Adaptive Coordination Ant Colony Optimisation for Multi-Point Dynamic Aggregation

Guanqiang Gao, Yi Mei, *Senior Member, IEEE,* Ya-hui Jia, *Member, IEEE,* Will N. Browne, *Member, IEEE,* and Bin Xin, *Member, IEEE*

***Abstract*—Multi-point dynamic aggregation is a meaningful optimisation problem, due to its important real-world applications such as post-disaster relief, medical resource scheduling, and bushfire elimination. The problem aims to design the optimal plan for a set of robots to execute geographically distributed tasks. Unlike the majority of scheduling and routing problems, the tasks in this problem can be executed by multiple robots collaboratively. Meanwhile, the demand of each task changes over time at an incremental rate and is affected by the abilities of the robots executing it. This poses extra challenges to the problem, as it has to consider complex coupled relationships among robots and tasks. To effectively solve the problem, this paper develops a new meta-heuristic algorithm, named adaptive coordination ant colony optimisation. We develop a novel coordinated solution construction process using multiple ants and pheromone matrices (each robot/ant forages a path according to its own pheromone matrix) to effectively handle the collaborations between robots. We also propose adaptive heuristic information based on domain knowledge to promote efficiency, a pheromone-based repair mechanism to tackle the tight constraints of the problem, and an elaborated local search to enhance the exploitation ability of the algorithm. Experimental results show that the proposed adaptive coordination ant colony optimisation significantly outperforms the state-of-the-art methods in terms of both effectiveness and efficiency.**

***Index Terms*—Ant Colony Optimisation, Multi-Robot System, Task Allocation, Multi-Point Dynamic Aggregation**

I. Introduction

The Multi-Point Dynamic Aggregation (MPDA) problem [1] widely exists in real-world applications, such as bushfire elimination routing, post-disaster relief, and medical resource scheduling domains [2]–[7]. For a better understanding of this problem, Fig. 1 shows a team of robots equipped with fire-elimination devices to eliminate a bushfire. As a fire grows over time, the robots have to increase their efforts to extinguish the fire as time goes. On the other hand, the robots can collaborate to eliminate the fire more efficiently (i.e. multiple robots eliminating the same fire point simultaneously).

This work was supported in part by the National Outstanding Youth Talents Support Program 61822304, in part by the National Natural Science Foundation of China under Grant 61673058, in part by the NSFC-Zhejiang Joint Fund for the Integration of Industrialization and Informatization under Grant U1609214, in part by the Projects of Major International (Regional) Joint Research Program of NSFC under Grant 61720106011, and in part by Consulting Research Project of the Chinese Academy of Engineering (2019-XZ-7). (Corresponding author: Bin Xin.)

Guanqiang Gao and Bin Xin are with the School of Automation, Beijing Institute of Technology, Beijing 100081, China; Yi Mei, Ya-hui Jia, and Will N. Browne are with School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand.

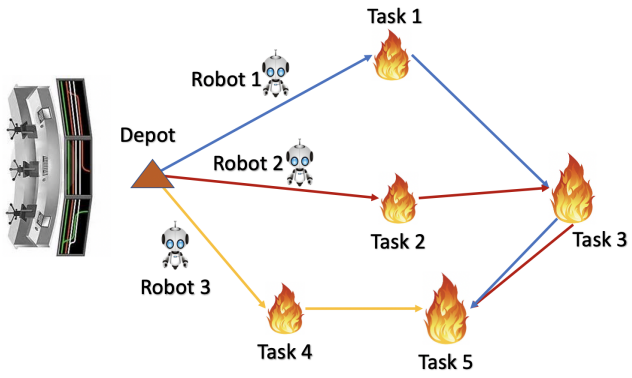


Fig. 1. An example of the task allocation problem with time-varying demands and complex dependencies.

Consequently, the time taken for a robot to eliminate a fire point strongly depends on the actions of other robots. Similar characteristics also exist in the post-disaster relief and medical resource scheduling domains.

In the MPDA problem, a number of tasks (e.g. fire points) are distributed at different locations. Each task has a time-varying demand, which grows over time. There is a set of robots located at a central depot that can be sent to execute these tasks (e.g. eliminate the fire points). The goal is to design the execution plan (i.e. sequence of tasks) of the robots to complete all the tasks so that the maximal completion time of the tasks is minimised and each task is visited by each robot at most once.

Compared with the classic routing and scheduling problems, the MPDA problem has extra challenges. First, due to the collaborations among robots, a robot needs to consider the plans of other robots during its planning process. Second, the execution time of a robot for a task is affected not only by the inherent increment rate of the task, but also by the abilities of robots executing the same task simultaneously. It is challenging to design the routes to minimise both the travel time and execution time (more robots executing the same task simultaneously). Last but not least, the time-varying demand and collaboration between robots make it non-trivial to design an effective solution encoding/decoding scheme for optimisation algorithms.

The MPDA problem is similar to some traditional optimisation problems such as Vehicle Routing Problems (VRPs) [8] and scheduling problems [9], [10]. However, it has important differences from VRPs and scheduling problems. Firstly, VRPs typically do not allow multiple vehicles to serve a customer

collaboratively, and the scheduling problems do not allow multiple machines to process a job/operation collaboratively. In contrast, the tasks are allowed to be executed by multiple robots collaboratively in the MPDA problem. Secondly, the execution (processing) time of a task is usually fixed in VRPs and scheduling problems [11]. However, it is time-dependent (i.e. grows over time) in the MPDA problem. These differences make the existing algorithms for VRPs and scheduling problems ineffective or even inapplicable for the MPDA problem.

The MPDA problem is an NP-hard problem [12]. So far, several meta-heuristic methods have been designed to solve the MPDA problem [12]–[15]. Most of these methods use permutation-based encoding schemes with decoding methods to derive the departure and arrival time of the robots at each task from the permutations. However, the existing decoding methods are not effective enough, leading to various drawbacks of the existing algorithms such as unstable convergence speed and limited applicability of a certain type of instances.

Ant Colony Optimisation (ACO), which is an effective nature-inspired optimisation method in evolutionary computation, has been successfully applied to address many routing problems [16]–[18] and scheduling problems [19]–[21]. ACO algorithms generate solutions by a probabilistic constructive mechanism [19]. The solution construction process uses ants to forage solutions according to the pheromone value, which is updated by the solutions generated during the search process. Due to the similarities between the MPDA problem and other routing and scheduling problems, we expect ACO to be effective in solving the MPDA problem as well. To our best knowledge, there is no previous work that has applied ACO to the MPDA problem.

When applying ACO to solving the MPDA problem, there are some design issues. First, the solution construction process in ACO inherits the limitations and inefficiency of the existing permutation decoding schemes. Second, the traditional single-pheromone matrix scheme shared by all ants cannot reflect the collaboration between robots. Third, the constraints in the MPDA problem are very tight, so that the traditional solution construction process in ACO may find it hard to generate feasible solutions. Fourth, it is challenging to balance between the exploration and exploitation in ACO for the MPDA problem.

To address the above issues, we propose a new ACO algorithm called the adaptive coordination ACO (AC-ACO) in this paper. Specifically, the main contributions of AC-ACO are as follows.

- To improve the efficiency of the solution construction process, a new coordinated parallel constructive mechanism, pheromone matrix representation, and pheromone updating mechanism are designed.
- A novel heuristic value adaptation scheme is designed and used in the solution construction process. This scheme removes obviously poor solutions so that AC-ACO can focus on promising solutions. Thus, the search space of the algorithm is narrowed down without losing effectiveness.
- It is possible that some tasks need many robots to complete them collaboratively. It is non-trivial to find feasible solutions by ACO in this case. To handle this situation, a pheromone-based repair mechanism is developed and embedded in the solution construction process to enhance the ability to construct feasible solutions.
- A problem-specific local search scheme is proposed to improve the balance between exploration and exploitation in AC-ACO.

The rest of this paper is organised as follows. Section II presents the mathematical model of the MPDA problem, and reviews the related work. Afterwards, the proposed AC-ACO is described in Section III. Section IV presents the computational experiments, which include empirical comparisons with other algorithms and the demonstration of the efficacy of the proposed algorithm. Finally, conclusions are drawn in Section V.

II. Background

A. Multi-Point Dynamic Aggregation

In this section, we first give the formulation of the MPDA problem, and then compare it with existing problems such as VRP and scheduling that look similar to the MPDA problem.

1) Problem Formulation: the MPDA problem can be defined on an undirected graph $G(\mathcal{V}, \mathcal{E})$. The set of nodes is $\mathcal{V} = \{v_0, v_1, \ldots, v_N\}$, where v_0 indicates the depot, and $\{v_1, \ldots, v_N\}$ indicates the set of tasks. Each task v_i has an inherent time-varying demand $q_i(t)$, which is changed based on the following equation

$$q_i(t) = q_i(0) + \gamma_i \times t, \tag{1}$$

where γ_i represents the inherent increment rate of v_i. Each edge in $e_{ij} = (v_i, v_j)$ represents the path from task v_i to v_j with the travel time t_{ij}.

There is a set of robots $\mathcal{R} = \{rob_1, \ldots, rob_M\}$ initially located at the depot v_0 to execute the tasks. Each $rob_k \in \mathcal{R}$ has the ability of μ_k, indicating the amount of demand which it can fulfil per time unit. To demonstrate the relationship between the task demand and abilities of robots, an example of the task demand accumulated over time is shown in Fig. 2. In this example, the inherent increment rate γ_i of the task is 3, and the abilities of the two robots are both 2. rob_1 arrives at the task at time 2, and rob_2 arrives at the task at time 4. From time 0 to 2, the demand increases at the inherent increment rate of 3, and reaches 6 at time 2. Then rob_1 arrives, and its ability reduces

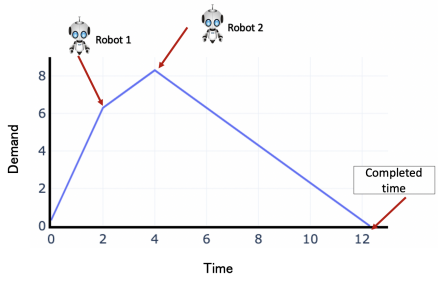


Fig. 2. An example of the demand of an executed task over time.

the increment rate from 3 to 1. From time 2 to time 4, the demand is accumulated to 8. At time 4, rob_2 arrives, making the total robot ability greater than the inherent increment rate. Since then, the demand decreases over time with a rate of 1. Finally, at time 12, the demand becomes 0, and the task is completed.

