How Good Are Distributed Allocation Algorithms for Solving Urban Search and Rescue Problems? A Comparative Study With Centralized Algorithms

Na Geng, Qinggang Meng, *Senior Member, IEEE*, Dunwei Gong, *Member, IEEE*, and Paul W. H. Chung

***Abstract*— In this paper, a modified centralized algorithm based on particle swarm optimization (MCPSO) is presented to solve the task allocation problem in the search and rescue domain. The reason for this paper is to provide a benchmark against distributed algorithms in search and rescue application area. The hypothesis of this paper is that a centralized algorithm should perform better than distributed algorithms because it has all the available information at hand to solve the problem. Therefore, the centralized approach will provide a benchmark for evaluating how well the distributed algorithms are working and how much improvement can still be gained. Among the distributed algorithms, the consensus-based bundle algorithm (CBBA) is a relatively recent method based on the market auction mechanism, which is receiving considerable attention. Other distributed algorithms, such as PI and PI with softmax, have shown to perform better than CBBA. Therefore, in this paper, the three distributed algorithms mentioned earlier are compared against three centralized algorithms. They are particle swarm optimization, MCPSO, described in this paper, and genetic algorithms. Two experiments were conducted. The first involved comparing all the above-mentioned algorithms, both centralized and distributed, using the same set of application scenarios. It is found that MCPSO always outperforms the other five algorithms in time cost. Due to the high failure rate of CBBA and the other two centralized methods, the second experiment focused on carrying out more tests to compare MCPSO against PI and PI with softmax. All the results are shown and analyzed to determine the performance gaps between the distributed algorithms and the MCPSO.**

***Note to Practitioners*—This paper was motivated by the limitation of current distributed task allocation algorithms as they cannot achieve performances that are as good as the centralized ones. Therefore, a centralized algorithm is designed to evaluate the performance gap between the state-of-the-art distributed and centralized approaches. In the future research section, a new distributed particle swarm optimization (PSO) algorithm is proposed based on this paper as the research has shown that the proposed centralized PSO algorithm delivers the best results so it is potentially a strong candidate for adaptation.**

***Index Terms*— Centralized task allocation, distributed task allocation, particle swarm optimization (PSO), search and rescue.**

I. Introduction

Motivated by recent advances in intelligent systems and cooperative control, researchers have been studying various applications of multivehicle systems to accomplish difficult missions such as search and rescue [1] and logistics [2], [3]. One of the key problems that need to be addressed in multivehicle systems is task allocation, which is to assign vehicles to tasks without conflicts and to minimize a global cost objective or to maximize some global rewards.

The test scenarios in this paper are based on the rescue aspect of search-and-rescue applications. A team of unmanned air vehicles (UAVs) is to be assigned tasks to visit specified locations to supply medicine or food to victims. The multivehicle task allocation problem has been shown to be NP-hard [4]; therefore, obtaining a globally optimal solution is computationally intensive for complex scenarios.

Multivehicle task allocation problems are often solved in a distributed manner such as using the market-based mechanisms [5], [6]. The auction approaches are among the most popular since they are efficient in terms of both computation and communication [7]. Recently, consensus-based bundle algorithm (CBBA) [8], [9] has attracted more interest. It is a distributed algorithm that iterates between the bundle construction phase and the conflict resolution phase. Based on it, Turner *et al*. [10] presented a decentralized method for maximizing the number of tasks allocation under strict time constraints. It is based on auction theory, and multiple reassignments among networked robots may be required to create a feasible time according to the performance requirements. Whitbrook *et al*. [11] addressed two main problems, i.e., solution trapped in local minima and static structure, with many heuristic task allocation approaches. The existing distributed task allocation algorithm known as performance impact is used to solve the above-mentioned problems. Zhao *et al*. [1] proposed a novel heuristic distributed task allocation algorithm, i.e., the PI algorithm, for multivehicle multitask assignment problems, and a new concept of significance was defined for each task. "Significance" is a measure of the local cost to a vehicle if it was to execute that task. Whitbrook *et al*. [12] modified the proposed PI algorithm by integrating ε-greedy and Soft-max-based action selection strategies; simulation results showed that the modified algorithms perform better on the number of tasks allocated than PI algorithm but slower than the PI algorithm.

This work was supported in part by the National Natural Science Foundation of China under Grant 61703188, Grant 61673404, and Grant 61873105, in part by the National Basic Research Program of China under Grant 2014CB046306-2, in part by the Natural Science Foundation of Jiangsu under Grant BK20160219, and in part by the Natural Science Foundation of Jiangsu Normal University under Grant 16XLR043. This paper was recommended for publication by Associate Editor S. Genc and Editor D. Popa upon evaluation of the reviewers' comments. *(Corresponding author: Dunwei Gong.)*

N. Geng is with the School of Electrical Engineering and Automation, Jiangsu Normal University, Xuzhou 221116, China (e-mail: gengna@126.com).

Q. Meng and P. W. H. Chung are with the Department of Computer Science, Loughborough University, Loughborough LE11 3TU, U.K. (e-mail: q.meng@lboro.ac.uk; p.w.h.chung@lboro.ac.uk).

D. Gong is with the School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221008, China, and also with the School of Information Science and Technology, Qingdao University of Science and Technology, Qingdao 266061, China (e-mail: dwgong@vip.163.com).

