original value
7:     End
8:     calculate the distance between $sx_{ij}$ and the remaining robots
9:     find the robot with the shortest distance
10:    assign $sx_{ij}$ to the list of the above robot
11:    remove the robot out of robot set
12:    judge whether the robot set is empty
13:        if empty, then the robot set update to the original set
14:    End
15: While decode all N dimension in $X_i$
16: Output rescue sequence of all robots
```

Position and velocity update method

The particle is decoded as a series of integers, a ceil function is performed on the original position update formula to guarantee the updated position is an integer. Therefore, the particle's position and velocity update formulae in this paper are listed as follows:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1r_1(p_{ij}(t) - x_{ij}(t)) + c_2r_2(p_{gj}(t) - x_{ij}(t)) \tag{11}$$

$$x_{ij}(t+1) = \lceil x_{ij}(t) + v_{ij}(t+1) \rceil \tag{12}$$

where ⌈ ⌉ indicates the ceil function.

Comparison between particles

In this paper, l_{best} and g_{best} are updated by comparing the fitness values between them and other particles. Because all

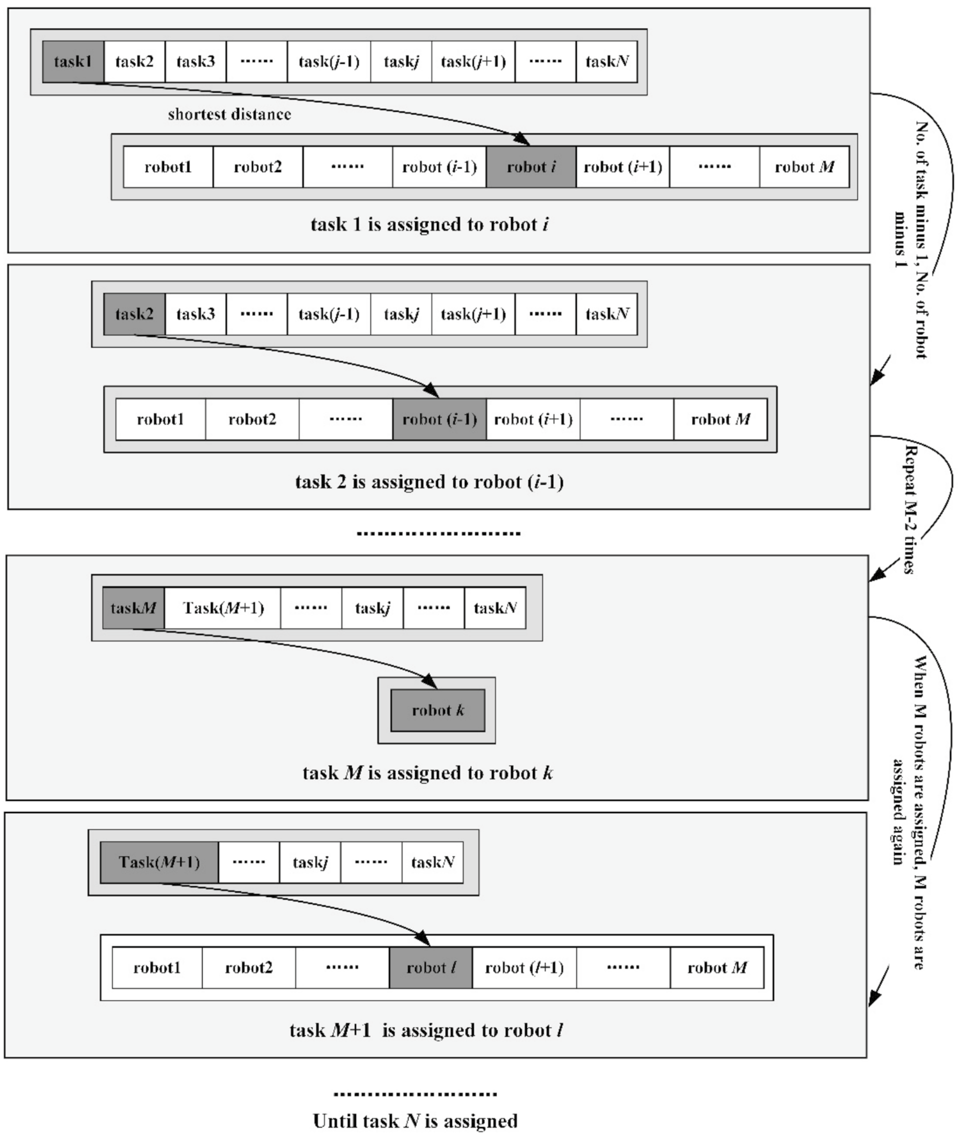
task1
task2
task3
······
task(j-1)
taskj
task(j+1)
······
taskN
shortest distance
robot1
robot2
robot (i-1)
robot i
robot (i+1)
robot M
task 1 is assigned to robot i
No. of task minus 1, No. of robot minus 1
task 2 is assigned to robot (i-1)
Repeat M-2 times
taskM
Task(M+1)
robot k
task M is assigned to robot k
When M robots are assigned, M robots are assigned again
robot l
robot (l+1)
task M+1 is assigned to robot l
Until task N is assigned

Fig. 2 The assignment procedure for each robot

fitness values are in the form of intervals, how to select the optimal one is essential.

Consequently, the comparison between the two intervals is given in this subsection. To select the optimal one, several criteria are determined; two common-used criteria definitions—the midpoint and distance of an interval—are given. For an interval $[a^{-}, a^{+}]$, its midpoint and distance are $(a^{-} + a^{+})/2$ and $|a^{+} - a^{-}|$, respectively.

The principle of intervals comparison is that the larger the midpoint of an interval and the smaller the interval distance, the better the interval is. In this TAPSO algorithm, the particle with a larger midpoint and smaller distance will be elected as the candidates of l_{best} or g_{best}. The midpoint relates to the interval's two bounds: the larger the better—it represents more successful tasks. The interval distance represents the difference between the upper and lower bounds, where a smaller value represents a smaller difference, and the rescued number is significantly more precise.