Based on the aforementioned notations, the MPDA model can be defined as follows.

$$\min \max_{i\in\{1,\cdots,N\}} ct_i \tag{2}$$

$$\text{s.t.} \sum_{j=0}^{N} x_{ij}^k = \sum_{j=0}^{N} x_{ji}^k, \forall i = 1,\ldots,N, \forall k = 1,\ldots,M \tag{3}$$

$$\sum_{i=0}^{N}\sum_{k=1}^{M} x_{ij}^k \geq 1, \forall j = 1,\ldots,N \tag{4}$$

$$\sum_{i=0}^{N} x_{ij}^k \leq 1, \forall j = 1,\ldots,N, \forall k = 1,\cdots,M \tag{5}$$

$$at_{0k} = ct_0 = 0, \forall k = 1,\cdots,M \tag{6}$$

$$at_{jk} = \sum_{i=0}^{N}(ct_i + t_{ij})x_{ij}^k, \forall j = 1,\ldots,N, \forall k = 1,\cdots,M \tag{7}$$

$$q_j(0) + \gamma_j ct_j = \sum_{i=0}^{N}\sum_{k=1}^{M} x_{ij}^k \mu_k (ct_j - at_{jk}), \forall j = 1,\ldots,N \tag{8}$$

$$\gamma_j < \sum_{k=1}^{M}\sum_{i=0}^{N} x_{ij}^k \mu_k, \forall j = 1,\ldots,N \tag{9}$$

$$x_{ij}^k \in \{0,1\}, i \neq j, \forall i,j = 1,\ldots,N, \forall k = 1,\ldots,M \tag{10}$$

where the binary decision variables x_{ij}^k takes 1 if rob_k goes from v_i to v_j, and 0 otherwise. at_{jk} represents the arrival time of rob_k at v_j, and ct_j represents the completion time of v_j. The notations used in the problem formulation are summarised in Table I.

The objective (2) is to minimise the maximum completion time of all the tasks. It should be noted that the objective in the MPDA problem does not contain the cost to return to the depot, which is different from the classical VRPs [22], [23]. Constraint (3) indicates that the number of incoming paths equals the number of outgoing paths for each robot at each task. Constraint (4) ensures that each task is executed by at least one robot. Constraint (5) ensures that each task is executed by each robot at most once. Constraint (6) sets the arrival time and completion time for the depot to 0. Constraint (7) specifies the relationship between ct_j, at_{jk} and x_{ij}^k. Constraint (8) implies that a task is completed when its demand decreases to zero (i.e. the accumulated demand from time 0 to ct_j equals the total demand reduced by the robots executing the task during this time period). It also shows the time-varying characteristic of the task demand. Constraint (9) indicates that for each task, the total ability of the robots executing it must be greater than its inherent increment rate. Otherwise, the task can never be completed. Constraint (10) sets the binary domain of the decision variables.

TABLE I
THE NOTATIONS USED IN THE PROBLEM FORMULATION

Notation	Description
N	the number of tasks
M	the number of robots
v_j	the task with index j
$q_j(t)$	the demand of v_j accumulated at time t
γ_j	the inherent increment rate of v_j
t_{ij}	travel time from v_i to v_j
rob_k	the robot with index k
μ_k	the ability of rob_k
at_{jk}	the arrival time of rob_k at v_j
ct_j	the completion time of v_j
x_{ij}^k	1 if rob_k goes from v_i to v_j, and 0 otherwise

TABLE II
COMPARISON BETWEEN THE MPDA PROBLEM AND OTHER SIMILAR COMBINATORIAL OPTIMISATION PROBLEMS.

Problem	Time-varying demand	Efficiency affected by robot ability	Task executed by multiple robots
MPDA	✓	✓	✓
VRP [22]			
CARP [23]			
DCARP [24]	✓		
MCVRP [25]		✓	
SDVRP [26]			✓
PMS [27]		✓	
FJSS [28]		✓	
MRTA [29]			✓

As the problem model (particularly constraints (7) and (8)) shows, the MPDA problem contains complex dependencies among robots and tasks. The completion time of a task depends on the arrival time of all the robots executing the task and the completion time of multiple previous tasks. In other words, the plans of the robots interact with each other. If the routes of different robots are more synchronised, i.e. they arrive at the same tasks at the same time, then they can complete their tasks more efficiently. Therefore, a good solution to the MPDA problem should have the following properties: (1) the travel time between adjacent tasks is short; (2) different robots have similar arrival time at the same task.

2) Comparison with Other Problems: Table II compares the MPDA problem with several representative classical combinatorial optimisation problems that are similar to the MPDA problem, including VRP [22], capacitated arc routing problem (CARP) [23], dynamic capacitated arc routing problem (DCARP) [24], multi-depot capacitated vehicle routing problem (MCVRP) [25], split delivery vehicle routing problem (SDVRP) [26], parallel machine scheduling problem (PMS) [27], flexible job-shop scheduling problem (FJSS) [28], and multi-robot task allocation problem (MRTA) [29]. Without loss of generality, vehicles in VRPs and machines in the scheduling problems can be regarded as robots in the MPDA problem, while customers in VRPs and jobs in the scheduling problems can be regarded as tasks in the MPDA problem.

Compared with other well-known similar problems, the MPDA problem has three unique characteristics. First, the demands of the tasks are time-varying, i.e. they grow over time. Second, the execution efficiency of a task is affected by the abilities of the robots executing it. Third, the tasks can be executed by multiple robots collaboratively, and the execution

time depends on the arrival time of the robots at the task. From Table II, none of the compared problems contains all the three characteristics.

B. Related Work

1) Multi-point Dynamic Aggregation problem: The MPDA problem is a multi-robot task allocation problem with complex dependencies [29]. There have been some studies about the task allocation problem with complex dependencies. A genetic algorithm designed for complex task allocation problems of self-driving fire trucks is proposed in [30]. The experimental results [30] show that the designed algorithm provides better overall results to the task allocation problem with faster convergence. A flexible chromosome structure, customised mutation operators, and a penalty function were also designed to deal with the coordinated characteristic of a task in [31]. the MPDA problem and the aforementioned task allocation problem both have the characteristic where the efficiency of a robot executing a task depends on the plans of other robots. However, the aforementioned approaches are based on the assumption that the demand of a task is constant. Thus, they can not be directly applied to the MPDA problem since the demand of a task in the MPDA problem is time-varying, leading to more complex decisions (e.g. a task with a larger inherent increment rate should be executed earlier).

The MPDA problem was proposed by Xin *et al.* [1] recently. So far, only a few heuristic approaches have been proposed for it. First, market-based approaches were designed to address the MPDA problem. Although these market-based algorithms can obtain a plan in a short time, the plan is not of a high quality due to the NP-hardness characteristic of the MPDA problem. Hao *et al.* [15] combined the differential evolution and estimation of distribution algorithm (EDA) to address the MPDA problem. The designed hybrid algorithm outperforms two versions of differential evolution in terms of the convergence speed and solution quality. Xin *et al.* [12] designed an EDA using K-means clustering and multi-modal Gaussian distribution, in which a multi-permutation encoding/decoding method was adopted. Experimental results have shown that the proposed EDA finds high-quality solutions and outperforms the genetic algorithm and random search method. To promote the exploitation ability of algorithms, we have designed two memetic algorithms with different individual learning strategies in our previous work [14]. The first one is an equality one-step local search (OLS) approach with better exploration ability. In contrast, the second one is an elite multi-step local search (MLS) approach with improved exploitation ability.

However, these existing methods do not consider the collaborative relationships among robots and the chronological order. Their decoding and modelling mechanisms do not match the parallel processes of robots executing tasks and are time-consuming. Besides, these methods do not use the intermediate domain knowledge during the solution construction process, and thus the solution construction process is inefficient. Finally, these methods proposed for the MPDA problem do not have any mechanism to deal with the tight constraints, e.g., the instances in which tasks have large inherent increment rates. As a result, these methods cannot find feasible solutions in some extreme cases. To overcome these drawbacks, a coordinated solution construction process, an adaptive heuristic, and a repair mechanism are designed in AC-ACO.

2) Ant Colony Optimisation: Artificial ants in ACO algorithms forage solutions collaboratively by learning messages from pheromones on their search paths. ACO was first proposed by Dorigo *et al.* to address the travelling salesman problem [16]. Later, researchers proposed several ACO variants such as ant colony system [17] and max-min ant system [32]. In general, there are two main procedures in ACO algorithms [33]–[35]: *a) Solution construction*: candidate solutions are constructed according to a probability distribution based on the pheromone values (and heuristic information). *b) Pheromone updating*: the pheromone values are updated during the execution of the algorithm using the fitness values of the generated solutions as feedback.

So far, ACO has been demonstrated to be effective on several combinatorial optimisation problems, such as VRPs, scheduling problems, and task allocation problems [21], [36]–[40]. Zhang *et al.* [41] developed an ACO algorithm hybridised with mutation operators which exploits the neighbourhood space to solve VRPs. Wang *et al.* [42] proposed an ACO algorithm combined with the 2-opt and tabu search to address the routing problem in post-disaster scenarios. Jia *et al.* [21] proposed an adaptive ACO algorithm to meet the quality of service criterion and orchestrate tasks in a cloud workflow scheduling system. Pendharkar [43] presented an ACO algorithm to address the constraint task allocation problem. Huang *et al.* [44] proposed an ACO algorithm with a novel solution construction strategy and an incremental flow assignment method to address the evacuation path optimisation problem.

However, these existing ACO algorithms cannot be applied to the MPDA problem directly due to the time-varying and multi-robot collaboration characteristics. To solve MPDA with ACO, the solution construction process, pheromone model, and pheromone updating scheme need to be carefully redesigned based on the characteristics of the MPDA problem.

III. Adaptive Coordination Ant Colony Optimisation

The flowchart of the proposed AC-ACO is shown in Fig. 3. AC-ACO contains five main components, 1) pheromone initialisation, 2) solution construction, 3) evaluation, 4) local search, and 5) pheromone updating. In the algorithm, M pheromone matrices are maintained for solution construction, each corresponding to a robot. At the beginning of AC-ACO, the M pheromone matrices are initialised based on four greedy methods in the initialisation process. Then, the other four components iterate until the stopping criterion is satisfied. In each iteration of AC-ACO, nt solutions are first generated based on the pheromone matrices and a repair mechanism, each using a team of M ants. Then a local search algorithm is applied to the best-so-far solution sol^* to further refine the solution quality. Finally, the pheromone matrices are updated according to the fitness values of the generated solutions and the best-so-far solution.

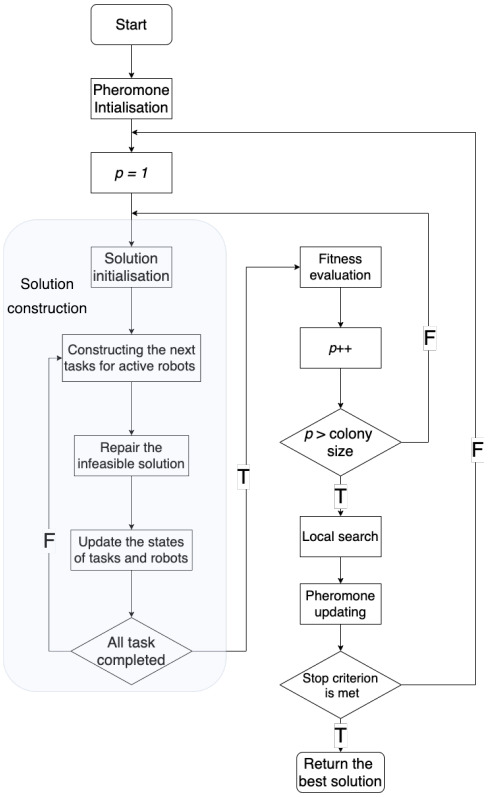


Fig. 3. Flowchart of the proposed AC-ACO.