Some researchers have used centralized approaches to solve the multivehicle task allocation problems [4], [13], [14]; among all these approaches, intelligent optimization algorithms are the most widely used. Zhu *et al*. [15] proposed an integrated multiple autonomous underwater vehicle dynamic task assignment and path planning algorithm by combing the improved self-organizing map neural network and a novel velocity synthesis approach. Pujol-Gonzalez *et al*. [16] focused on RoboCup rescue simulation challenge and used the max-sum approach to solve the problem of multi-team allocations.

Many evolutionary algorithms were also employed in the field of task allocation; for example, Salman *et al*. [17] presented a new algorithm based on particle swarm optimization (PSO), and its performance was evaluated in comparison with the well-known genetic algorithm (GA). The results showed that the solution quality of the PSO algorithm is better than that of GA in most cases. In addition, the PSO algorithm runs faster compared with GA in experiments [18].

PSO is a computational method that optimizes a problem by iteratively improving a candidate solution with regards to a given measure of quality [19]. The population of candidate solutions is represented in the problem solution search space, and they are moved around by altering their velocities and positions according to some simple mathematical formulas [20].

PSO is a metaheuristic algorithm as it makes no, or few, assumptions about the problem being optimized and is suitable for searching very large solution spaces. More specifically, PSO does not use the gradient of the problem being optimized, which means it does not require the optimization problem to be differentiable as required by methods such as gradient descent and quasi-Newton methods. PSO is therefore applicable to optimization problems that are partially irregular, noisy, and adaptive [21].

Another advantage PSO has over other optimization techniques is that it can be implemented in a few lines of computer codes using just a few primitive mathematical operators [22], [23]. Therefore, in this paper, PSO is chosen as the centralized approach for comparing with the advanced distributed algorithms. Two PSO algorithms, PSO and modified centralized algorithm based on PSO (MCPSO), are used. Another centralized approach chosen is GA, which is a popular random search method for solving optimization problems.

From previous studies, we can conclude that even though the task allocation problem has been studied for years, dealing with complex task scenarios is still an open problem. In this paper, we investigate the use of centralized algorithms to solve the multivehicle task allocation problem. As our assumption is that a well-designed centralized algorithm should outperform any distributed algorithms because it has all the available information, therefore, the proposed algorithm, named MCPSO, is compared against distributed task allocation algorithms to gain an insight into how well those distributed algorithms actually work.

The contributions of this paper are as follows.

1) A benchmark algorithm is proposed to estimate the difference between centralized and distributed algorithm.
2) A priority sequence and a new decode method are given.
3) The insert operation and local search methods are presented.
4) New methods for updating global and local best solutions are proposed.
5) Two experiments are conducted to verify the proposed method.

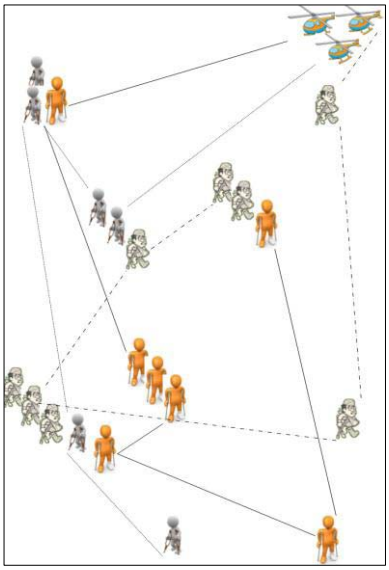


Fig. 1. Rescue environment.

II. Problem Description

The rescue environment is shown in Fig. 1; we can see that there exist different types of survivors (tasks) as indicated by different icons. The vehicles are assigned tasks so that collectively they will be able to rescue all the survivors. The goal of task allocation is to schedule and assign a sequence of tasks to each available vehicle without any conflicts among all vehicles. The same assumptions are made as stated in [12].

More specifically, we select one optimization criteria to achieve the objective, e.g., minimum cost. In this paper, the aim is to minimize the sum of the cost of all paths over all vehicles. In addition, the path cost includes the travel time and rescue time. The minimum average time is used as the problem's objective.

To formulate the problem mathematically, a list of key symbols used is provided in Table I. The aim of the problem is to assign the m locations to n heterogeneous unmanned vehicles with the minimum average time. The objective for the scenario can be expressed as

$$\min F = \min\left\{\frac{1}{m}\sum_{i=1}^{n}\sum_{k=1}^{\alpha_i} c_{i,k}(a_i)\right\}. \tag{1}$$

III. MCPSO for Task Allocation Method

In this section, the proposed MCPSO algorithm is presented, which is a modification of the basic PSO algorithm. The basic idea of PSO is first described in Section III-A, followed by the contributions of the proposed algorithm for optimizing the stated objective.

TABLE I
SYMBOL DEFINITIONS

Notation	Definitions
$Ve=[ve_1,\cdots,ve_n]^T$	the set of n vehicles
$T=[t_1,\cdots,t_m]^T$	the set of m tasks
$A=[a_1,\cdots,a_n]^T$	an allocation, a partition of T
a_i	an ordered list of tasks to be rescued by vehicle ve_i.
$c_{i,k}(a_i)$	the time cost of task t_k in a_i
Distance(i,j)	the distance between tasks i and j
Time(i,j)	time needed for a vehicle between tasks i and j
Endtime(j)	the time limitation for task j
DifTime(i,j)	the difference between time(i,j) and endtime(j)
$f(\cdot)$	the minimum time cost
$fail(\cdot)$	the number of failures
curPar	the current particle
Set1	the set of possible tasks
Set2	the set of impossible tasks
Priority	a priority sequences
Ma	Numbers of tasks assigned to vehicle a
Mb	the numbers of tasks assigned to vehicle b
n_i	the numbers of vehicles performing the same kind of tasks
$\leftrightarrow$	exchange
$\lceil\ \rceil$	a ceiling function
$\lfloor\ \rfloor$	a floor function
$\vec{x}_i=(x_{i1},x_{i2},\cdots,x_{in})$	the position of the ith particle
$\vec{v}_i=(v_{i1},v_{i2},\cdots,v_{in})$	the velocity of the ith particle
lbest: $\vec{p}_i=(p_{i1},p_{i2},\cdots,p_{in})$	the local best solution
gbest: $\vec{p}_g=(\vec{p}_{g1},\vec{p}_{g2},\cdots,\vec{p}_{gn})$	the global best solution