Assume two particles, X_i and X_j, whose fitness is $[F_i^{-}, F_i^{+}]$ and $[F_j^{-}, F_j^{+}]$, respectively. The following formula is used to find the better one:

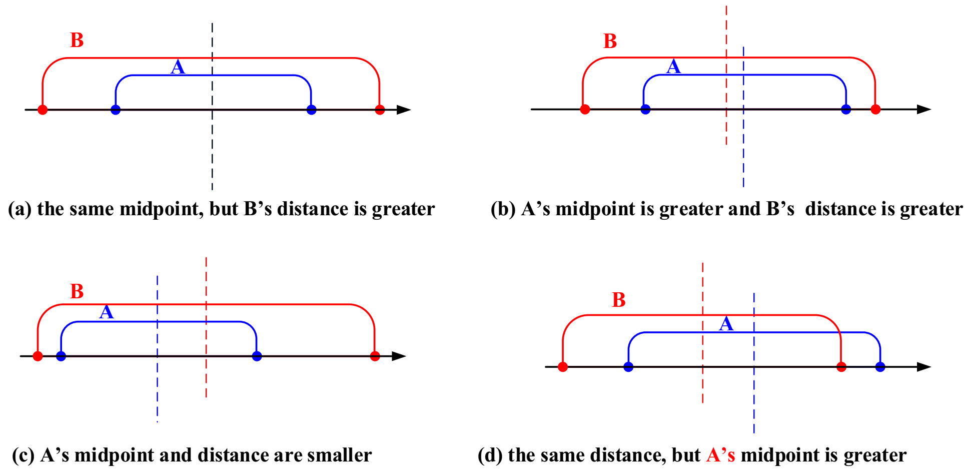
B
A
(a) the same midpoint, but B's distance is greater
B
A
(b) A's midpoint is greater and B's distance is greater
B
A
(c) A's midpoint and distance are smaller
B
A
(d) the same distance, but A's midpoint is greater

Fig. 3 Four cases

$$f_{\max}(X_i, X_j) = \begin{cases} X_i, \left(\dfrac{F_i^- + F_i^+}{2} > \dfrac{F_j^- + F_j^+}{2}\right) \text{ or} \\ \left(\dfrac{F_i^- + F_i^+}{2} = \dfrac{F_j^- + F_j^+}{2} \text{ and } F_i^+ - F_i^- < F_j^+ - F_j^-\right) \\ X_j, \text{otherwirse} \end{cases} \quad (13)$$

Based on Formula (13), one can well perceive that A is better than B in Fig. 3a; in Fig. 3b, A is better; in Fig. 3c, B is better than A; in Fig. 3d, A is better than B.

l_{best} and g_{best} update strategies

During the search, g_{best} spreads information to particles, guides them to fly, and guarantees the particles find the optimal solution. g_{best} plays a significant role in the search process; consequently, the selection of g_{best} is particularly important, as discussed in this section.

Since g_{best} is selected from l_{best}, before update g_{best}, l_{best} should be determined in advance. If the current particle's fitness is larger than or equal to l_{best}, then l_{best} updates to the current particle, and saves it in a set called the l_{best} set, which is used to update g_{best} later. Otherwise, l_{best} remains unchanged.

g_{best} is selected from the l_{best} set, but due to the particular nature of the problem, there may exist two situations while updating g_{best}: (1) only one l_{best} l_{best} in the set; (2) more than one l_{best} in the set. For these different cases, g_{best} update method is different and given as follows:

1. **Only one $\mathbf{l_{\text{best}}}$ in the set**

   If there is only one l_{best} in the set, compare it with the current g_{best}. If its fitness value is larger than or equal to g_{best} g_{best}, then, g_{best} updates to l_{best}; otherwise g_{best} stays the same.

2. **More than one $\mathbf{l_{\text{best}}}$ in the set**

There may exist three different cases:

Case 1: all l_{best} s in the set have different fitness values but are smaller than g_{best}.

In this case, g_{best} stays the same.

Case 2: all l_{best} s in the set have different fitness values.

In this case, g_{best} updates to the l_{best} with the largest fitness value.

Case 3: more than one l_{best} l_{best} have the same fitness, and their fitness values are larger than or equal to g_{best}.

In this case, l_{best} s may have different rescue sequences, so how to select one from many l_{best} s with the same fitness value is a problem that needs to be solved. To guarantee the swarm's diversity and convergence, g_{best} updates dynamically.

At the initial stage of the search process, l_{best} which is equal to or larger than g_{best}, and having the greatest difference with g_{best}, is selected as the new g_{best} to guarantee the diversity of the swarm. In the later phase, the particle that is equal to or larger than g_{best}, but having the least difference with g_{best}, is selected as the new g_{best} to guarantee the swarm's convergence. This ensures the swarm's diversity and guarantees the algorithm's convergence. This update method balances exploitation and exploration more effectively.

The difference between the two particles is defined and estimated by the method presented in Algorithm 3.

Algorithm 3 Global best solution update method

```
Input: the current particles; gbest; lbest set, lbest
Output: gbest
1: Num=get the number of elements in lbest set
2: If (Num>=1)&(with different fitness)
3:      Compare gbest and lbest
4:      If gbest<=lbest
5:          gbest=lbest
6:      End
7: End
8: If (Num>1) & (with the same fitness)
9:      Decode gbest and the current particles
10:     Calculate difference offs between the current particle and gbest
12:     If the algorithm phase is prophrase
13:         gbest=lbest(index(max(offs)))
14:     Elseif the algorithm phase is anaphase
15:         gbest=index(min(offs))
16:     Else
17:         gbest=lbest(index(max(offs)))
18:     End
19: End
20: Output gbest
```

Assume the fitness value of l_{best} is equal to or larger than g_{best}, in which case g_{best} should update. g_{best} is denoted as $g_{\text{best}} = (p_{g1}, p_{g2}, \cdots, p_{gl}, \cdots, p_{gN})$. First, select a l_{best} from the l_{best} set, denoted as $lbest_k = (p_{k1}, p_{k2}, \cdots, p_{kj}, \cdots, p_{iN})$. After decoding, the rescue route of the current g_{best}, $S_{g\text{best}}$, and one l_{best}, RS_{lbest_k}, are

$$S_{g\text{best}} = \begin{bmatrix} \text{rs}_{11}^{g}, \text{rs}_{12}^{g}, \cdots, \text{rs}_{13}^{g}, \cdots, \text{rs}_{1\lceil N/M\rceil}^{g} \\ \text{rs}_{21}^{g}, \text{rs}_{22}^{g}, \cdots, \text{rs}_{23}^{g}, \cdots, \text{rs}_{2\lceil N/M\rceil}^{g} \\ \cdots \\ \cdots \quad \text{rs}_{ij}^{g} \quad \cdots \\ \cdots \\ \text{rs}_{M1}^{g}, \text{rs}_{M2}^{g}, \cdots, \text{rs}_{M3}^{g}, \cdots, \text{rs}_{M\lceil N/M\rceil}^{g} \end{bmatrix}$$

and $$\text{RS}_{lbest_k} = \begin{bmatrix} \text{rs}_{11}^{lk}, \text{rs}_{12}^{lk}, \cdots, \text{rs}_{1j}^{lk}, \cdots, \text{rs}_{1\lceil N/M\rceil}^{lk} \\ \text{rs}_{21}^{lk}, \text{rs}_{22}^{lk}, \cdots, \text{rs}_{2j}^{lk}, \cdots, \text{rs}_{2\lceil N/M\rceil}^{lk} \\ \cdots \\ \cdots \quad \text{rs}_{pq}^{lk} \quad \cdots \\ \cdots \\ \text{rs}_{M1}^{lk}, \text{rs}_{M2}^{lk}, \cdots, \text{rs}_{Mj}^{lk}, \cdots, \text{rs}_{M\lceil N/M\rceil}^{lk} \end{bmatrix}$$

,respectively. Based on these, the difference between $S_{g\text{best}}$ and RS_{lbest_k} can be calculated. The particle having the greatest or least difference with g_{best} is selected as the new g_{best} based on the algorithm's running phase, listed in lines 8–13.