A. Solution and Pheromone Representation

Due to the complexity of the MPDA problem, the solution and pheromone representations are important for the search process [45]. We will describe them as follows.

1) Solution representation: In AC-ACO, each solution is represented as a variable-length sequence of events, denoted as

$$sol = [event_1, event_2, \ldots, event_p]^T \tag{11}$$

Each event contains four elements, including the robot rob_e, the task v_e, the time t_e, and the event type C_e, i.e.

$$event_e = (rob_e, v_e, t_e, C_e). \tag{12}$$

C_e is a binary variable distinguishing two kinds of events. $C_e = 1$ represents an arrival event, i.e. the robot rob_e arrives at the task v_e and starts to execute it at time t_e. $C_e = 0$ represents a departure event, i.e. the robot rob_e departs from the task v_e at time t_e.

2) Pheromone Representation: In the MPDA problem, different robots should have different paths to collaborate with each other effectively. Therefore, we maintain M pheromone matrices in the algorithm for each robot. The pheromone matrix PM_k for rob_k is a $(N+1) \times (N+1)$ matrix, denoted as

$$PM_k = \begin{bmatrix} \tau^k_{0,0} & \cdots & \tau^k_{0,N} \\ \vdots & \ddots & \cdots \\ \tau^k_{N,0} & \ddots & \tau^k_{N,N} \end{bmatrix}. \tag{13}$$

where $\tau^k_{i,j}$ represents the pheromone value of robot rob_k from v_i to v_j in PM_k. Note that index 0 indicates the depot. For example, $\tau^k_{0,1}$ indicates the pheromone value of rob_k from the depot to v_1.

B. Pheromone Initialisation

In ACO algorithms, the initial pheromone values are usually set based on greedy methods [17], [19]. Specifically, one or more solutions with reasonably good quality are generated by a greedy method. Then, the initial pheromone values are set based on the qualities of these solutions.

In the MPDA problem, multiple factors such as the ability of robot, inherent increment rate of the tasks, and the travelling distance between tasks contribute to the complexity of the problem. Thus, a single greedy method is not able to capture all the important information to generate good solutions for all instances. To address the issue, we designed four different greedy methods, considering different problem aspects including the travel time, workload balance, and task demand. These four greedy methods are described as follows.

- Minimal travel Time (MT) method: The motivation of the MT method is to reduce the travel time from one task to another (including the depot) in the path. The idea is similar to the nearest neighbour method in TSP [17]. Specifically, all the robots are initially at the depot and ready to depart, i.e. *active*. As soon as a robot becomes active, the remaining task closest to the robot's current location is selected to be executed next.
- Average Abilities (AA) method: The motivation of the AA method is to balance the workload between robots, which is similar to the greedy method of the set partitioning problem [46]. In the AA method, whenever a robot becomes active (ties are broken randomly), the remaining task with the maximal actual growth rate (i.e. the inherent increment rate minus the total ability of the robots that are executing it) is selected to be executed next.
- MAXimal (MINimal) inherent increment Rate (MAXR/MINR) methods: The motivation of the MAXR and MINR methods are inspired by the greedy method for the scheduling problem [47]. Specifically, all the robots follow the same path that executes the tasks from the maximal (minimal) inherent increment rate to the minimal (maximal) inherent increment rate.

The initial pheromone value $\tau^k_{i,j}(0)$ is calculated as follows:

$$\tau^k_{i,j}(0) = \frac{1}{\min\{f_{\text{MT}}, f_{\text{AA}}, f_{\text{MAXR}}, f_{\text{MINR}}\}}, \tag{14}$$

where f_{MT}, f_{AA}, f_{MAXR}, and f_{MINR} are the objective values of the solutions generated by the four greedy methods, respectively.

C. Solution Construction

The solution construction method proposed in this paper is shown in Algorithm 1. It can be divided into four parts: solution initialisation (lines 1-2), construction (lines 4-13), repairing (lines 14-30), and state updating (line 31).

1) Solution Initialisation: Initially, all the robots are at the depot and are active. The solution is initialised as an empty sequence.

Algorithm 1 Solution Construction

INPUT: The instance, pheromone matrices $PM_1, \ldots, PM_M$.
OUTPUT: The constructed solution sol.

Set $sol \leftarrow ()$, all the tasks not completed, all the robots at the depot;
Set active robots $\mathbf{AR} \leftarrow \mathcal{R}$;
while not all the tasks are completed **do**
  Shuffle the active robots $\mathbf{AR}$ randomly;
  for $rob_k \in \mathbf{AR}$ **do**
    Get the current location pos_k of rob_k;
    Get the unvisited tasks $\mathbf{UT}_k$ of rob_k;
    for each task $v_j \in \mathbf{UT}_k$ **do**
      Calculate the heuristic value $\eta^k_{pos_k,j}$;
    end for
    Calculate the probabilities for each $v_j \in \mathbf{UT}_k$ by Eq. (15);
    Select the next task v^*_k based on the probabilities;
  end for
  // Repair
  Calculate the actual growth rate $\lambda_j = \gamma_j - \sum_{i=0}^{N}\sum_{k=1}^{M}\mu_k x^k_{ij}$ $(j = 1, \ldots, N)$;
  while $\min_{j=1}^{N}\{\lambda_j\} \geq 0$ **do**
    Find the task $v^* = \arg\min_{j=1}^{N}\{\lambda_j\}$;
    Find the active robots not allocated to v^*, denoted by $\overline{\mathbf{AR}}^*$;
    if $\overline{\mathbf{AR}}^* = \emptyset$ **then**
      break;
    end if
    Select a random index k_{rand} from $\overline{\mathbf{AR}}^*$;
    for $rob_k \in \overline{\mathbf{AR}}^*$ **do**
      Calculate P_k according to Eq. (18);
      if $k = k_{rand}$ or rand() $< P_k$ **then**
        Change the next task v^*_k to v^*;
      end if
    end for
    Update λ_j $(j = 1, \ldots, N)$;
  end while
  Set $\Pi = \{[rob_k, v^*_k] \mid rob_k \in \mathbf{AR}\}$;
  $[\mathbf{AR}, sol] =$ `StateUpdate`(Π, sol) (See Algorithm 3);
end while
return sol;

2) Constructing: During the solution construction process, at each step, we determine the next task for an active robot based on the probabilities shown in Eq.(15). In case there are multiple active robots, we randomly shuffle them to avoid biased planning. At iteration g, the probability P^k_{ij} of rob_k moving from v_i to v_j is calculated as follows.

$$P^k_{ij}(g) = \begin{cases} \frac{[\tau^k_{ij}(g)]^\alpha [\eta^k_{ij}(g)]^\beta}{\sum_{s \in \mathbf{UT}_k}[\tau^k_{is}(g)]^\alpha [\eta^k_{is}(g)]^\beta}, & if\ v_j \in \mathbf{UT}_k, \\ 0 & otherwise \end{cases} \tag{15}$$

where $\mathbf{UT}_k$ represents the set of tasks which have not been visited by rob_k. α and β are the parameters controlling the relative importance of pheromone versus heuristic in Eq. (17). To consider the time-varying task demand and the collaborations between the robots effectively, we design a problem-specific heuristic information η^k_{ij} in Eq. (15) that contains both static and dynamic components.

The static heuristic component $\eta^{k,s}_{ij}$ depends on the travel time t_{ij} from v_i to v_j, which is static information. It is calculated as follows.

$$\eta^{k,s}_{ij} = \frac{1}{t_{ij}}. \tag{16}$$

The dynamic heuristic component $\eta^{k,d}_{ij}$, on the other hand, depends on the information that changes over time during the process of solution construction. Its calculation process is shown in Algorithm 2. There are three situations considered when the dynamic heuristic component is calculated.

Algorithm 2 Calculate the dynamic heuristic component

INPUT: Instance, current partial solution sol, robot rob_k, parameter ω.
OUTPUT: The dynamic heuristic component $\eta^{k,d}_{ij}$.

1: **if** $ct(v_j) < ct(v_i) + t_{ij}$ **then**
2: **return** 0;
3: **end if**
4: Get the number of robots NR assigned to v_j;
5: Calculate x^s_{ij} from sol;
6: **if** $\sum_{i=0}^{N}\sum_{s=1}^{M}\mu_s x^s_{ij} \leq \gamma_j$ **then**
7: $\eta^{k,d}_{ij} = NR + 1$;
8: **else**
9: **if** $\sum_{i=1}^{N}\gamma_i \geq \omega \times \sum_{s=1}^{M}\mu_s$ **then**
10: $\eta^{k,d}_{ij} = NR + 1$;
11: **else**
12: $\eta^{k,d}_{ij} = 1/(NR+1)$;
13: **end if**
14: **end if**
15: **return** $\eta^{k,d}_{ij}$;

- First, some potential poor decisions are disregarded based on domain knowledge. Specifically, if the predicted arrival time of rob_k from v_i to v_j is later than the predicted completion time of v_j, i.e. $ct_i + t_{ij} > ct_j$, then there is no need for rob_k to go to v_j anymore. In this case, the dynamic heuristic component is set to 0 to prevent rob_k from going to v_j (lines 1-3).
- Second, if v_j cannot be completed by the current assignment, i.e. $\sum_{i=0}^{N}\sum_{s=1}^{M}\mu_s x^s_{ij} \leq \gamma_j$, more robots are needed to execute v_j. In this case, the dynamic heuristic component for assigning rob_k to v_j should increase to encourage more robots to go to v_j. To this end, we set the dynamic heuristic component to $NR+1$, where NR is the number of robots already assigned to v_j (lines 6-8).
- Third, if v_j can be completed in finite time, then a cluster dynamic heuristic information (lines 9-14) is designed. All the tasks are aggregated into a virtual task with an inherent increment rate $\gamma = \sum_{i=1}^{N}\gamma_i$ and all the robots are aggregated into a virtual robot with an ability $\mu = \sum_{s=1}^{M}\mu_s$. We define a measure $\phi = \gamma/\mu$ to represent the efficiency of the virtual robot to execute the virtual task. If ϕ is large, then the task demand grows relatively fast compared with the robot abilities, and the robots should collaborate more to complete the tasks (with a large $\eta^{k,d}_{ij}$ value). Otherwise, the task demand grows relatively slowly, and the robots have more freedom to execute different tasks respectively (with a small $\eta^{k,d}_{ij}$ value). Here we use a parameter ω to make the decision. If $\phi \geq \omega$ (i.e. $\gamma \geq \omega \times \mu$), we set a larger dynamic heuristic component $\eta^{k,d}_{ij} = NR+1$. Otherwise, we set a smaller value $\eta^{k,d}_{ij} = 1/(NR+1)$ to distribute the robots to different tasks.

Combining the two aforementioned components, the heuristic value of rob_k moving from v_i to v_j is given by

$$\eta^k_{ij} = \eta^{k,s}_{ij} \times \eta^{k,d}_{ij}. \tag{17}$$

3) Repair: During the solution construction process, there may be cases where none of the robots can complete their

allocated tasks, leading to an infeasible solution. For example, given an MPDA instance with two robots and two tasks, where the inherent increment rate of each task is larger than the individual ability of both robots. Thus, the robots have to go to the same task at the same time to complete it. If the plan allocates each robot to execute a task separately, neither of the tasks can be completed by the single allocated robot. To address this issue, we develop a repair mechanism to re-allocate the robots to make the solution feasible.