A. Basic PSO

The PSO algorithm is inspired by how a flock of birds seeks food collaboratively. 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. The bird is abstracted as a particle without weight and volume. Each particle owns a fitness value determined by the function that needs to be optimized, and a record of the best location so far is kept, i.e., lbest. Each particle also knows the best global location, i.e., gbest. The particles determine their next and further movements according to their own and their companions' experiences. Taking the PSO algorithm with inertia weight [24] as an example, the updated formulas of a particle's velocity and location are as follows:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1r_1(p_{ij}(t) - x_{ij}(t)) + c_2r_2(p_{gj}(t) - x_{ij}(t)) \quad (2)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (3)$$

where w is the inertia weight, which is a user-specified parameter, c_1 and c_2 are the positive constants called acceleration coefficients, r_1 and r_2 are the random numbers in the range of [0, 1], $j = 1, 2, \ldots, n$; $i = 1, 2, \ldots m$, and m is the size of the swarm and n is the dimension of the particles.

B. Priority Determination

If the distance between a rescue location and a vehicle is too far or the mission's end time is too short, then the vehicle is unable to reach the location before the end time even if it is the first task in the schedule. In this paper, we call it an impossible task. It is better to ignore such tasks to allow time for other tasks.

Priorities are set for the tasks according to the time cost between the rescue locations and vehicles. The priority is determined by the time difference between the end time and the travel time from the vehicle to a location. If the time difference is greater than zero but small, then the priority of the task for the vehicle is high; if the time difference is greater than zero but the value is large, then the priority for the vehicle is low; otherwise, the priority is set to the lowest. Priority assignment is given in Algorithm 1.

Algorithm 1 Getting the Priority of Each Task

Input: locations of tasks and vehicles, end time of each task
Output: priority set of each vehicle
1: **For** $i = 1$ to n
2: **For** $j = 1{:}m$
3: Distance(i, j) = |vehicle(i)-task(j)|
4: Time(i, j) = Distance(i, j)/velocity(i)
5: DifTime(i, j) = Endtime(j)-Time(i, j)
6: **End**
7: **If** DifTime(i,j) > 0
8: Set1(i) = {Set1(i) $\cap$ j}
9: **Else**
10: Set2(i) = {Set2(i) $\cap$ j}
11: **End**
12: Priority = { sort(Set1(i)) $\cap$ sort(Set2(i)) }
13:**End**

C. Particle Coding and Decoding Methods

One particle represents one solution, and it should contain all of the tasks that need to be carried out. For simplicity, the length of particle coding is the same as the number of rescue locations that need to be visited. Since there are m locations, the length of the particle coding is set to m.

In order to calculate the average time cost, we need to know the rescue sequence, so the decoding strategy of the particle is needed. In this paper, an integer coding method is used to determine the task sequence. Assuming the coding of a particle is $x_i = (x_{i1}, x_{i2}, \ldots, x_{im})$ and after decoding the task sequence is $X_i = (X_{i1}, X_{i2}, \ldots, X_{im})$, that there are u kinds of tasks to be carried out, and the task numbers for each vehicle are

$$tN = \{tN_1, tN_2, \cdots, tN_u\} = \lceil m/u \rceil, \left\lceil \frac{m - \lceil m/u \rceil}{u-1} \right\rceil, \cdots, m - \left\lceil \frac{m}{u} \right\rceil - \left\lceil \frac{m - \lceil m/u \rceil}{u-1} \right\rceil - \cdots$$

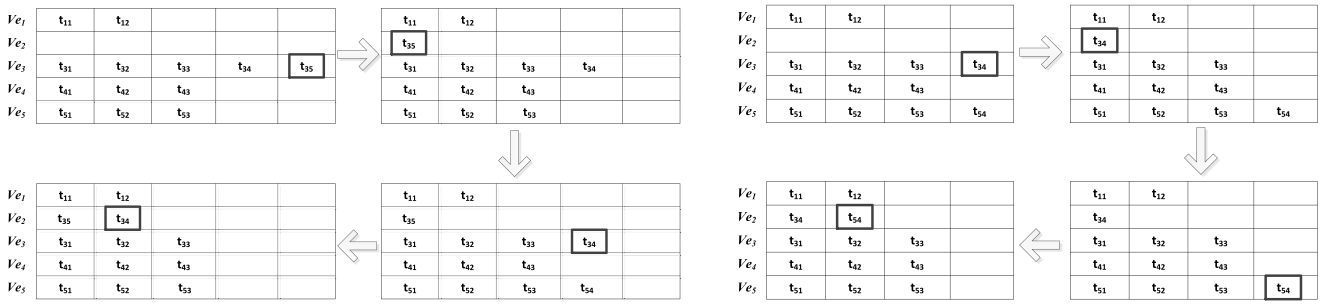


Fig. 2. Insert operation. (a) One vehicle has the max number of tasks. (b) Two vehicles have the max number of tasks.

and the number of vehicles for each type is

$$vN = \{vN_1, vN_2, \cdots, vN_u\} = \left\lceil \frac{n}{u} \right\rceil, \left\lceil \frac{n - \lceil n/u \rceil}{u-1} \right\rceil, \cdots, n - \left\lceil \frac{n}{u} \right\rceil - \left\lceil \frac{n - \lceil n/u \rceil}{u-1} \right\rceil - \cdots$$

Then, the decoding method is as described as follows.