The difference calculating method is described as follows: the indexes of the same tasks in both $lbest_k$ and g_{best} are found, and the deviation between the two indexes is calculated. Assuming $\text{rs}_{pq}^{lk} = \text{rs}_{ij}^{g}$, their index difference (one task's deviation value), *off*, is presented as follows:

$$\text{index}(\text{rs}_{ij}^{g}) - \text{index}(\text{rs}_{pq}^{lk}) \tag{14}$$

where $\text{index}(\text{rs}_{ij}^{g})$ is the index of rs_{ij}^{g} in $S_{g\text{best}}$, and $\text{index}(\text{rs}_{pq}^{lk})$ is the index of rs_{pq}^{lk} in RS_{lbest_k}.

Consequently, based on formula (14), the difference between $lbest_k$ and g_{best} is calculated by formula (15):

$$\text{offs} = \sum^{N} |\text{index}(\text{rs}_{ij}^{g}) - \text{index}(\text{rs}_{pq}^{lk})| \tag{15}$$

Hence, the particle with the maximum or minimum value of offs is selected as g_{best} based on the algorithm's running phase, listed in lines 14–15.

The index definition is set as formula (16) shown, from which, any two particles' difference can be calculated. The difference between formula (16) and normal matrix lays on the additional labels below each element, which are used to mark the sequence number for each element, where a_{ij} denotes the jth task of robot i, $(i-1)\lceil N/M\rceil + j$ is the sequence number of a_{ij}, *i.e.*, $\text{index}(a_{ij})$. Therefore, the sequence number in formula (16) is designed to calculate offs in formula (15).

$$\begin{bmatrix} \underbrace{a_{11}}_{1}, & \underbrace{a_{12}}_{2}, & \underbrace{a_{13}}_{3}, & \cdots, & \underbrace{a_{1(\lceil N/M\rceil-1)}}_{\lceil N/M\rceil-1}, & \underbrace{a_{1\lceil N/M\rceil}}_{\lceil N/M\rceil} \\ \underbrace{a_{21}}_{\lceil N/M\rceil+1}, & \underbrace{a_{22}}_{\lceil N/M\rceil+2}, & \underbrace{a_{23}}_{\lceil N/M\rceil+3}, & \cdots, & \underbrace{a_{2(\lceil N/M\rceil-1)}}_{2\lceil N/M\rceil-1}, & \underbrace{a_{2\lceil N/M\rceil}}_{2\lceil N/M\rceil} \\ \cdots & & \underbrace{a_{ij}}_{(i-1)\lceil N/M\rceil+j} & & \cdots & \\ \underbrace{a_{M1}}_{(M-1)\lceil N/M\rceil+1}, & \underbrace{a_{M2}}_{(M-1)\lceil N/M\rceil+2}, & \underbrace{a_{M3}}_{(M-1)\lceil N/M\rceil+3}, & \cdots, & \underbrace{a_{M(\lceil N/M\rceil-1)}}_{M\lceil N/M\rceil-1}, & \underbrace{a_{M\lceil N/M\rceil}}_{M\lceil N/M\rceil} \end{bmatrix} \tag{16}$$

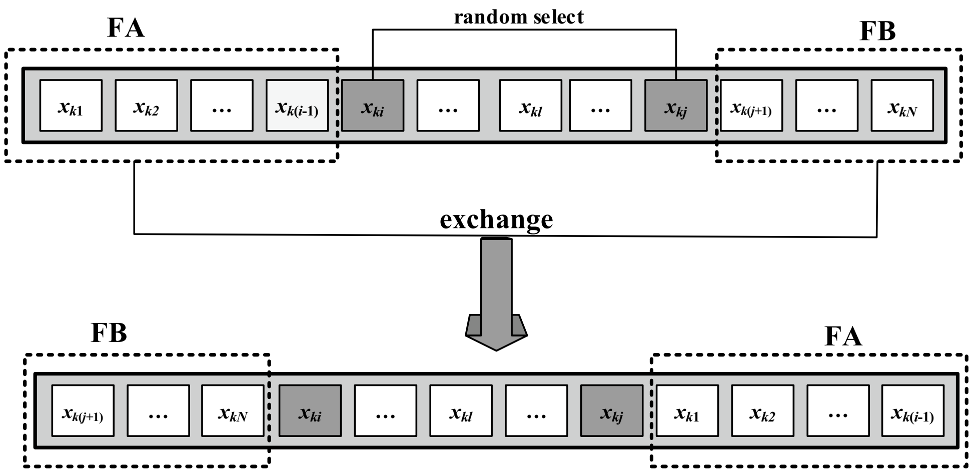

Fig. 4 Exchange operation

To better illustrate the process of calculating *offs*, an example is given as following: suppose $l\text{best}_k$ and g_{best} are decodes as $S_{g\text{best}} = \begin{bmatrix} 1,2,3 \\ 4,5,0 \end{bmatrix}$ and $RS_{l\text{best}_k} = \begin{bmatrix} 5,2,0 \\ 4,1,3 \end{bmatrix}$, offs can be gained by formula (16), hence, $S_{g\text{best}} = \begin{bmatrix} \underbrace{1}_{1}, \underbrace{2}_{2}, \underbrace{3}_{3} \\ \underbrace{4}_{4}, \underbrace{5}_{5}, \underbrace{0}_{6} \end{bmatrix}$ and $RS_{l\text{best}_k} = \begin{bmatrix} \underbrace{5}_{1}, \underbrace{2}_{2}, \underbrace{0}_{3} \\ \underbrace{4}_{4}, \underbrace{1}_{5}, \underbrace{3}_{6} \end{bmatrix}$. Firstly, start from task 1, find its indexes in $S_{g\text{best}}$ and $RS_{l\text{best}_k}$, they are 1 and 5, respectively, and its off is $|5-1| = 4$; then, task 2, the indexes are 2 and 2, respectively and its off is 0; task 3′s off is $|3-6| = 3$; after that, task 4′s off is $|4-4| = 0$; finally, it's task 5, and its off is $|5-1| = 4$; as a result, offs $= 4+0+3+0+4 = 11$.