In the proposed repair mechanism, we first check whether an infeasible solution is generated by calculating the actual growth rate λ_j of each task v_j based on the current allocation, i.e. $\lambda_j = \gamma_j - \sum_{i=0}^{N}\sum_{k=1}^{M}\mu_k x_{ij}^k$, $j = 1, \ldots, N$. If all the tasks have non-negative actual growth rate, i.e. $\min_{j=1}^{N}\{\lambda_j\} \geq 0$, then the repair operator is conducted to re-allocate the robots. We focus on completing the task with the smallest actual growth rate, denoted as v^*, since it can be completed most easily. For this purpose, we first randomly select a robot that is not allocated to v^*, and change its next destination to v^*. If this is still not sufficient, we shift more robots to v^* based on their pheromone information. Specifically, the probability of rob_k changing its current target to v^* is

$$P_k = \frac{\tau_{pos_k,j}^k}{\sum_{q\in\overline{\mathbf{AR}}^*}\tau_{pos_q,j}^q} \tag{18}$$

where $\overline{\mathbf{AR}}^*$ represents the active robots which are not allocated to v^*. The repair process is continued until the task can be completed, i.e. $\min_{j=1}^{N}\{\lambda_j\} < 0$.

4) State Updating: After deciding the next tasks Π of the active robots, the states of the robots and tasks are updated by Algorithm 3. The state of a task v_j includes its current demand $q_j(t)$, actual growth rate of the demand λ_j, and predicted completion time ct_j. The state of a robot rob_k includes its visiting sequence, current task v_{pos_k}, and predicted event $event_k^*$. The demonstrated updating algorithm adopts an event-triggered mechanism, where the concept of $event$ was defined in Eq. (12). As a result, the partial solution sol is updated (new events are added in chronological order), and the new set of active robots are obtained for the next round of the *constructing* process.

The predicted event $event_k^*$ of a robot implies what the robot is going to do in its next stage. If the predicted event is an arrival event, the robot is on the way to execute the assigned tasks. If the predicted event of a robot is a departure event, the robot is executing a task. Note that the predicted events are different from the events in sol, as they have not been added into sol yet. The predicted events can be disregarded, and their event time information can be modified by other events before being added into sol.

This updating algorithm contains two processes: arrival input (lines 1-3) and execution loop (lines 4-37). In the arrival input process (lines 1-3), the predicted event of each active robot is set according to the next task in Π, the partial solution sol, and the travel time. Note that the predicted events of the active robots are all arrival events, i.e. $C_e = 1$.

During the execution loop process (lines 4-37), the state of robots and tasks are further updated. First, to make sure the events in sol are generated in chronological order, the earliest predicted event (lines 5-8) is found by `EarliestPredictedEvent()`. If the time of the earliest predicted event is ∞, which implies that none of the robots can complete their allocated tasks, no active robot is returned, and the entire solution construction process is terminated.

If the earliest event is an arrival event (i.e. $C_e = 1$), it is added into the current partial plan sol. Then, the predicted event of the corresponding robot rob_e is updated as the departure event ($C_e = 0$) of rob_e from v_e. To calculate the departure time, three conditions are considered during the event updating (lines 9-26).

- If v_e has been completed, rob_e will depart from it immediately (lines 11-13). The predicted event of rob_e is set as $[rob_e, v_e, t_e, 0]$.
- If v_e can be completed in a finite time according to the current partial solution, i.e. $\sum_{i=0}^{N}\sum_{k=1}^{M}\mu_k x_{ie}^k > \gamma_e$, the predicted completion time ct_e can be calculated/updated by Eq. (8). Then, the predicted events of all the robots that are executing v_e need to be updated in terms of departure time (lines 14-21).
- If v_e cannot be completed in a finite time according to the current partial solution, the departure time of the predicted event is set to ∞ (lines 22-24).

Otherwise, the earliest predicted event is a departure event (i.e. $C_e = 0$), indicating that v_e is completed at time t_e. The predicted departure events of all the robots executing v_e are added into sol. Then, all the robots executing the completed v_e are selected as the new active robots, and their next tasks will be determined by the next iteration of the *constructing* process (lines 27-36).

To better understand the solution construction process, an example is shown in Fig. 4 that corresponds to the scenario of Fig. 1, including three robots and five tasks. At the beginning, all the robots are activated and assigned to three different tasks ($rob_1 \rightarrow v_1$, $rob_2 \rightarrow v_2$, and $rob_3 \rightarrow v_4$). Judged by the function `EarliestPredictedEvent()`, the arrival event of rob_3 arriving at v_4 is the earliest event, and it is added into sol as the first event. Then, the event of rob_2 arriving at v_2 and the event of rob_1 arriving at v_1 are added into sol in chronological order. When the robots start working, the departure event of each robot is estimated and added into sol. The example shows that the departure event of rob_1 departing from v_1 is the fourth event in sol. When the former three events finish and this departure event becomes the first one, rob_1 becomes active again, and selects v_3 as its next task. rob_2 also selects v_3 as its next task. Thus, the two robots would collaborate at v_3 after finishing v_1 and v_2. This process continues and the following events are appended into sol sequentially. Finally, we can see that there is only v_5 left, and all the three robots work together to execute this task. After finishing v_5, the whole solution sol is obtained.

D. Evaluation

For any feasible solution, the fitness function value is set directly to the maximal completion time of all the tasks. If all the tasks are completed in a finite time, the fitness value can

Algorithm 3 The state updating procedure

INPUT: the next tasks Π, current partial solution sol.
OUTPUT: the active robots $\mathbf{AR}$, updated solution sol.

for $[rob_k, v_j]$ in Π **do**
  $event_k^*$ = $(rob_k, v_j, ct(v_{pos_k}) + t_{pos_k,j}, 1)$;
end for
while all tasks have not been completed **do**
  (rob_e, v_e, t_e, C_e) = `EarliestPredictedEvent()`;
  if $t_e = \infty$ **then**
    return $\emptyset$, sol;
  end if
  if $C_e == 1$ **then**
    Add (rob_e, v_e, t_e, C_e) to sol;
    if v_e has been completed **then**
      Set the predicted event of rob_e as $(rob_e, v_e, t_e, 0)$;
    else
      Calculate x_{ie}^k from sol;
      if $\sum_{i=0}^{N}\sum_{k=1}^{M} \mu_k x_{ie}^k > \gamma_e$ **then**
        Calculate ct_e according to Eq. (8);
        for all $k \in \{1, \cdots, M\}$ **do**
          if $\sum_{i=0}^{N} x_{i,e}^k == 1$ **then**
            $event_k^* = (rob_k, v_e, ct_e, 0)$;
          end if
        end for
      else
        Set the predicted event of rob_e as $(rob_e, v_e, \infty, 0)$;
      end if
    end if
  else
    Let the set of active robots $\mathbf{AR} = \emptyset$;
    Calculate x_{ie}^k from sol;
    for all $k \in \{1, \cdots, M\}$ **do**
      if $\sum_{i=0}^{N} x_{ie}^k == 1$ **then**
        $\mathbf{AR} = \mathbf{AR} \cup rob_k$
        Add $(rob_k, v_e, t_e, 0)$ to sol;
      end if
    end for
    return $\mathbf{AR}$, sol;
  end if
end while
return $\emptyset$, sol;

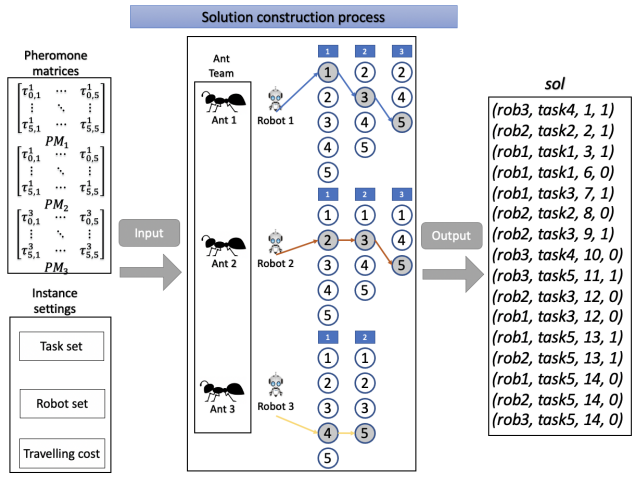


Fig. 4. The solution construction process for the instance in Fig. 1.

be obtained easily by the time of the last event in sol. If the solution is infeasible and some tasks can never be completed, then a sufficiently large fitness value is set to bias the search to the feasible solutions.

Algorithm 4 Local search of sol^*

INPUT: The best-so-far solution sol^*, number of neighbours to explore nsb, and maximal number of swapped permutations nsp ($nsp \leq M$).
OUTPUT: The updated best-so-far solution sol^*.

Simplify sol^* to the visiting sequences $\mathbf{VS}$;
Complement the original sequences $\mathbf{VS}$ to $\mathbf{VS}'$;
for $l = 1 \rightarrow nsb$ **do**
  Sample ns uniformly from $\{1, 2, \cdots, nsp\}$;
  Randomly select ns permutations from $\mathbf{VS}'$;
  for each selected permutation **do**
    Randomly select two elements and swap them;
  end for
  Decode the modified $\mathbf{VS}'$ to sol';
  if sol' is better than sol^* **then**
    $sol^* = sol'$;
  end if
end for
return sol^*;

E. Local Search

A local search procedure is designed to further enhance the exploitation ability of AC-ACO. The designed local search first transforms the sequence of events into the task sequences of the robots. Then, it modifies the event sequences to generate potentially better neighbouring solutions.

The proposed local search procedure is conducted on the best-so-far solution after each iteration of AC-ACO. Algorithm 4 shows its process. Given the original solution (i.e. the current best-so-far solution sol^*), it first transforms the sequence of events into task sequences $\mathbf{VS}$ by ignoring the event time and type. Then, it fills the task sequences to complete permutations of all the tasks, denoted as $\mathbf{VS}'$. This is because in a solution, a robot may not be assigned to all the tasks. To fill its task sequence, the remaining tasks that are not executed by the robot are appended to the end of the sequence one by one according to their index. After that, each complete task sequence $VS \in \mathbf{VS}'$ undergoes a traditional local search process with the swap operator (i.e. swapping the location of two elements in the sequence) [48]. Specifically, to generate a neighbour sol', first the number of task permutations ns is randomly sampled between 1 and nsp ($nsp \leq M$). Then, ns task permutations are randomly selected from $\mathbf{VS}'$. For each selected task sequence, two tasks are randomly selected and swapped. Finally, the modified task permutations $\mathbf{VS}'$ is decoded into sol' according to Algorithm 3. The maximum number of nsb neighbours are generated. Then, the best neighbour is compared with sol^*, and the better one is kept.

An example is shown in Fig. 5. It should be noticed that at least one of the swapped elements must come from the original permutation to ensure that the original solution is modified by the swap operator.