First, for the first kind of tasks, the former $\lceil m/u \rceil$ elements in x_i, i.e., $x_i^1 = (x_{i1}, x_{i2}, \ldots, x_{itN_1})$, the length of x_i^1 is $\lceil m/u \rceil$; for the first element x_{i1}, apply the mod and add operations, i.e., mod $(x_{i1}, \lceil n/u \rceil)+1$, the result is an integer representing the numerical order of the vehicle to which the first task is allocated. The process is repeated until all the elements in x_i^1 are assigned. Second, for the remaining elements in x_i, e.g., $x_i^2 = (x_{i(tN_1+1)}, \ldots, x_{i(tN_u-1)}, x_{itNu})$, it is also necessary to calculate which task is assigned to which vehicle by applying the mod and add operations, i.e., mod $(x_{i(tN_2+1)}, vN_2) + \lceil n/u \rceil + 1$. The process continues until all elements in x_i are assigned to a task sequence for each vehicle. Third, since the number of tasks is larger than the number of vehicles, a vehicle may have more than one task assigned to it so the assigned tasks will need to be reordered.

When multiple tasks are assigned to the same vehicle, then the rescue sequence is determined according to the priority described later in Section III-B.

D. Lbest and Gbest Solutions Update Methods

In this paper, the aim is to obtain the minimum mission time cost to complete all tasks with the minimum number of failures. The following method is used to obtain lbest and gbest at each iteration with the condition that the final solution has the tasks assigned with the minimum number of failures. The update method is given in Algorithm 2.

E. Particle Velocity and Location Update Method

As particles use an integer coding method, the update formulas for the particle's velocity and location are different from those of the common PSO. The following formulas are used instead:

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_1 (p_{ij}(t) - x_i(t)) + c_2 r_2 (g_j(t) - x_{ij}(t)) \quad (4)$$

$$x_{ij}(t+1) = \lceil x_{ij}(t) + v_{ij}(t+1) \rceil. \quad (5)$$

Algorithm 2 Local and Global Best Solutions Update

Input: *lbest*, *gbest*, fail, and the current particle
Output: *lbest*, *gbest*, fail
If fail(curPar) < fail(*lbest*)
    lbest = curPar
Elseif (f(*lbest*) > f (curPar)) and (*fail*(curPar) = *fail*(*lbest*))
    lbest = curPar
Else
    lbest = *lbest*
End
If *fail*(*gbest*) > *fail* (*lbest*)
    gbest = *lbest*
Elseif (f (*gbest*) > f (*lbest*)) and (*fail*(*lbest*) = *fail*(*gbest*))
    gbest = *lbest*
Else
    gbest = *gbest*
End

F. Insert Operation

PSO is a stochastic algorithm so there may exist situations such that some vehicles are assigned to no task, or very few tasks, and other vehicles are assigned to too many tasks. This will result in mission failure or a high mission time cost. In order to avoid such situations, an insert operation is proposed.

First, the number of tasks assigned to each vehicle is calculated. The vehicles with the largest and smallest number of tasks are noted. Second, determine whether the smallest number of tasks assigned to a vehicle is lower than $[m/n]$. If yes, delete the last task in the task list of the vehicle with the largest number of tasks, and insert it into the vehicle with the smallest number of tasks. Repeat the above-mentioned steps until the smallest number of tasks assigned to a vehicle is equal to $[m/n]$. After the iteration terminates, the new task list is reordered according to the priority set. Fig. 2 shows an example.

G. Local Search Methods

After applying the gbest update method, local methods are employed to increase the convergence speed to find better solutions. However, it is time consuming to apply local search

Algorithm 3 Modified 2-OPT Method

Input: *gbest*, and fail
Output: *gbest*
1: **For** $i = 1$: u
2: **For** $j = 1$:ni
3: [*a,b*]=randint(2,1, ni), and $b \neq a$
4: c = randint(1, ma), d = randint(1, mb),
5: task $c \leftrightarrow$ task d
6: sort task list according to the priority set
7: **If** total time cost improves or failure time declines
8: modify the sequence
9: **End**
10: **End**
11: **End**

Algorithm 4 Modified Exchange Method

Input: *gbest*, and fail
Output:*gbest*
1: **For** $i = 1$ to u (number of vehicle types)
2: $[k \ j] =$ randint(2, 1, ni), and $k \neq j$
3: $a_k \leftrightarrow a_j$
4: sort the task list according to the priority set
5: **If** solution improves,
6: modify the task list
7: **End**
8: **End**

methods to all of the particles or even a portion of the particles. Therefore, local search methods are performed on gbest after decoding.