There are three phases when running the algorithm: prophase, metaphase, and anaphase. The greatest and least differences at the prophase and anaphase can be selected to complete the g_{best} update, respectively. For the metaphase of the algorithm, the g_{best} update method is the same as the traditional update method, listed in lines 16–17.

Local search method

In this paper, local search methods are adopted to speed up the convergence to improve the solution's superiority. Several classic local search methods could apply to the problem of task allocation in [41–43]. To find the best solutions and guarantee the algorithm against being trapped in the local optimum, the following local search method is used to improve this approach.

First, randomly select two elements in g_{best}, p_{gl} and p_{gk}, and add a random number to each element;p_{gl} becomes $p_{gl} \cdot (1+\text{rand})$ and p_{gk} becomes $p_{gk} \cdot (1+\text{rand})$. Second, calculate the fitness of the newly generated particles, and once the fitness is larger than g_{best}, g_{best} updates to the new one; otherwise, exchange some portions of the two particles depicted in Fig. 4. Then, calculate the fitness of the new particle; if its fitness is better, g_{best} updates to the new one; otherwise, g_{best} remains the same. Repeat the above operations until reaching the stopping iterations.

Simulations

Parameter setting

To verify the proposed method, several scenarios are presented in this section, and different methods were used to compare with the proposed method. All the methods were run in Matlab R2013a, and parameters were set the same as [44]: population size of 100, evolutionary generation of 400, and $\gamma = 0.037$. In all experiments, the world x and y coordinates ranged from -5000 m to 5000 m; the initial survival time was 2000s; the threshold was 100 s; the velocity of the robot was 3.0 m/s.

In this section, 10 different scenarios were generated in different 2-Dimensional planes for 5 trails: 50 cases were tackled to determine the reference sequence. Furthermore, 18 difference cases were generated in different trails to determine the three phases. The contributions of the g_{best} update method were illustrated by 63 different cases.

Three comparison methods were used to verify the method in this paper. One method is the canonical PSO algorithm (CPSO); other strategies are the same with TAPSO. The second method is the Genetic Algorithm (GA), using the same strategies as TAPSO; the crossover and mutation probability were 0.80 and 0.10. The last comparison method

Table 2 Results and running time for different reference sequences

Case	*M*	*N*	Randomly		Order		Survival time		Distance	
			Results	CPU time	Results	CPU time	Results	CPU time	Results	CPU time
1	10	20	19	9.53	19	9.15	19	**9.00**	19	9.16
	5	30	**25**	**10.70**	[24 25]	11.65	24	11.34	[22 24]	11.33
	5	40	[23 24]	**14.17**	**[25 26]**	14.69	24	14.98	[22 23]	15.27
	5	50	[24 25]	**17.51**	**25**	18.19	[24 26]	18.41	23	18.27
	13	39	35	**17.48**	35	18.19	[34 35]	18.19	35	18.28
	14	56	[48 53]	24.51	[47 51]	22.32	[48 50]	**22.59**	**52**	22.72
	17	51	**50**	**20.31**	[46 49]	20.97	48	21.33	[47 48]	21.5
	6	60	**28**	**24.00**	[27 29]	25.07	[27 29]	25.07	[27 28]	25.25
	10	60	[42 43]	**24.85**	[43 45]	25.87	[40 45]	25.97	**[43 44]**	26.15
	3	60	[14 15]	**24.62**	[14 15]	25.87	**[15 16]**	25.58	15	25.55
No. of best			**3**	**8**	**2**	**0**	**1**	**1**	**2**	**0**
2	10	20	20	18.08	20	17.84	20	17.83	20	**17.79**
	5	30	**[24 25]**	**22.34**	[22 23]	23.25	24	23.47	[22 23]	23.53
	5	40	**[25 26]**	**29.22**	**[25 26]**	30.63	[24 25]	30.82	[22 25]	30.79
	5	50	24	**37.08**	[22 24]	38.66	[22 24]	38.87	**[24 25]**	38.89
	13	39	**35**	**33.37**	[33 35]	34.76	34	34.83	[34 35]	34.74
	14	56	[44 46]	**48.56**	[44 46]	50.26	[43 46]	50.49	**[45 46]**	50.21
	17	51	[45 46]	**44.74**	[44 45]	45.97	45	46.58	**46**	46.46
	6	60	**26**	**44.36**	[24 27]	46.31	[25 26]	46.76	25	46.75
	10	60	**[42 43]**	**47.85**	[38 43]	49.97	[39 44]	50.05	40	50
	3	60	**[15 16]**	**46.12**	[14 16]	47.88	[14 15]	48.62	14	48.26
No. of best			**6**	**9**	**1**	**0**	**0**	**0**	**3**	**1**
3	10	20	20	9.11	20	**8.97**	20	8.92	20	9.16
	5	30	**[23 24]**	**11.35**	[22 23]	11.71	23	11.83	23	11.98
	5	40	[23 27]	**14.85**	**[24 26]**	15.42	[24 25]	15.47	23	15.57
	5	50	[23 24]	**18.72**	**24**	19.56	22	19.61	**24**	19.73