F. Pheromone updating

After the solution construction and local search, the pheromones are updated. The pheromone updating in AC-ACO contains two complementary steps: pheromone evaporation and pheromone deposition, which is shown as

$$\tau_{ij}^k = (1-\rho)\tau_{ij}^k + \sum_{p=1}^{nt} \triangle\tau_{ij}^k(sol_p) + \triangle\tau_{ij}^k(sol^*) \qquad (19)$$

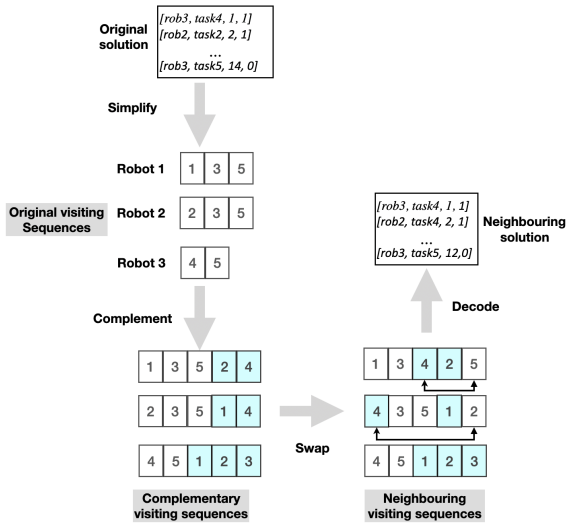


Fig. 5. An example to describe the neighbouring solutions. White blocks mean original elements, and shadow blocks mean complementary elements.

$$\triangle \tau_{ij}^{k}(sol) = \begin{cases} \frac{1}{NM \cdot f(sol)}, & \text{if } rob_k \text{ goes from } v_i \text{ to } v_j \text{ in } sol, \\ 0, & \text{otherwise,} \end{cases} \tag{20}$$

where ρ is the evaporation rate and nt represents the ant colony size (i.e. number of solutions constructed per iteration). It should be noticed that all the solutions constructed in the previous generation and the best-so-far solution sol^* are considered to deposit pheromones.

In summary, the above four newly developed mechanisms help AC-ACO address the MPDA problem effectively. First, the designed solution construction process allows the ants to generate a solution cooperatively to fit the characteristics of robots executing tasks in parallel in the MPDA problem. Second, the adaptive heuristic information is designed according to the domain knowledge of the MPDA problem in the solution construction process. Then, a repair mechanism is designed to repair the infeasible solutions. Finally, a local search for the elite solution is designed to enhance exploitation.

IV. Experimental Studies

A. Benchmark

Previous studies only tested their algorithms on a few small-scale instances with less than 25 tasks. The scale and number of these existing instances are too small to show the ability of the algorithms [12], [14], [15]. To further investigate the effectiveness and efficiency of AC-ACO thoroughly, in this paper, we designed a new set of benchmark instances that have large range and more comprehensive considerations about the problem characteristics following the approach that generates benchmarks for the capacitated VRP [49]. The full details of the designed 45 benchmark instances are shown in Section SI in the supplemental material. The benchmark instances designed by previous studies [14] are also adopted. All the instances are named by their scale, number of robots, number of tasks, and the ratio between the sum of all the task inherent increment rates and the sum of all the robot abilities.

TABLE III
Parameter settings of AC-ACO.

Parameter	Value
ρ	0.05
α	1
β	1
Ant colony size	$N \times M$
Stop criterion	$M \cdot N \cdot \max(M, N)/10$ seconds
ω	2
nsp	3
nsb	$40N$

B. Experimental Settings

To show the competitiveness of AC-ACO, two other representative meta-heuristic algorithms for the MPDA problem, i.e. the MAs [14] and EDA [12], are compared. Experimental results have shown that the EDA [12] outperforms the genetic algorithm and random search method. The MAs [14] also obtained good performance. It has two variants: the MA with an equality one-step local search (OLS) and the MA with an elite multi-step local search (MLS). To the best of our knowledge, there was no other approaches directly for the MPDA problem. Thus, due to the similarities of the MPDA problem with MDCVRP, a state-of-the-art iterative local search (ILS) method for MDCVRP [25] is also compared as a baseline method. To adapt ILS to the MPDA problem, the encoding and decoding methods proposed in Section III are used in the searching process of ILS. Regarding the parameter setting of AC-ACO, following the conventional settings [16] and the parameter sensitivity analysis (details shown in Section SII and SIII in the supplemental material), the detailed parameter settings are shown in Table

To keep fair comparisons, all the compared algorithms are implemented in Python. The complexity of the solution construction process in AC-ACO and the decoding process in other algorithms depends on the number of tasks and robots. Therefore, the stopping criterion of all algorithms is set to a maximum computational time of $M \cdot N \cdot \max(M, N)/10$ seconds. Each algorithm is executed 30 independent times on each instance.

C. Comparison with Other Methods

The overall results of the compared five methods are shown in Table IV, including the mean and standard deviation of the objective values over all the independent runs, and the results of Wilcoxon rank-sum test with a significance level of 0.05. For each compared algorithm, "$(+)$", "$(\approx)$", and "$(-)$" indicates that the compared algorithm performed significantly better, statistically comparable, and significantly worse than AC-ACO, respectively. An additional row at the bottom of Table IV represents the number of instances on which the corresponding compared algorithm achieves significantly better, statistically comparable, and significantly worse results than AC-ACO.

1) Constraint handling ability: AC-ACO has found feasible solutions for all the instances. MA-MLS, MA-OLS and ILS obtained feasible solutions on 48 instances except S_10_20_6.04 and S_15_20_5.98. EDA obtained feasible solutions on 37 tested instances. The reason that AC-ACO can

TABLE IV
COMPUTATIONAL RESULTS BETWEEN AC-ACO AND OTHER METHODS ON THE BENCHMARK INSTANCES.