We modified the 2-opt and exchange operations as described in [25] and used them in this paper for their advantages. 2-opt is a well-known local search method [26] with an $O(n^2)$ computational complexity that aims to improve a solution by changing the sequence of tasks for each vehicle. However, in this paper, the task sequence for each vehicle is sorted according to the priority order, so changing the sequence of tasks within a task list may have the opposite effect. Therefore, changes take place between two different types of vehicles rather than the same type. The detailed steps of this operation are given in Algorithm 3 (note: the modified 2-opt method in this paper can only be done between the vehicles of the same task type).

Exchange operation [25] does not change the positions between tasks among vehicles of the same type; it changes the tasks between different vehicles with different types, as different types of vehicles are located in different positions (note: the exchange operation only can be performed between vehicles that carry out the same type of tasks). The details are given in Algorithm 4.

Algorithms 3 and 4 are combined into Algorithm 5. If an improved solution is found, then the algorithm stops. Otherwise, it carries on until the maximum number of iterations is reached.

Algorithm 5 Local Search Method

Input: *gbest*, and fail
Output: *gbest*
1: **If (iter1** $<$ **maxiter1)**
2: perform modified 2-opt method
3: **If** solution improves
4: modify the task list
5: **Break**
6: **Else**
7: **If** (iter2 $<$ maxiter2)
8: perform modified exchanged method
9: **If** solution improves
10: modify the task list
11: **Break**
– 12: **Else**
13: record the best one so far
14: **End**
15: iter2 = iter2+1
16: **End**
17: **End**
18: iter1 = iter1+1;
19:**End**
20:Output the *gbest* after coding the new generated task list

H. Local Optimum Avoidance

Although the PSO algorithm has many advantages, it may trap into a local optimum. If the particles cannot jump out of the local optimum, then the whole swarms will be trapped. The following strategy is used to avoid trapping into a local optimum.

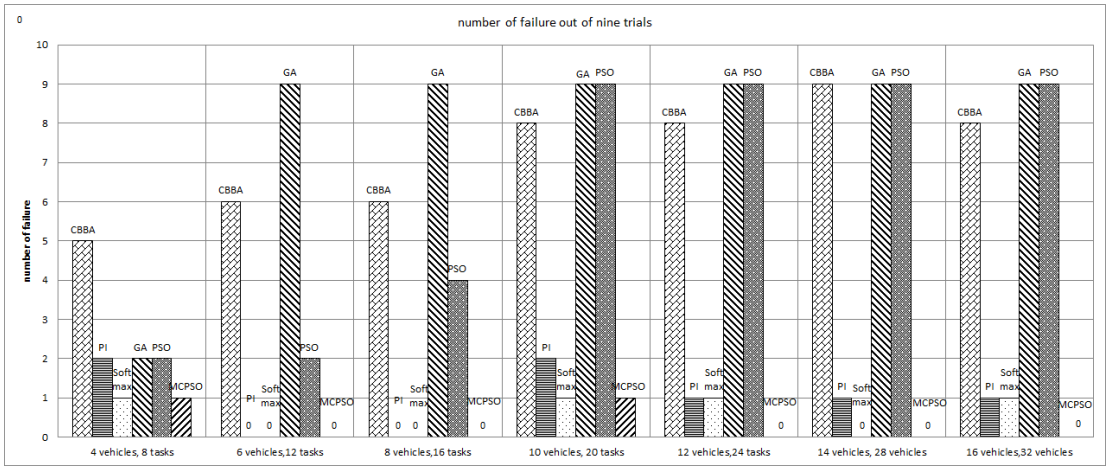
First, the global best solutions of the current and previous generations are compared and the difference between the two is calculated. Second, count the number of consecutive gains that are smaller than a set threshold. The threshold number is to evaluate whether the optimum value is trapped into a local optimum. If the number is larger than the threshold, then the population is reinitialized.

IV. Overview of Simulation Configurations

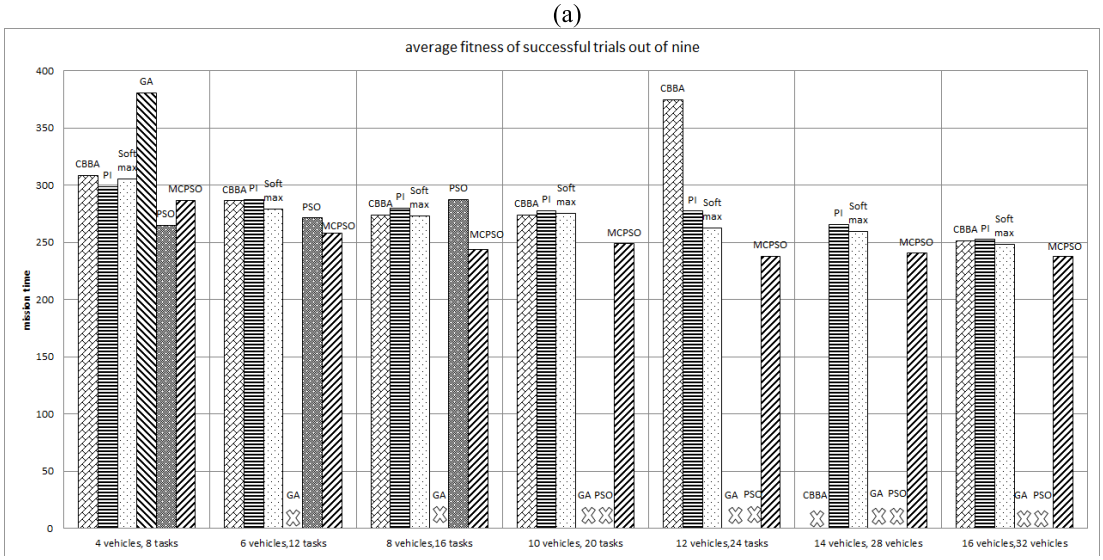
This section presents the simulation scenarios for evaluating the proposed centralized task allocation method together with other centralized and decentralized methods. The scenario description and its corresponding results are introduced and analyzed first in Section V, followed by the detailed simulation results in Section VI obtained from a large number of scenarios with different initial settings using random seeds and different numbers of vehicles and tasks.