	13	39	**35**	26.80	34	**17.49**	[33 34]	17.52	34	17.79
	14	56	**[45 46]**	**24.52**	[42 46]	25.23	[43 45]	25.37	**[45 46]**	25.6
	17	51	45	**22.45**	44	23.28	45	23.54	46	23.72
	6	60	[24 27]	**22.52**	**[25 29]**	23.28	[26 28]	23.52	[25 27]	23.65
	10	60	**[40 42]**	**24.21**	40	25.04	[39 44]	25.29	[38 41]	25.33
	3	60	**[14 15]**	**23.17**	14	24.08	[13 14]	24.5	[13 14]	24.3
No. of best			**5**	**8**	**3**	**2**	**0**	**0**	**3**	**0**
4	10	20	[20 20]	9.29	[20 20]	20.52	[20 20]	9.93	[20 20]	**8.56**
	5	30	**[22 26]**	**11.50**	[22 24]	11.68	[22 23]	11.92	[21 22]	11.53
	5	40	**[23 26]**	**15.36**	[23 23]	16.14	[23 23]	15.47	[22 25]	15.49
	5	50	[23 25]	**19.02**	**[24 24]**	20.65	[23 24]	19.76	**[24 24]**	19.8
	13	39	**[37 38]**	**16.43**	[37 37]	17.96	[36 36]	16.56	[37 37]	16.73
	14	56	**[48 50]**	**23.27**	[47 49]	25.44	[46 48]	24.03	[46 48]	25.09
	17	51	**[47 48]**	**22.16**	[46 47]	22.36	[45 47]	22.97	[45 48]	22.98
	6	60	**[29 31]**	**23.29**	**[29 31]**	25.13	[27 31]	24.00	[28 31]	23.69
	10	60	[44 47]	**25.16**	**[44 48]**	26.73	[43 47]	25.34	[43 44]	24.29
	3	60	**[18 18]**	**22.46**	[16 17]	24.94	[17 17]	23.15	[17 19]	23.32
No. of best			**7**	**9**	**3**	**0**	**0**	**0**	**1**	**1**

Table 2 (continued)

Case	*M*	*N*	Randomly		Order		Survival time		Distance	
			Results	CPU time	Results	CPU time	Results	CPU time	Results	CPU time
5	10	20	[20 20]	42.45	[20 20]	39.98	[20 20]	40.1	[20 20]	39.97
	5	30	[23 23]	**51.79**	**[24 24]**	54.09	[22 23]	54.07	[22 24]	54.08
	5	40	[23 24]	**67.86**	[23 24]	71.09	**[24 24]**	70.97	[23 23]	71.87
	5	50	**[24 25]**	**88.20**	[23 25]	89.5	**[24 25]**	89.65	[24 24]	89.67
	13	39	**[37 37]**	**72.20**	**[37 37]**	74.14	[36 36]	74.53	**[37 37]**	74.82
	14	56	**[48 50]**	**105.07**	[48 49]	108.47	[46 48]	109	[46 46]	108.5
	17	51	**[47 48]**	**101.14**	[46 47]	102.52	[45 47]	102.27	[45 48]	101.91
	6	60	**[28 33]**	**105.42**	[29 31]	106.94	[28 29]	107.76	[30 30]	107.37
	10	60	**[45 46]**	**109.38**	[44 46]	111.66	**[45 46]**	111.55	**[45 46]**	111.03
	3	60	[17 19]	**104.69**	**[18 19]**	106.78	[16 17]	106.69	[17 18]	107.35
No. of best			**6**	**9**	**3**	**0**	**3**	**0**	**2**	**0**
Total % of best			**57%**	**86%**	**24%**	**4%**	**8%**	**2%**	**22%**	**4%**

Table 3 Results and running time for different periods

Case	Prophase (%)	Metaphase (%)	Anaphase (%)	No. of best results	Total no. rescued
1	10	80	10	2	[257 264]
2	20	70	10	3	[257 267]
3	30	60	10	5	[259 271]
4	40	50	10	3	[255 270]
5	35	55	10	4	[258 271]
6	35	45	20	4	[257 273]
7	30	50	20	5	[259 271]
8	20	60	20	3	[257 267]
9	**30**	**40**	**30**	**7**	**[260 272]**
10	40	30	30	4	[256 270]
11	50	20	30	5	[259 270]
12	50	40	10	4	[257 268]
13	40	20	40	3	[254 265]
14	40	40	20	3	[254 268]
15	50	30	20	4	[258 268]
16	50	20	30	4	[255 262]
17	50	10	40	4	[257 267]
18	0	100	0	2	[257 265]

is the consensus-based bundle auction (CBBA), which is based on a market auction strategy [45].

All of the following simulation results are based on 20 independent runs, with the best results highlighted in bold.

Selection of results under different scenarios

The reference rescue sequence,S_R, influences the decode result; thus, generating S_R is essential to exploring a good solution.

In this paper, four different methods of generating S_R are presented: random generation method, generated by the order from small to large, generated by survival time from small to large, and generated by distance from near

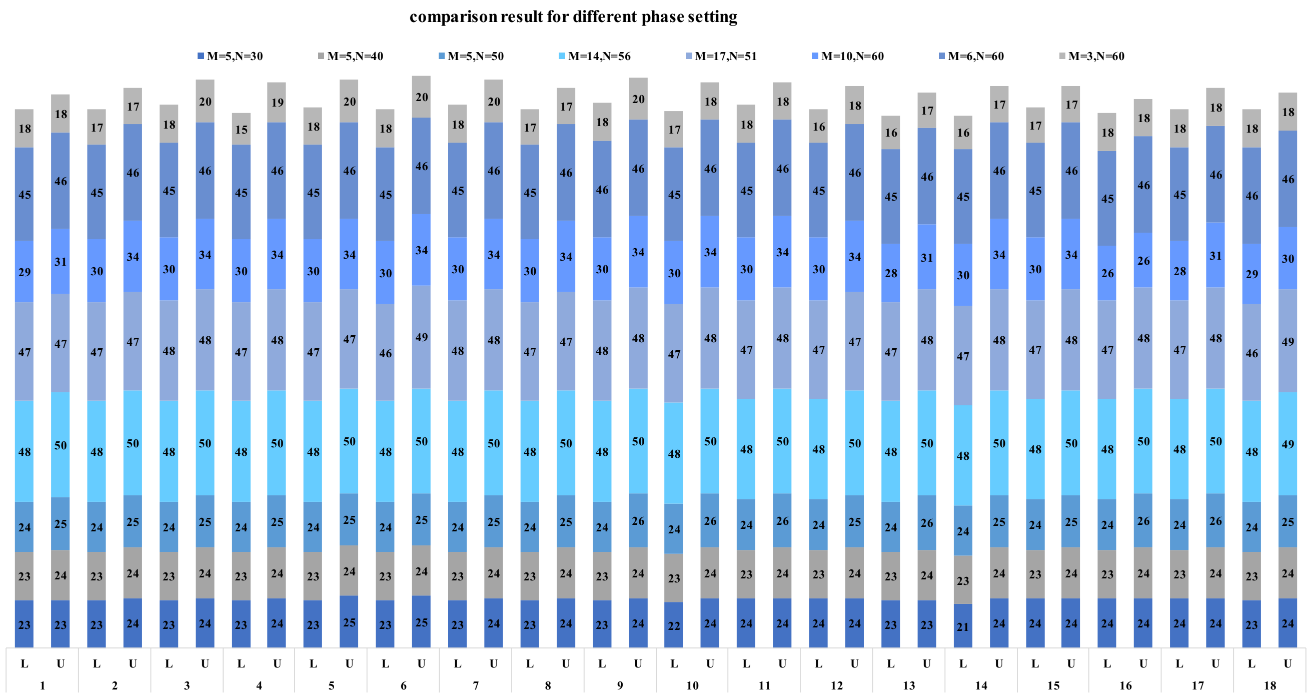

Fig. 5 Comparison results for different phase settings

Table 4 Results and running time for different g_{best} update methods

Case		Proposed gbest update method		Common gbest update method	
n	m	Result	CPU time	result	CPU time
4	60	**[182 192]**	**45.74 s**	[179 189]	45.84 s
4	56	**[186 194]**	47.38 s	[179 187]	**47.32 s**
2	60	**[109 115]**	48.32 s	[107 113]	**48.12 s**
30	60	**[498 505]**	**62.22 s**	[498 504]	61.82 s
20	100	**[608 645]**	94.87 s	[604 647]	**94.28 s**
5	100	**[217 234]**	**82.00 s**	[219 228]	83.64 s
4	20	**[155 159]**	**15.80 s**	[153 158]	16.90 s

to far. Assume there are N tasks, for the first method, S_R is an integer array generated randomly within [1, N], and each integer can only appear once; for the second method, $S_R = (1, 2, 3, \cdots, N)$, and the other two methods are generated based on the N tasks' survival time and distance from each task to the robots (in this paper, initial points are same for all robots).