Instance	MA-MLS		MA-OLS		EDA		ILS		AC-ACO	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
S_5_4_0.39	1.07E+2 (-)	1.9E+0	1.06E+2 (≈)	1.5E+0	1.05E+2 (+)	1.2E-1	1.06E+2 (≈)	1.3E+0	1.06E+2	1.3E+0
S_5_5_1.66	1.22E+3 (≈)	4.6E+1	1.17E+3 (+)	3.2E+1	1.31E+3 (-)	5.9E+1	1.20E+3 (+)	5.7E+1	1.23E+3	2.9E+1
S_3_10_1.51	9.52E+2 (-)	2.1E+1	9.22E+2 (-)	3.6E+1	1.03E+3 (-)	2.9E+1	8.99E+2 (-)	3.2E+1	8.77E+2	1.5E+1
S_3_15_5.03	1.09E+5 (-)	4.4E+4	5.98E+4 (-)	1.3E+4	*	*	4.00E+4 (-)	7.4E+3	2.73E+4	10.0E+1
S_5_10_0.93	4.33E+2 (-)	1.1E+1	4.20E+2 (-)	1.2E+1	4.78E+2 (-)	9.4E+0	4.15E+2 (-)	9.4E+0	4.09E+2	8.2E+0
S_10_5_1.39	6.00E+2 (-)	2.2E+1	5.87E+2 (-)	2.6E+1	6.53E+2 (-)	2.7E+1	6.01E+2 (-)	3.6E+1	5.68E+2	1.7E+1
S_5_10_3.67	3.12E+4 (-)	6.3E+3	2.23E+4 (-)	2.9E+3	*	*	1.83E+4 (-)	1.8E+3	1.31E+4	7.1E+2
S_5_20_4.36	5.24E+4 (-)	7.2E+3	3.74E+4 (-)	4.3E+3	*	*	3.35E+4 (-)	4.7E+4	1.79E+4	1.0E+3
S_10_10_3.79	3.53E+4 (-)	7.8E+3	3.19E+4 (-)	6.8E+3	*	*	1.76E+4 (-)	2.5E+3	7.44E+3	3.7E+2
S_11_11_1.28	3.05E+2 (-)	1.7E+1	3.10E+2 (-)	1.0E+1	4.06E+2 (-)	1.7E+1	3.36E+2 (-)	2.3E+1	2.79E+2	1.1E+1
S_30_5_0.46	1.50E+2 (≈)	9.7E-1	1.50E+2 (≈)	6.9E-1	1.55E+2 (-)	1.7E+0	1.50E+2 (≈)	1.8E+0	1.50E+2	1.9E+0
S_15_10_1.17	3.39E+2 (≈)	1.0E+1	3.59E+2 (-)	9.0E+0	4.19E+2 (-)	1.1E+1	3.61E+2 (-)	2.1E+1	3.42E+2	8.0E+0
S_10_15_1.3	6.28E+2 (≈)	9.7E+0	6.44E+2 (-)	1.0E+1	8.07E+2 (-)	3.3E+1	6.55E+2 (-)	3.7E+1	6.30E+2	1.3E+1
S_20_10_0.47	1.04E+2 (-)	1.9E+0	1.03E+2 (-)	1.7E+0	1.13E+2 (-)	2.5E+0	9.85E+1 (-)	1.8E+0	9.67E+1	1.7E+0
S_20_10_0.96	3.41E+2 (≈)	7.0E+0	3.61E+2 (-)	4.5E+0	4.05E+2 (-)	1.2E+1	3.61E+2 (-)	2.4E+1	3.41E+2	7.2E+0
S_20_10_0.94	2.73E+2 (-)	8.5E+0	2.79E+2 (-)	3.7E+0	3.12E+2 (-)	6.5E+0	2.70E+2 (-)	1.6E+1	2.50E+2	3.6E+0
S_5_40_3.95	1.51E+4 (-)	7.3E+2	1.43E+4 (-)	5.1E+2	2.73E+4 (-)	2.2E+3	1.19E+4 (-)	4.8E+2	1.17E+4	6.3E+2
S_10_20_6.04	*	*	*	*	*	*	*	*	9.19E+4	4.1E+3
S_30_10_0.65	1.83E+2 (-)	3.9E+0	1.84E+2 (-)	3.4E+0	1.97E+2 (-)	4.6E+0	1.75E+2 (-)	7.9E+0	1.61E+2	2.9E+0
S_15_20_0.67	3.51E+2 (-)	7.0E+0	3.67E+2 (-)	1.8E+0	3.92E+2 (-)	6.1E+0	3.54E+2 (-)	1.0E+1	3.33E+2	4.5E+0
S_30_10_1.34	4.14E+2 (-)	1.6E+1	4.58E+2 (-)	8.0E+0	5.35E+2 (-)	2.9E+1	4.20E+2 (-)	5.8E+1	3.68E+2	8.5E+0
S_15_20_5.98	*	*	*	*	*	*	*	*	7.83E+4	2.1E+3
S_17_23_1.71	4.07E+2 (-)	2.4E+1	6.10E+2 (-)	2.5E+1	8.12E+2 (-)	4.1E+1	5.24E+2 (-)	5.4E+1	3.43E+2	1.5E+1
M_20_20_0.58	2.74E+2 (-)	5.7E+0	2.88E+2 (-)	2.5E+0	3.05E+2 (-)	4.4E+0	2.73E+2 (-)	5.5E+0	2.60E+2	3.7E+0
M_40_10_0.67	2.34E+2 (-)	3.4E+0	2.39E+2 (-)	1.3E+0	2.56E+2 (-)	3.8E+0	2.37E+2 (-)	7.2E+0	2.27E+2	3.2E+0
M_20_20_0.97	1.92E+2 (-)	1.1E+1	2.53E+2 (-)	6.3E+0	2.82E+2 (-)	1.2E+1	2.39E+2 (-)	2.1E+1	1.65E+2	6.3E+0
M_20_20_0.967	4.01E+2 (-)	7.8E+0	4.26E+2 (-)	5.7E+0	4.80E+2 (-)	1.3E+1	4.35E+2 (-)	3.1E+1	3.78E+2	5.9E+0
M_15_30_2.16	1.68E+3 (-)	7.4E+1	2.09E+3 (-)	5.8E+1	2.44E+3 (-)	1.0E+2	1.66E+3 (-)	1.8E+2	1.34E+3	3.6E+1
M_40_15_0.67	2.99E+2 (-)	3.1E+0	3.05E+2 (-)	2.2E+0	3.20E+2 (-)	3.7E+0	2.96E+2 (-)	1.2E+1	2.80E+2	4.3E+0
M_20_40_3.61	2.21E+4 (-)	3.0E+3	5.28E+4 (-)	4.6E+3	*	*	1.32E+4 (-)	4.0E+3	5.53E+3	4.3E+2
M_30_30_1.04	5.58E+2 (-)	8.9E+0	5.80E+2 (-)	6.7E+0	6.36E+2 (-)	7.5E+0	5.48E+2 (-)	4.1E+1	4.66E+2	6.4E+0
M_30_30_1.94	1.37E+3 (-)	6.3E+1	1.54E+3 (-)	4.3E+1	1.53E+3 (-)	8.6E+1	1.31E+3 (-)	7.8E+1	1.10E+3	4.0E+1
M_15_60_3.64	1.73E+4 (-)	1.3E+3	2.50E+4 (-)	1.2E+3	2.08E+4 (-)	1.4E+3	1.12E+4 (-)	1.9E+3	1.01E+4	4.8E+2
L_80_15_0.76	1.98E+2 (-)	2.3E+0	2.00E+2 (-)	2.4E+0	2.07E+2 (-)	3.5E+0	2.00E+2 (-)	1.1E+1	1.70E+2	3.6E+0
L_20_60_1.51	1.01E+3 (-)	2.9E+1	1.06E+3 (-)	9.6E+0	1.15E+3 (-)	1.4E+1	9.60E+2 (-)	2.7E+1	7.53E+2	1.9E+1
L_80_20_0.7	3.18E+2 (-)	3.1E+0	3.24E+2 (-)	2.1E+0	3.35E+2 (-)	4.1E+0	3.31E+2 (-)	1.5E+1	2.86E+2	5.5E+0
L_80_20_1.12	3.90E+2 (-)	6.2E+0	4.11E+2 (-)	4.6E+0	4.53E+2 (-)	1.1E+1	4.16E+2 (-)	2.2E+1	3.10E+2	7.2E+0
L_20_80_4.33	1.25E+5 (-)	2.5E+4	2.33E+5 (-)	2.0E+4	*	*	6.42E+4 (-)	5.2E+3	2.21E+4	2.0E+3
L_60_30_1.14	5.52E+2 (-)	9.9E+0	5.91E+2 (-)	6.9E+0	6.95E+2 (-)	2.2E+1	6.21E+2 (-)	3.7E+1	5.08E+2	7.2E+0
L_60_40_0.69	3.91E+2 (-)	3.4E+0	3.99E+2 (-)	3.0E+0	4.04E+2 (-)	4.4E+0	4.00E+2 (-)	1.2E+1	3.40E+2	3.0E+0
L_40_60_1.54	9.48E+2 (-)	1.7E+1	9.84E+2 (-)	8.9E+0	1.05E+3 (-)	1.5E+1	9.05E+2 (-)	2.3E+1	6.82E+2	2.0E+1
L_80_40_0.97	4.64E+2 (-)	5.4E+0	4.79E+2 (-)	3.9E+0	5.01E+2 (-)	8.5E+0	4.99E+2 (-)	2.0E+1	3.86E+2	5.2E+0
L_40_80_1.05	6.24E+2 (-)	8.1E+0	6.40E+2 (-)	5.0E+0	6.60E+2 (-)	9.5E+0	6.23E+2 (-)	1.7E+1	5.00E+2	6.6E+0
L_80_40_2.25	6.54E+3 (-)	1.0E+3	1.01E+4 (-)	8.8E+2	3.69E+3 (-)	3.4E+2	1.42E+4 (-)	5.0E+3	1.69E+3	6.0E+1
L_60_60_0.92	4.29E+2 (-)	5.3E+0	4.40E+2 (-)	3.3E+0	4.36E+2 (-)	6.2E+0	4.48E+2 (-)	1.8E+1	3.17E+2	4.0E+0
L_60_60_0.922	5.51E+2 (-)	6.3E+0	5.64E+2 (-)	4.8E+0	5.56E+2 (-)	5.8E+0	5.58E+2 (-)	1.5E+1	4.18E+2	6.9E+0
L_120_30_1.2	4.44E+2 (-)	6.1E+0	4.64E+2 (-)	4.9E+0	4.94E+2 (-)	1.2E+1	4.97E+2 (-)	1.8E+1	3.40E+2	6.7E+0
L_80_60_0.72	4.18E+2 (-)	4.1E+0	4.28E+2 (-)	3.9E+0	4.17E+2 (-)	4.7E+0	4.51E+2 (-)	1.5E+1	3.29E+2	2.8E+0
L_80_80_0.54	3.23E+2 (-)	3.3E+0	3.28E+2 (-)	3.5E+0	3.08E+2 (-)	2.5E+0	3.54E+2 (-)	1.5E+1	2.49E+2	1.8E+0
L_60_120_2.07	3.25E+3 (-)	6.5E+1	3.36E+3 (-)	4.6E+1	3.09E+3 (-)	8.2E+1	4.02E+3 (-)	3.3E+2	2.19E+3	6.9E+1
(+)/(≈)/(-)	0/5/45		1/2/47		1/0/49		1/2/47		–	

(+)/(≈)/(-) represent the method is statistically better/equal/worse than AC-ACO. * represents that the method cannot obtain a feasible solution in the given time. S_5_4_0.39, S_5_5_1.66, S_11_11_1.28, S_17_23_1.7 and M_20_20_0.97 are instances designed in [14], and other instances are designed in this paper.

find feasible solutions on all the test cases is the repair mechanism and the adaptive heuristic information used in the solution construction process. In most cases, these two factors can guarantee the feasibility of the solutions.

The experiment shows that it is hard for MA-MLS, MA-OLS, EDA, and ILS to find feasible solutions on the instances with large ϕ values. The reason is that MA-MLS, MA-OLS, EDA and ILS adopt purely random initialisation processes and offspring producing processes without using any domain knowledge. The probability of constructing a feasible solution within a purely random method will decrease when the instances have large ϕ. Considering an extreme case that every task requires all the M robots to execute, the probability of constructing a feasible solution by a purely random method is $\frac{N!}{N!^M}$. As M and N increase, the probability will become very small. Since S_10_20_6.04 and S_15_20_5.98 have large N, M, and ϕ, MA-MLS, MA-OLS, EDA, and ILS cannot obtain a feasible solution under the given computational budget.

2) Objective value: First, from the results of the Wilcoxon rank-sum test, it is clear that AC-ACO significantly outperformed all the compared algorithms for most instances. The reason is that AC-ACO successfully employs domain

knowledge on robot ability and collaborations during the solution construction process. On the contrary, the compared algorithms did not employ sufficient domain knowledge, and wasted time on generating and evaluating poor solutions in the enormous solution space.

Second, we analyse the results from the perspective of the scale of instances. On most small-scale instances (whose names start with "S_"), AC-ACO is the best method among all the compared algorithms. MA-OLS, MA-MLS, and ILS have similar performances with AC-ACO on some small-scale instances such as S_30_5_0.46, S_15_10_1.17, S_10_15_1.3, and S_20_10_0.94. Especially, on S_5_4_0.39 and S_5_5_1.66, MA-OLS, MA-MLS, and EDA outperform AC-ACO. The reason is that the inaccurate domain knowledge may lead AC-ACO to local optima. In contrast, MA-OLS, MA-MLS, EDA, and ILS can explore the solution space efficiently on the small-scale instances under the given computational budget. For the medium-scale and large-scale instances, EDA, MA-OLS, MA-MLS, and ILS, which do not use the domain knowledge, cannot match the performance of AC-ACO. The solution space of the MPDA problem grows very quickly with the growth of M and N. The results showed that EDA, MA-OLS, MA-MLS, and ILS cannot search effectively in the huge solution space. AC-ACO can effectively focus on the potentially promising solutions by using the adaptive heuristic information and the elite local search mechanism. Thus, AC-ACO outperforms the other methods on the majority of the medium-scale and large-scale instances.

3) Convergence curve: To analyse the convergence behaviour of the compared algorithms, we give the time complexity of the five compared approaches first. When all these methods are given the same number of fitness evaluations (NFE), the worst-case time complexity can be shown as follows. The worst-case time complexity of EDA is $NM(NM + M \times \mathrm{NFE} + (\mathrm{NFE})^2 log(M))$ [12]. The worst-case time complexity of MA-OLS, MA-MLS, and ILS is $\mathrm{O}(\mathrm{NFE} \times N \times M(N + M))$ [14], [25]. Due to the repair mechanism, the worst-case time complexity of AC-ACO is $\mathrm{O}(\mathrm{NFE} \times N \times M(M^2 + N))$. Theoretically speaking, the worst-case time complexity of AC-ACO is lower than EDA, but slightly higher than MAs and ILS. In practice, if the constraint is not so tight, the complexity of AC-ACO becomes comparable with that of MA-OLS, MA-MLS, and ILS.

Fig. 6 shows the convergence curves of these meta-heuristic methods on four representative instances with different scales. From the figure, it is clear that AC-ACO converges much faster than the other compared algorithms. It has the best initial objective value among the algorithms, and can always keep a stable convergence speed during the evolutionary process for all the tested instances. On the other hand, due to the high computational complexity, EDA has the slowest convergence speed among the compared algorithms, and its convergence speed gradually decreases as the scale of instances increases. ILS has the fastest convergence speed in the early stage, however, it slows down apparently in the later phase of evolution. The convergence speeds of MA-MLS and MA-OLS are between that of ILS and EDA.

Overall, by checking the constraint handling ability, objective value, and convergence speed, it can be concluded that AC-ACO outperforms the state-of-the-art methods in solving the MPDA problem.