Since CBBA and its modified versions are among the most popular market-based methods to solve the task allocation problem, MCPSO is compared with CBBA, PI, and PI with softmax. In order to further illustrate the effectiveness of MCPSO, it is also compared with the PSO and GA.

Two experiments are carried out. The first one is based on the search-and-rescue scenarios reported in [1] and [12]; the second one is conducted using more difficult scenarios to test MCPSO further. The scenarios used in Section VI are



(a)



Note: in the figure, × represents the algorithm cannot solve the problem, and it is indicated as N/A in the tables.

(b)

Fig. 3. Comparison results between MCPSO and the other five methods in seven different scenarios. (a) Total number of failure for nine missions in each scenario. (b) Average mission time for all nine missions in each scenario.

more complex than those in Section V with a larger number of vehicles and tasks and also the parameter values are smaller; besides, there is no situation that the ratio between the number of the tasks and the vehicles is 2.

V. Experiment One

A. Test Scenario

In this section, MCPSO is evaluated for multivehicle multitask allocation problem based on the 69 search-and-rescue scenarios reported in [1] and [12].

In this experiment, nine seeds are used to generate the 3-D scenarios with 4, 6, 8, 10, 12, 14, and 16 vehicles, i.e., 63 different problems are tackled. The corresponding number of tasks is exactly double the number of vehicles, and the number of helicopters and UAVs is always the same. The problems are solved using CBBA, the PI algorithm, the modified PI algorithm with softmax, PSO, GA, and MCPSO. If the problem is solved, i.e., all tasks are scheduled to complete on time, the fitness is calculated and recorded. If some tasks are not completed within the allocated time, then the number of failed tasks for each task type will be recorded instead.

For all the test scenarios, the x and y coordinates range from -5000 to 5000 m and the z coordinate ranges from 0 to 1000 m. In addition, the unmanned helicopters travel at 30 m/s and the UAVs at 50 m/s, and all the vehicles are available from the time it is scheduled to start. The time window within which the scenario must finish is set at 2000 s and the earliest start time is 0 s for a task. The latest start time is generated for each task using a random fraction of 2000 s. The time set for delivering medicine is 300 s, and the time needed to supply food is set to 350 s.

The parameters of MCPSO are set the same as [27]. All the experiments are conducted using MATLAB R2013a in the same 64-bit machine running Windows 7 Enterprise Edition and using a 2.5-GHz Intel Core i5-2400S Processor.

B. Simulation Results and Discussion

Fig. 3 shows the comparison results among CBBA, PI, PI with softmax, GA, PSO, and MCPSO using the 63 problems (nine missions in seven configurations) presented by Zhao *et al*. [1]. Fig. 3(a) shows the total number of failures of all nine missions for the seven configurations.

TABLE II
AVERAGE ALGORITHM RUNING CPU TIMES (SECONDS)

N	M	CBBA	PI	Soft max	PSO	GA	MCPSO
4	8	0.37	**0.14**	6.76	1.183	1.272	6.04
6	12	0.89	**0.23**	15.39	1.727	N/A	8.79
8	16	0.86	**0.52**	52.83	2.124	N/A	13.42
10	20	1.25	**1.16**	46.33	N/A	N/A	16.70
12	24	N/A	**2.00**	91.82	N/A	N/A	18.87
14	28	N/A	**2.46**	143.71	N/A	N/A	22.98
16	32	**2.33**	4.20	70.78	N/A	N/A	26.88

Fig. 3(b) shows the average mission time of the successful missions for each scenario.

Table II summarizes the CPU times for all the algorithms for all the scenarios, and the best results are highlighted in bold.

From Fig. 3, we can see that CBBA has a high failure rate. Out of 63 problems, CBBA is only able to solve 13 of them and none of them is one best solution when compared with the other algorithms. However, CBBA is fast, only slightly slower than the PI algorithm.

The PI algorithm is modified from the CBBA algorithm, and it inherits the merits of the original algorithm. It can solve the different scenarios very quickly even when there are many vehicles and tasks. All the scenarios, although not all nine missions in each of the configuration, can be solved by PI. However, PI cannot offer one best solution out of the 63 problems. Hence, the fitness values are in the middle place among all algorithms.

The PI with softmax algorithm consistently outperforms the original PI algorithm with a small margin but it cannot offer any best solution either. However, it is the slowest of all the algorithms.

PSO solved 18 out of all the 63 problems, and three of them are the best solutions. The PSO algorithm is faster than PI with softmax, MCPSO, and GA, but it is slower than PI and CBBA. However, PSO cannot solve the complex problems, i.e., with more than 10 vehicles and 20 tasks.

GA, which is modified to solve the task allocation problem, has some improvements, including the coding and decoding methods (the same as PSO), modified one-point crossover operator, and modified selection operator (adding the number of failures as an index to select the best solution). The parameters in GA are given the same values as those of MCPSO. Fig. 3 shows that GA only solved 7 out of the 63 problems, and none of them is the best solution; besides, its CPU running time is a bit longer than that of PSO.