Based on four different generation methods, we executed our TAPSO algorithm respectively, and the results are listed in Table 2.

Table 2 reports that, in different rescue environments, the random generation method performed best, and 57% of cases were better than the other three methods except for the scenarios in which all the four methods obtained the same results. Furthermore, the CPU running time was relatively shorter than the other three methods, where 86% of cases obtained results in the shortest CPU running time. Survival time is the most important factor that affects the rescue results, however, among the 50 different cases, S_R generated by survival time only beat its rivals 2 times, and its CPU running time performed best only 4 times. Although S_R generation methods based on order and distance ranked in the middle, their CPU running time performed best 2 times, and the best results account for 24% and 22% of all cases, respectively. The random generation method for S_R had the highest performance, as a result, in the following simulation, it was used to determine S_R.

Determining different running phases for TAPSO

In this paper, the search process is divided into three phases: prophase, metaphase, and anaphase. However, what were the corresponding periods?

In this paper, 18 different cases are given to select the three phases, as listed in Table 3. Moreover, eight different scenarios are presented, and the corresponding results are illustrated in Fig. 5.

In Table 3, the percentage indicates how much each phase accounted for the maximum iterations, and Fig. 5 lists the comparison results for each scenario. For case 9, its lower and upper values of the total number of successful tasks are the largest at the range of [260 272]. Also, the number of

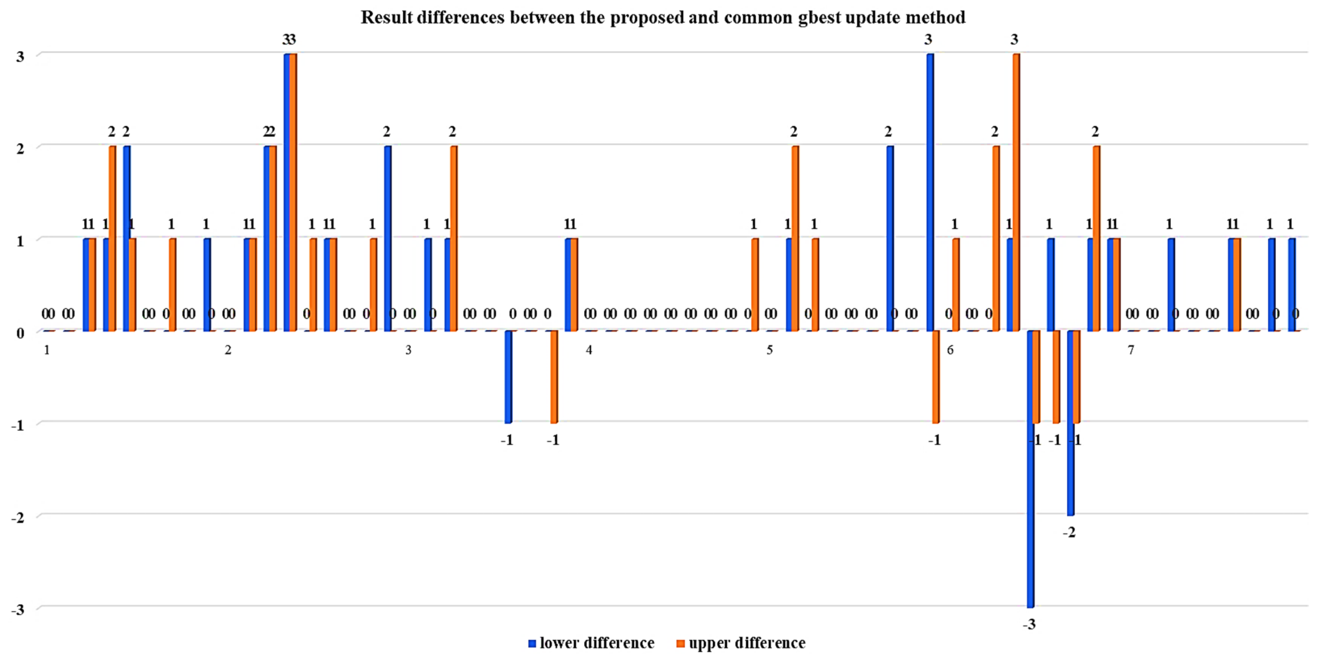

Fig. 6 Result differences between the proposed and common g_{best} update methods for seven different trials

best results for case 9 is the largest. Because we are most interested in Case 9, so, prophase, metaphase, and anaphase are set as 30%, 40%, and 30% of the maximum iterations.

The contribution of the g_{best} update method

To estimate the contribution of the g_{best} update method, we compare the proposed method with the method that uses the common g_{best} update method in this section. Seven different scenarios, each with nine different cases, were used to verify the contribution of the g_{best} update method in TAPSO. Comparison results are listed in Table 4 and Fig. 6.

Table 4 reports that TAPSO always out-performed the common g_{best} update method in terms of the total number of successful tasks in 63 different cases. When the numbers of tasks are greater, the CPU running time of TAPSO made little difference from the common update method; in most instances, the CPU running time of TAPSO is even slightly shorter than the common method, since the new g_{best} update method can increase the convergence speed. For example, for all 9 cases when 4 robots rescued 56 tasks, the results for the two methods were [186 194] and [179 187], respectively. Their difference was 7; in the rescue situation, one more rescued survivor is of great value, not to mention 7. Furthermore, their difference in CPU running time was 0.06 s, which is extremely small compared with the entire rescue process.