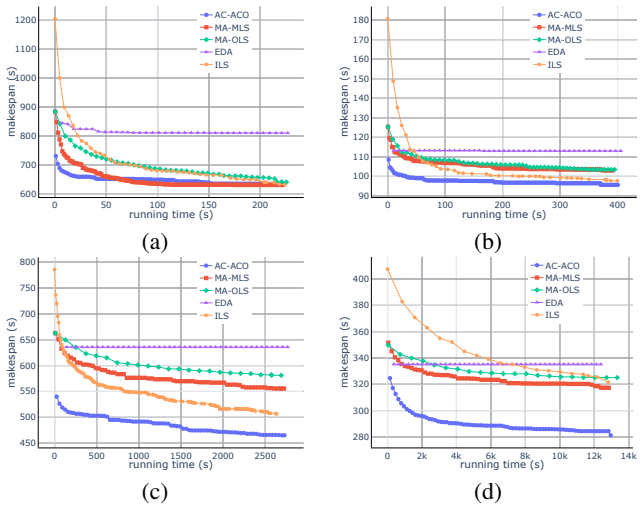


Fig. 6. Evolutionary progress of different meta-heuristic methods for the selected problem instances. (a) S_15_10_1.17 (b) S_20_10_0.47 (c) M_30_30_1.04 (d) L_80_20_1.12

D. Effectiveness of Solution Construction Process

The proposed solution construction process in AC-ACO uses multiple ants and multiple pheromone matrices to construct the solution in parallel, which matches the process of robots executing tasks well. To further show the efficiency of the solution construction process, two questions will be explored: 1) whether the adaptive heuristic information is effective 2) whether the repair mechanism will promote the algorithm's performance. In addition, the local search process will weaken the importance of the designed heuristic information and mechanisms in the solution construction process [50]. Thus, all the following ACO algorithms do not use the local search mechanism in this subsection.

1) Investigation of heuristic information: To verify the effectiveness of the designed adaptive heuristic information, S_5_10_3.67, S_20_10_0.47, M_20_20_0.967, and L_60_60_0.92 with different scales and ϕ are used as the tested cases. First, an AC-ACO without the cluster dynamic heuristic information, an AC-ACO without the dynamic heuristic information, and an AC-ACO without the heuristic information are designed as the control group, denoted as AC-ACO-WC, AC-ACO-WD, and AC-ACO-WH respectively. Fig. 7 shows the curves of the average minimum fitness over the generations during the evolutionary process of these three algorithms and AC-ACO.

From the figure, it can be seen that AC-ACO outperforms other ACO algorithms, which further implies the designed heuristic information is effective. AC-ACO-WH performs worst on S_5_10_3.67, M_20_20_0.967, and L_60_60_0.92, but AC-ACO-WH outperforms AC-ACO-WD on S_20_10_0.47 which have a small ϕ. It can be seen that the all designed heuristic information can play a positive role in most of the instances, except that the static heuristic information plays a negative role in the instances with

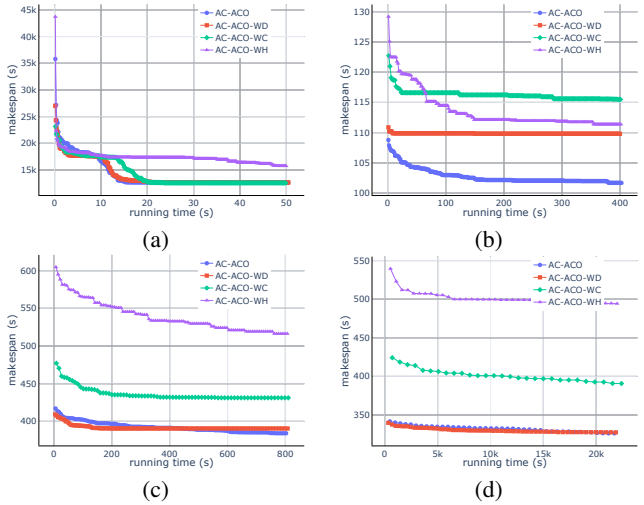


Fig. 7. Evolutionary progress of AC-ACO with different heuristic information for the selected problem instances. (a) S_5_10_3.67 (b) S_20_10_0.47 (c) M_20_20_0.967 (d) L_60_60_0.92

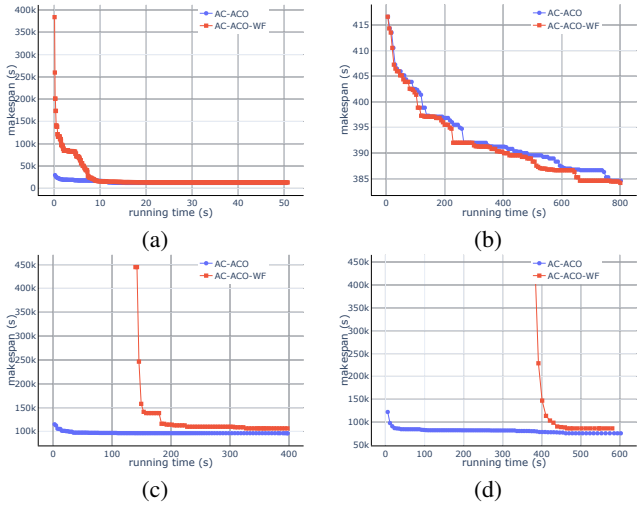


Fig. 8. Evolutionary progress of AC-ACO-WF and AC-ACO for the selected problem instances. (a) S_5_10_3.67 (b) M_20_20_0.967 (c) S_10_20_6.04 (d) S_15_20_5.98

small ϕ values. The reason is that the purely static heuristic information makes robots execute a task with small travelling cost together implicitly. The task with small travelling cost will get a higher probability for all robots' choices so that the number of robots aggregating at the task will be more than needed. However, the plan in which robots scatter to execute tasks fits the characteristics of the instances with small ϕ values well. From the figure, compared with AC-ACO-WD, AC-ACO-WC and AC-ACO have a better convergence speed and final objective value. Thus, we can conclude that AC-ACO-WD is outperformed by AC-ACO-WC and AC-ACO and the efficiency of dynamic heuristic information is verified. Comparing AC-ACO-WC and AC-ACO, we can find these two algorithms do not have a significant difference on S_5_10_3.67, M_20_20_0.967, and L_60_60_0.92. On S_20_10_0.47, AC-ACO outperforms AC-ACO-WD because the designed scattered heuristic information gives a correct guide during the search process. The conclusion is that the designed adaptive heuristic information is useful for the performance of AC-ACO.

2) Investigation of the repair process: To verify the effectiveness of the designed repair mechanism, S_5_10_3.67, M_20_20_0.967, S_10_20_6.04, and S_15_20_5.98 with different levels of tight constraints are used as the tested cases. An AC-ACO without the designed repair mechanism is used as the control group, denoted as AC-ACO-WF.

Fig. 8 shows the curves of the average minimum fitness in a generation during the evolutionary process of this algorithm and AC-ACO. From Fig. 8, AC-ACO and AC-ACO-WF do not have a significant difference at the final objective value on S_5_10_3.67 and M_20_20_0.967. However, AC-ACO has a faster convergence speed during the searching process on S_5_10_3.67. The repairing process in the solution construction process can convert infeasible solutions to feasible solutions so that the probability of obtaining a good solution becomes large. From Table IV, it can be found that S_10_20_6.04 and S_15_20_5.98 have the tightest constraints among benchmarks. AC-ACO has a better initial objective value and better final objective value than AC-ACO-WF. The conclusion is that the designed repair mechanism is useful.

E. Effectiveness of Local Search

An AC-ACO with the local search strategy where several solutions are self-learning and an AC-ACO without the local search strategy are used as the control group, denoted as AC-ACO-ELS and AC-ACO-WLS. Every solution has a certain probability to participate in the local search in AC-ACO-ELS, and the probability is set 0.2 as same as in [14], [48]. We use the mean and standard deviation of best-so-far individuals' objective value as the comparison metric, which is shown as Table SIII in the supplemental material. Fig. 9 shows curves of the average minimum fitness in a generation during the evolutionary process of AC-ACO with different strategies.

From the table, some general views of the results can be deduced. AC-ACO achieved the minimum fitness among the three algorithms at the end of evolution on all instances except for S_5_10_0.93. AC-ACO-ELS achieved the worst performance among these three algorithms. Fig. 9 shows that AC-ACO can maintain a fast and steady convergence speed during the whole optimisation process. The convergence speed of AC-ACO-ELS is the slowest and the convergence speed of AC-ACO-WLS is between AC-ACO and AC-ACO-ELS. Since the time complexity of the decoding method is $\mathrm{O}(M \times N(N+M))$, the time complexity of searching the neighbourhood of the original solution is $\mathrm{O}(nsb \times (M \times N(N+M)))$, which is an enormous cost. AC-ACO-ELS which searches every solution's neighbourhood pays so much effort on the individual's exploitation that its convergence speed decreases.

Overall, by comparing the convergence speeds and final objective values of these algorithms, we can conclude that the designed local search strategy is effective.

V. Conclusions and Future Work

The MPDA problem is an emerging and novel problem which can describe a wide large of real-world applications

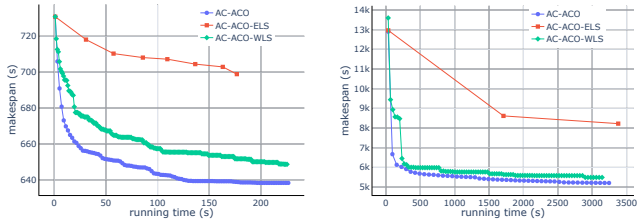


Fig. 9. Evolutionary progress of AC-ACO with different local search strategies for the selected problem instances. (a) S_10_15_1.3 (b) M_20_40_3.61

and complement traditional optimisation problems. Due to the difficulties and challenges caused by the complex dependencies among robots and tasks in the MPDA problem, this paper designed an AC-ACO algorithm to address the MPDA problem efficiently. First, in the designed AC-ACO method, a coordinated ant team and its corresponding multiple pheromone matrices are suitable for the parallel execution process of the robots in the MPDA problem such that the efficiency of the solution construction process is promoted. Meanwhile, the adaptive heuristic information and pheromone-based repair mechanism were developed and embedded in the solution construction process to aid AC-ACO to focus on the more promising solutions. Finally, the designed local search method was conducted on the best-so-far solution to promote the exploitation ability of AC-ACO. The framework of solution constructing process using the coordinated ant team in AC-ACO can be extended to many other parallel and collaboration systems. Experimental results show that AC-ACO significantly outperforms the state-of-the-art algorithms in terms of solution quality and convergence speed, as well as stability. The effectiveness and necessity of the repair mechanism, the adaptive heuristic values, and the local search method have also been demonstrated by further analysis.Besides, from the experimental results, we can find that the adaptive heuristic information contributes the most to the performance of the AC-ACO algorithm in most instances. However, if there is an extremely large task that requires many robots to execute, the repair process is more important than the other designed strategies.

There are still some aspects that need to be further generalized to cope with real-world multi-robot systems in the bushfire elimination mission and the medical resource scheduling mission: 1) multi-objective optimisation for the maximal completion time and the number of robots; 2) designing distributed algorithms to obtain an acceptable plan in a short time; 3) considering the communication delay and loss problem among robots; 4) implementing a practical multi-robot system to verify algorithms for the MPDA problem.