For MCPSO, there are only four problems unsolved out of the 63, and 56 out of the 59 solutions are the best results. For MCPSO, the CPU running time increases quite significantly with the increase of the numbers of vehicles and tasks. However, the time required is still acceptable for the scenarios considered.

VI. EXPERIMENT TWO

A. Test Scenario

A limitation of experiment one is that for each scenario, the number of tasks is always twice the number of vehicles. In experiment two, a wider range of more realistic scenarios is considered by changing the parameters for the scenario simulation platform, enlarging the search and rescue environment, and extending the end time for each mission. In addition, the number of the tasks is increased to make the scenarios more complex, and the number of tasks is not exactly twice that of the vehicles. Eight vehicles are used to carry out 20, 30, 40, 60, 80, and 100 rescue tasks. For each scenario, nine seeds are used to generate nine missions.

The parameters in the simulation experiment are changed as well. In all algorithms, the world x and y coordinates range from $-10\,000$ to $10\,000$ m, and z coordinates range from 0 to 1000 m. The time window within a mission must finish is set as $20\,000$ s for all algorithms, and the earliest start time is always 0 s for all tasks. The latest start time was generated for each task using a random fraction of $20\,000$ s. Other parameters stay the same as experiment one.

Since GA, CBBA, and PSO produced high failure rates in experiment one, in this experiment, MCPSO is only compared with PI and PI with softmax.

B. Simulation Results and Discussion

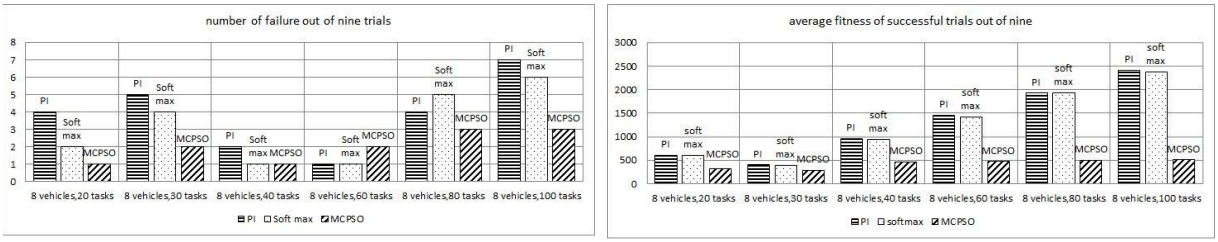
Fig. 4 shows the results for PI, PI with softmax, and MCPSO from six scenarios, each with nine missions. Fig. 4(a) shows the total number of failures out of nine missions in a scenario for each algorithm. Fig. 4(b) shows the average fitness, i.e., the average mission time, for all successful missions for each algorithm.

Table III shows the CPU times taken to tackle the scenarios.

Fig. 4 shows that all three methods can solve all the missions with 20 and 30 tasks but with several failure missions. PI can solve them significantly faster than the other two methods, but its average mission time is the worst one, and its failure number is the largest; while for its modified version, PI with softmax method, it can solve both of the two problems within nine missions but cannot offer any best results for these six different missions, and its failure number ranks the second among the three algorithms. MCPSO method offers six best results and the differences between PI and PI with softmax is big, its failure time is the least, and MCPSO makes a huge improvement.

Fig. 4 also demonstrates that MCPSO can solve most of the missions in a configuration with just a small number of failures. When the task number is 40, 60, 80, and 100, respectively, MCPSO performs the best in offering the best results and least failure times. While for the other rescue configurations where there are eight vehicles and 60 tasks, it tells a different story. MCPSO cannot reduce the failure time; among the total of nine missions, it fails twice, and PI and PI with softmax only fail once. However, our method offers seven best results including the failure times, and the average results are the best.

Table III shows the CPU running time of the three methods. PI with softmax consumes the longest time. When the tasks number is small, the PI runs the fastest. For example, when there are eight vehicles and 40 tasks, the CPU running time of PI is just 7.27 s, and when the number of tasks increases to 60, the CPU running time of PI is 17.48 s, which is 0.95 s shorter than MCPSO (18.43 s). However, when the number



(a) (b)

Note: to make the figures more readable, PI with softmax is referred to as softmax.

Fig. 4. Results for MCPSO, PI, and PI with softmax methods in six different missions. (a) Total number of failure for all successful missions out of nine missions. (b) Average mission time for all nine missions.

TABLE III
AVERAGE CPU TIMES (SECONDS) FOR THE SIX MISSIONS

N	M	PI	PI with softmax	MCPSO
8	20	**1.37**	15.93	13.22
8	30	**2.18**	40.60	29.12
8	40	**7.27**	122.99	13.78
8	60	**17.48**	311.23	18.43
8	80	35.53	700.21	**21.09**
8	100	57.67	1166.83	**23.04**

of tasks is higher, e.g., above 80, MCPSO outperforms PI in terms of CPU running time.

VII. CONCLUSION

From the above-mentioned experiments, we can see that MCPSO can solve almost all scenarios in this paper. Even in complex scenarios, MCPSO only fails 10 times, while the PI fails 23 times, and PI with softmax fails 19 times. These experiments demonstrate that the proposed centralized MCPSO solves more problems and with better solutions than state-of-the-art distributed algorithms. Therefore, the development of distributed algorithms still has a long way to go to improve their performance. In closing, we can see that MCPSO can be used as a benchmark for distributed algorithms for determining the performance gap, which will guide further development.