Figure 6 illustrates the differences in results between TAPSO and the common g_{best} update method for 63 different cases, where the numbers denote the difference between two intervals of the two methods, the blue rectangle is the difference between the lower bounds of two intervals, and the orange rectangle is the difference between the upper bounds. Further, 0 represents two intervals being the same, and the larger of the difference represents the superior of the proposed methods. From Fig. 6, 46% (29 cases) of the solutions were improved from the use of the g_{best} update method in this paper, while 6.35% of the solutions did not outperform its counterpart, and 46% of the solutions were the same as the common method, especially in some relatively simple scenarios. For example, 30 robots rescued 60 tasks, and 8 cases out of 9 have the same results as the common method. For several complex scenarios, TAPSO performs better. From Table 4 and Fig. 6, we can assert that TAPSO handles certain complex scenarios better, and CPU running time makes little difference for the common g_{best} update method.

Comparisons with other methods

For the previous scenarios, the tasks can be assigned to the robot averagely. To better verify the proposed method, more complex scenarios should be considered. Furthermore, CPSO, GA and CBBA were employed as the comparison algorithms. For comparability, the parameters are the same for all algorithms, with simulation results listed in Table 5.

Based on Table 5, even though the tasks were not assigned equally to each robot, TAPSO still performed the best; CBBA was always the fastest to obtain the results in terms of CPU running time, which is important for rescue,

Table 5 Comparison results with methods in some more complex scenarios

Task no.	Robot no.	TAPSO		CPSO	
		Results	CPU time	Results	CPU time
5	2	[4 5]	3.81s	[4 5]	3.32s
13	2	[12 13]	6.11s	[12 13]	5.84s
27	2	**[26 27]**	15.95s	[21 27]	10.53s
13	5	[12 12]	8.23s	[11 12]	6.04s
21	5	**[18 21]**	13.63s	[16 19]	8.63s
33	5	**[30 33]**	19.62s	[27 33]	13.69s
22	10	**[21 22]**	17.47s	[18 20]	10.06s
44	10	**[29 35]**	26.70s	[24 28]	16.38s
66	10	**[53 64]**	36.61s	[35 53]	22.74s
55	20	**[45 52]**	38.16s	[37 48]	27.00s
88	20	**[58 64]**	52.37s	[48 57]	34.76s
Task No.	Robot No.	GA		CBBA	
		Results	Results	Results	CPU time
5	2	[4 5]	7.36s	[4 5]	**0.18**
13	2	[12 13]	13.86s	[8 11]	**0.34**
27	2	[22 27]	25.65s	[10 17]	**0.40**
13	5	[12 12]	14.16s	[8 10]	**0.37**
21	5	[17 21]	20.77s	[14 17]	**0.60**
33	5	[28 33]	30.21s	[18 22]	**1.20**
22	10	[18 21]	24.65s	[14 20]	**0.95**
44	10	[25 32]	43.61s	[25 33]	**1.93**
66	10	[44 61]	62.22s	[30 39]	**2.74**
55	20	[39 50]	58.44s	[40 44]	**5.58**
88	20	[51 59]	87.69s	[47 56]	**9.57**

and GA required the longest time to solve the problem, but without obtaining any best result.

We can conclude that all methods except CBBA are relative computationally expensive. Despite this, TAPSO has the highest performance and can be used to solve the task allocation problem effectively. It is slightly time-consuming; however, it is still acceptable.

Conclusion

This study introduced TAPSO, an upgraded version of the well-known algorithm PSO, to address the robot rescue task allocation in uncertain time constraints problem. The principle ideas of the algorithm are establishing an objective function of the maximal number of successful tasks based on the change of the survival time; and building a mathematical model of task allocation. On the other hand, TAPSO improves the decode method and the global best solution update strategy in the original PSO thus increases the effectiveness of the robot's performance in complex scenes.

Several experiments have been conducted in various scenarios where TAPSO was benchmarked against three methods CPSO, GA and CBBA. Among the algorithms, TAPSO was outstanding in identifying optimal solutions. For some non-optimal solutions, CPSO and CBBA show shorter CPU running time than TAPSO, but the differences are minor and do not really affect the overall rescue strategies.

Although TAPSO demonstrated effectiveness and efficiency in handling task allocation with uncertainty time condition, the uncertainty of the robot's velocity was not considered. In this paper, the velocity was declared as a constant, but it could be a variable. In many real-world scenarios, the velocity of moving robots might vary due to the complexity of the terrain and the accuracy of the detection of survivors' locations. Like evaluating the allocated time, evaluating a robot's velocity in different conditions remains a non-trivial work.

In the future, we attempt to address that uncertain velocity issue by integrating another function for the velocity changing over time. For rescuing in extremely complex environments, a modified approach powered by a distributed system will be investigated. Also, some latest technologies in AI and Machine Learning such as Transfer Learning and Reinforcement Learning will be considered to encounter some circumstances when the data about the locations and the survivors are incorrect or insufficient to train the robots.

Acknowledgements This work was supported by National Natural Science Foundation of China (Nos. 61703188, 61873105), and Natural Science Foundation of Jiangsu Normal University (No. 17XLR042).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interests.

References

1. Balta H, Bedkowski J, Govindaraj S, Majek K, Musialik P, Serrano D (2017) Integrated data management for a fleet of search-and-rescue robots. J Field Robot 34(3):539–582
2. Sawas Y, Takasaki Y (2017) Natural disaster, poverty, and development: an introduction. World Dev 16:2–15
3. Konyo M, Ambe Y, Nagano H, Yamauchi Y (2019) ImPACT-TRC thin serpentine robot platform for urban search and rescue. Disaster robotics. Springer, Berlin
4. Sousa P, Ferreira A, Moreira M, Santos T (2018) ISEP/INESC TEC aerial robotics team for search and rescue operations at the euRathlon 2015. J Intell Robot Syst 5:1–14
5. Zhang HT, Song GA (2017) System centroid position based tipover stability enhancement method for a tracked search and rescue robot. Adv Robot 28(23):1571–1585
6. Shi XM, Zhang Z, Chen XX (2013) Design and research of an active search and rescue robot used in ruin interstice. Adv Mater Res 655–657:1096–1100
7. Raison N, Challacimbe B (2015) The robot to the rescue! Editorial on robotic management of genitourinary injuries from obstetric and gynecological operations: a multi-institutional report of outcomes. BJU Int 115(3):349–350
8. Robin R, Murphy A (2012) A decade of rescue robot. In: IEEE 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 5448–5449.