REFERENCES

[1] W. Zhao, Q. Meng, and P. W. H. Chung, "A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario," *IEEE Trans. Cybern.*, vol. 46, no. 4, pp. 902–915, Apr. 2016.

[2] L. Pallottino, V. G. Scordio, A. Bicchi, and E. Frazzoli, "Decentralized cooperative policy for conflict resolution in multivehicle systems," *IEEE Trans. Robot.*, vol. 23, no. 6, pp. 1170–1183, Dec. 2007.

[3] D. Shishebori, "Reliable multi-product multi-vehicle multi-type link logistics network design: A hybrid heuristic algorithm," *J. Indusmission Syst. Eng.*, vol. 9, no. 1, pp. 92–108, 2016.

[4] W. Guo, J. Li, G. Chen, Y. Niu, and C. Chen, "A PSO-optimized real-time fault-tolerant task allocation algorithm in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3236–3249, Dec. 2015.

[5] D. Jiang, Y. Pang, and Z. Qin, "Coordinated control of multiple autonomous underwater vehicle system," in *Proc. 8th World Congr. Intell. Control Autom.*, Jul. 2010, pp. 4901–4906.

[6] Y. Eun and H. Bang, "Cooperative task assignment/path planning of multiple unmanned aerial vehicles using genetic algorithm," *J. Aircraft*, vol. 46, no. 1, pp. 338–343, 2009.

[7] K. Zhang, E. G. Collins, Jr., and D. Q. Shi, "Centralized and distributed task allocation in multi-robot teams via a stochastic clustering auction," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, no. 2, 2012, Art. no. 21.

[8] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. Robot.*, vol. 25, no. 4, pp. 912–926, Aug. 2009.

[9] S. D. Vries and R. Vohra, "Combinatorial auctions: A survey," *Discussion Papers*, vol. 15, no. 3, pp. 284–309, 2000.

[10] J. Turner, Q. Meng, G. Schaefer, A. Whitbrook, and A. Soltoggio, "Distributed task rescheduling with time constraints for the optimization of total task allocations in a multirobot system," *IEEE Trans. Cybern.*, vol. 48, no. 9, pp. 2583–2597, Sep. 2018.

[11] A. Whitbrook, Q. Meng, and P. W. H. Chung, "Reliable, distributed scheduling and rescheduling for time-critical, multiagent systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 732–747, Apr. 2018.

[12] A. Whitbrook, Q. Meng, and P. W. H. Chung, "A novel distributed task allocation algorithm for urban search and rescue," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Hamburg, Germany, Sep. 2015, pp. 6451–6458.

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

[14] J. Yang, H. S. Zhang, Y. Ling, C. Pan, and W. Sun, "Task allocation for wireless sensor network using modified binary particle swarm optimization," *Sensors J.*, vol. 14, no. 3, pp. 882–892, Mar. 2014.

[15] D. Zhu, H. Huang, and S. X. Yang, "Dynamic task assignment and path planning of multi-AUV system based on an improved self-organizing map and velocity synthesis method in three-dimensional underwater workspace," *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 504–514, Apr. 2013.

[16] M. Pujol-Gonzalez, J. Cerquides, A. Farinelli, P. Meseguer, and J. A. Rodríguez-Aguilar, "Binary max-sum for multi-team task allocation in RoboCup rescue," in *Proc. Int. Joint Workshop Optim. Multi-Agent Syst. Distrib. Constraint Reasoning*, Paris, France, 2014, pp. 1–15.

[17] A. Salman, I. Ahmad, and S. Al-Madani, "Particle swarm optimization for task assignment problem," *Microprocessors Microsyst.*, vol. 26, no. 8, pp. 363–371, 2002.

[18] D. R. Karger, S. Oh, and D. Shah, "Budget-optimal task allocation for reliable crowdsourcing systems," *Oper. Res.*, vol. 62, no. 1, pp. 1–24, 2014.

[19] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, Washington, DC, USA, 1995, pp. 1942–1948.

[20] J. Kennedy, "The particle swarm: Social adaptation of knowledge," in *Proc. IEEE Int. Conf. Evol. Comput.*, Indianapolis, IN, USA, Apr. 1997, pp. 303–308.

[21] P. Vasant, *Meta-Heuristics Optimization Algorithms in Engineering Business, Economics and Finance, Information Science Reference*. New York, NY, USA: Springer, 2012, pp. 6–7.

[22] D. Ruan, *Computational Intelligence in Complex Decision Making Systems*. Singapore: World Scientific, 2010, p. 23.

[23] K. E. Parsopoulos and M. N. Vrahatis, "Recent approaches to global optimization problems through particle swarm optimization," *Natural Comput.*, vol. 1, nos. 2–3, pp. 235–306, 2002.

[24] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann, 2001, pp. 475–495.

[25] J.-F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.-S. Sormany, "New heuristics for the vehicle routing problem," in *Logistics Systems: Design and Optimization*. New York, NY, USA: Springer, 2005, pp. 279–297.

[26] G. Laporte, M. Gendreau, J.-Y. Potvin, and F. Semet, "Classical and modern heuristics for the vehicle routing problem," *Int. Trans. Oper. Res.*, vol. 7, nos. 4–5, pp. 285–300, 2000.

[27] R. C. Eberhart and Y. Shi, "Particle swarm optimization: Developments, applications and resources," in *Proc. Congr. Evol. Comput.*, vol. 1. May 2001, pp. 81–86.