9. Tian ZJ, Zhang LY, Chen W (2013) Improved algorithm for navigation of rescue robots in underground mines. Comput Electr Eng 39:1088–1094
10. Balaguer B, Balakirsky S, Carpin S, Visser A (2009) Evaluating maps produced by urban search and rescue robots: lessons learned from RoboCup. Auton Robot 27:449–464
11. Sahashi T, Sahshi A, Uchiyama H, Fukumoto I (2013) Study and development of the rescue robot to accommodate victims under earthquake disasters. Lecture notes Electr Eng 174:89–100
12. Kozłowski M, Twomey N, Byrne D, Pope J, Santos-Rodríguez R, Piechocki RJ (2020) H4LO: automation platform for efficient RF fingerprinting using SLAM-derived map and poses. IET Radar Sonar Nav 14(5):694–699
13. Du GD, Zhang P (2015) A markerless human-robot interface using particle filter and Kalman filter for dual robots. IEEE T Ind Electron 62(4):2257–2264
14. Du G, Zhang P, Liu X (2016) Markerless human-manipulator interface using leap motion with interval Kalman filter and improved particle filter. IEEE T Ind Inform 12(2):694–704
15. Xie J, Li XF, Kuang SL (2020) Modeling and motion control of a liquid metal droplet in a fluidic channel. IEEE-ASME T Mecha 25(2):942–950
16. Abdelwahab M, Parque V, Fath Elbab AMR, Abouelsoud AA, Sugano S (2020) Trajectory tracking of wheeled mobile robots using Z-number based fuzzy logic. IEEE Access 8:18426–18441
17. Lee S, Chwa D (2020) Dynamic image-based visual serving of monocular camera mounted omnidirectional mobile robots considering actuators and target motion via fuzzy integral sliding mode control. IEEE T Fuzzy Syst **(early access)**
18. Geng N, Meng QG, Gong DW, Chung PWH (2019) How good are distributed allocation algorithms for solving urban search and rescue problems? A comparative study with centralized algorithms. IEEE T Autom Sci Eng 16(1):478–485
19. Geng N, Gong DW, Zhang Y (2014) PSO-based robot path planning for multisurvivor rescue in limited survival time. Math Probl Eng 2014:1–10
20. Kuwata Y, Takada S (2000) Rescue ability for earthquake causality during the 1995 KOBE earthquake. http://www.1ib.kohe-u.ac.jp/re.pository/00231322.Oaf, Accessed Now 2020
21. Feng X, Lai Z, Yu H (2019) A novel parallel object-tracking behavior algorithm based on dynamics for data clustering. Soft Comput 1–2:1–21
22. Wang J, Wang Y, Zhang D (2018) Multi-task allocation in mobile crowd sensing with individual task quality assurance. IEEE T Mobile Comput 17(9):2101–2113
23. Kurdi HA, Aloboud E, Alalwan M (2018) Autonomous task allocation for multi-UAV systems based on the locust elastic behavior. Appl Soft Comput 71:110–126
24. Jena T, Mohanty JR (2017) GA-based customer-conscious resource allocation and task scheduling in multi-cloud computing. Arab J Sci Eng 1:1–16
25. Pendharkar PC (2015) An ant colony optimization heuristic for constrained task allocation problem. J Comput Sci 7:37–47
26. Tkach I, Jevtić A, Nof SY (2018) A modified distributed bees algorithm for multi-sensor task allocation. Sensors 18(3):1–16
27. Emde S, Polten L, Gendreau M (2020) Logic-based benders decomposition for scheduling a batching machine. Comput Oper Res 113(104777):1–12
28. Li RY, He M, He HY, Deng QY (2020) Heuristic column generation for designing an express circular packaging distribution network. Oper Res-Ger. https://doi.org/10.1007/s12351-020-00570-w
29. Muter I, Birbil SI, Bülbül K (2018) Benders decomposition and column-and-row generation for solving large-scale linear programs with column-dependent-rows. Eur J Oper Res 264(1):29–45
30. Whitbrook A, Meng QG, Chung PWH (2015) A novel distributed scheduling algorithm for time-critical multi-agent system. In: IEEE 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 6451–6458
31. Whitbrook A, Meng QG, Chung PWH (2018) Reliable, distributed scheduling and rescheduling for time-critical, multiagent system. IEEE T Autom Sci Eng 15(2):732–747

32. Turner J, Meng QG, Schaefer G (2018) Distributed task rescheduling with time constraints for the optimization of total task allocations in a multirobot system. IEEE T Cybern 48(9):2583–2597
33. Yu HF(2020) Evaluation of cloud computing resource scheduling based on improved optimization algorithm. Comp Intell Syst 2020 **(early access)**
34. Nethravathi H M, Akhila S (2017) Optimal resource sharing amongst device-to-device communication using Particle Swarm Optimization, Evolutionary Computing and Mobile Sustainable Networks. In: IEEE Conference on Signal Processing and Communications Application, Antalya, Turkey, 2017, pp 1–11
35. Lin S M, Geng N, Sun J, Zhang Yong (2019) A grouping method based on improved PSO for task allocation in rescue environment. In: IEEE Congress on Evolutionary Conputation, Wellington, New Zealand, 2019, pp 619–626
36. Singh MR, Singh M, Mahapatra SS, Jagadev N (2016) Particle swarm optimization algorithm embedded with maximum deviation theory for solving multi-objective flexible job shop scheduling problem. Int J Adv Manuf Technol 85:2353–2366
37. Luo C, Nightingale J, Asemota E, Grecos C (2015) A UAV-cloud system for disaster sensing applications. In: IEEE 81st Vehicular Technology Conference, pp 1–5
38. Song XF, Zhang Y, Guo YN (2020) Variable-size cooperative coevolutionary particle swarm optimization for feature selection on high-dimensional data. IEEE T Evolut Comput 99:1–10
39. Mai G, Hong Y, Fu S, Lin Y (2020) Optimization of Lennard-Jones clusters by particle swarm optimization with quasi-physical strategy. Swarm Evolut Comput 57:1–13
40. Rong M, Gong DW, Gao XZ (2019) Feature selection and its use in big data: challenges, methods, and trends. IEEE Access 7:19709–19725
41. Gendreau M, Laporte G, Seguin R (1995) An exact algorithm for the vehicle routing problem with stochastic demands and customers. Transp Sci 29(2):143–155
42. Kindervater GAP, Savelsbergh MWP (1997) Vehicle routing: handling edge exchanges. Willey, Hoboken
43. Laporte G, Gendreau M, Potvin JY, Semet F (2000) Classical and modern heuristics for the vehicle routing problem. Int T Oper Res 7:285–300
44. Shi YH, Eberhart RC (1998) A modified particle swarm optimizer. IEEE CEC 1998, Anchorage, AK, USA, pp 63–79
45. Choi HL, Brunet L, How J (2009) Consensus-based decentralized auctions for robust task allocation. IEEE T Robot 25(4):912–926