REFERENCES

[1] B. Xin, Y.-G. Zhu, Y.-L. Ding, and G.-Q. Gao, "Coordinated motion planning of multiple robots in multi-point dynamic aggregation task," in *Proceedings of the 12th IEEE International Conference on Control and Automation*. IEEE, 2016, pp. 933–938.

[2] N. Zheng, S. Du, J. Wang, H. Zhang, W. Cui, Z. Kang, T. Yang, B. Lou, Y. Chi, H. Long *et al.*, "Predicting covid-19 in china using hybrid ai model," *IEEE Trans. Cybern.*, 2020.

[3] V. Akbari and F. S. Salman, "Multi-vehicle synchronized arc routing problem to restore post-disaster network connectivity," *Eur. J. Oper. Res*, vol. 257, no. 2, pp. 625–640, 2017.

[4] 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, 2017.

[5] A. Khan, B. Rinner, and A. Cavallaro, "Cooperative robots to observe moving targets," *IEEE Trans. Cybern.*, vol. 48, no. 1, pp. 187–198, 2016.

[6] G. Rigatos, P. Siano, P. Wira, K. Busawon, and R. Binns, "A nonlinear h-infinity control approach for autonomous truck and trailer systems," *Unmanned Syst.*, vol. 8, no. 01, pp. 49–69, 2020.

[7] G. Skorobogatov, C. Barrado Muxí, and E. Salamí San Juan, "Multiple uav systems: a survey," *Unmanned Syst.*, vol. 8, no. 2, pp. 149–169, 2020.

[8] Y. Li, H. Soleimani, and M. Zohal, "An improved ant colony optimization algorithm for the multi-depot green vehicle routing problem with multiple objectives," *J. Clean. Prod.*, vol. 227, pp. 1161–1172, 2019.

[9] X. Li, L. Gao, Q. Pan, L. Wan, and K.-M. Chao, "An effective hybrid genetic algorithm and variable neighborhood search for integrated process planning and scheduling in a packaging machine workshop," *IEEE Trans. Syst. Man CY-S.*, vol. 49, no. 10, pp. 1933–1945, 2019.

[10] R. Pellerin, N. Perrier, and F. Berthaut, "A survey of hybrid metaheuristics for the resource-constrained project scheduling problem," *Eur. J. Oper. Res*, vol. 280, no. 2, pp. 395–416, 2020.

[11] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job shop scheduling," *IEEE Trans. Cybern.*, 2020. Doi: 10.1109/TCYB.2020.3024849.

[12] B. Xin, S. Liu, Z. Peng, and G. Gao, "An estimation of distribution algorithm for multi-robot multi-point dynamic aggregation problem," in *Proceedings of the 2018 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2018, pp. 775–780.

[13] S. Lu, B. Xin, L. Dou, and L. Wang, "A multi-model estimation of distribution algorithm for agent routing problem in multi-point dynamic task," in *Proceedings of the 2018 37th Chinese Control Conference (CCC)*. IEEE, 2018, pp. 2468–2473.

[14] G. Gao, M. Yi, X. Bin, J. Ya-Hui, and B. Will, "A memetic algorithm for the task allocation problem on multi-robot multi-point dynamic aggregation missions," in *Proceedings of the 2020 IEEE world congress on computational intelligence (WCCI)*. IEEE, 2020.

[15] R. Hao, J. Zhang, B. Xin, C. Chen, and L. Dou, "A hybrid differential evolution and estimation of distribution algorithm for the multi-point dynamic aggregation problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2018, pp. 251–252.

[16] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Trans. Syst. MAN Cy. B.*, vol. 26, no. 1, pp. 29–41, 1996.

[17] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, 1997.

[18] X. Chen, P. Zhang, G. Du, and F. Li, "Ant colony optimization based memetic algorithm to solve bi-objective multiple traveling salesmen problem for multi-robot systems," *IEEE Access*, vol. 6, pp. 21 745–21 757, 2018.

[19] M. Dorigo and T. Stützle, "Ant colony optimization: overview and recent advances," in *Handbook of metaheuristics*. Springer, 2019, pp. 311–351.

[20] W.-N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements," *IEEE Trans. Syst. Man Cy. C.*, vol. 39, no. 1, pp. 29–43, 2008.

[21] Y.-H. Jia, W.-N. Chen, H. Yuan, T. Gu, H. Zhang, Y. Gao, and J. Zhang, "An intelligent cloud workflow scheduling system with time estimation and adaptive ant colony optimization," *IEEE Trans Syst. Man CY-S.*, pp. 1–16, 2019.

[22] P. Toth and e. Vigo, Daniele, *The vehicle routing problem*. Society for Industrial and Applied Mathematics, 2002.

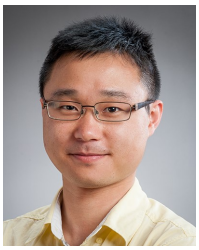
[23] P. Toth and e. Daniele Vigo, *Vehicle routing: problems, methods, and applications*. Society for Industrial and Applied Mathematics, 2014.

[24] Y. Mei, K. Tang, and X. Yao, "Capacitated arc routing problem in uncertain environments," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[25] X. Wang, T.-M. Choi, Z. Li, and S. Shao, "An effective local search algorithm for the multidepot cumulative capacitated vehicle routing problem," *IEEE Trans Syst. Man CY-S.*, 2019.

[26] C. Archetti, M. G. Speranza, and A. Hertz, "A tabu search algorithm for the split delivery vehicle routing problem," *Transport. Sci.*, vol. 40, no. 1, pp. 64–73, 2006.
[27] T. Cheng and C. Sin, "A state-of-the-art review of parallel-machine scheduling research," *Eur. J. Oper. Res*, vol. 47, no. 3, pp. 271–292, 1990.
[28] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3202–3212, 2008.
[29] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.
[30] M. U. Arif and S. Haider, *A Flexible Evolutionary Algorithm for Task Allocation in Multi-robot Team*, ser. Lecture Notes in Computer Science, 2018, book section Chapter 9, pp. 89–99.
[31] M. U. Arif, "A generic evolutionary algorithm for efficient multi-robot task allocations," in *Proceedings of the Canadian Conference on Artificial Intelligence*. Springer, 2019, pp. 486–491.
[32] T. Stutzle and H. Hoos, "Max-min ant system and local search for the traveling salesman problem," in *Proceedings of 1997 IEEE international conference on evolutionary computation (ICEC'97)*. IEEE, 1997, pp. 309–314.
[33] C. Fahy, S. Yang, and M. Gongora, "Ant colony stream clustering: A fast density clustering algorithm for dynamic data streams," *IEEE Trans. Cybern.*, vol. 49, no. 6, pp. 2215–2228, 2018.
[34] W.-N. Chen, D.-Z. Tan, Q. Yang, T. Gu, and J. Zhang, "Ant colony optimization for the control of pollutant spreading on social networks," *IEEE Trans. Cybern.*, 2019.
[35] M. Mavrovouniotis, F. M. Müller, and S. Yang, "Ant colony optimization with local search for dynamic traveling salesman problems," *IEEE Trans. Cybern.*, vol. 47, no. 7, pp. 1743–1756, 2016.
[36] X. Yu, W.-N. Chen, T. Gu, H. Yuan, H. Zhang, and J. Zhang, "Aco-a*: Ant colony optimization plus a* for 3-d traveling in environments with dense obstacles," *IEEE Trans. Evol. Comput.*, vol. 23, no. 4, pp. 617–631, 2018.
[37] M. López-Ibáñez, T. Stützle, and M. Dorigo, "Ant colony optimization: A component-wise overview." 2018.
[38] X. Li, Z. Liu, and F. Tan, "Multi-robot task allocation based on cloud ant colony algorithm," in *Proceedings of the 2017 International Conference on Neural Information Processing*. Springer, 2017, pp. 3–10.
[39] G. P. Rajappa, J. H. Wilck, and J. E. Bell, "An ant colony optimization and hybrid metaheuristics algorithm to solve the split delivery vehicle routing problem," *Int. J. Appl. Ind. Eng.*, vol. 3, no. 1, pp. 55–73, 2016.
[40] X. Wang, T.-M. Choi, H. Liu, and X. Yue, "Novel ant colony optimization methods for simplifying solution construction in vehicle routing problems," *IEEE Trans. Intell. Transp.*, vol. 17, no. 11, pp. 3132–3141, 2016.
[41] H. Zhang, Q. Zhang, L. Ma, Z. Zhang, and Y. Liu, "A hybrid ant colony optimization algorithm for a multi-objective vehicle routing problem with flexible time windows," *Inform. Sciences*, vol. 490, pp. 166–190, 2019.
[42] X. Wang, T.-M. Choi, H. Liu, and X. Yue, "A novel hybrid ant colony optimization algorithm for emergency transportation problems during post-disaster scenarios," *IEEE Trans Syst. Man CY-S.*, vol. 48, no. 4, pp. 545–556, 2016.
[43] P. C. Pendharkar, "An ant colony optimization heuristic for constrained task allocation problem," *J. Comput. Sci.*, vol. 7, pp. 37–47, 2015.
[44] Z.-M. Huang, W.-N. Chen, Q. Li, X.-N. Luo, H.-Q. Yuan, and J. Zhang, "Ant colony evacuation planner: An ant colony system with incremental flow assignment for multipath crowd evacuation," *IEEE Trans. Cybern.*, 2020.
[45] Y. Wang, H. Liu, H. Long, Z. Zhang, and S. Yang, "Differential evolution with a new encoding mechanism for optimizing wind farm layout," *IEEE Trans. Industr. Inform.*, vol. 14, no. 3, pp. 1040–1054, 2017.
[46] S. Mancini, "A combined multistart random constructive heuristic and set partitioning based formulation for the vehicle routing problem with time dependent travel times," *Comput. Oper. Res.*, vol. 88, pp. 290–296, 2017.
[47] M. Nouiri, A. Bekrar, A. Jemai, S. Niar, and A. C. Ammari, "An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem," *J. Intell. Manuf.*, vol. 29, no. 3, pp. 603–615, 2018.
[48] K. Tang, Y. Mei, and X. Yao, "Memetic algorithm with extended neighborhood search for capacitated arc routing problems," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 1151–1166, 2009.
[49] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, "New benchmark instances for the capacitated vehicle routing problem," *Eur. J. Oper. Res*, vol. 257, no. 3, pp. 845–858, 2017.
[50] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theor. Comput. Sci.*, vol. 344, no. 2-3, pp. 243–278, 2005.

Guanqiang Gao received the B.S. degree in automation from the Beijing Institute of Technology, Beijing, China, in 2015. He is currently pursuing the Ph.D. degree with the School of Automation, Beijing Institute of Technology, Beijing. His current research interests include evolutionary computation and multi-robot system.



Yi Mei (M'09-SM'18) received the BSc and PhD degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively. He is currently a Senior Lecturer at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research interests include evolutionary scheduling and combinatorial optimisation, machine learning, genetic programming, and hyper-heuristics. He has over 100 fully referred publications, including the top journals in EC and Operations Research such as IEEE TEVC, IEEE TCYB, Evolutionary Computation Journal, European Journal of Operational Research, ACM Transactions on Mathematical Software. He serves as a Vice-Chair of the IEEE CIS Emergent Technologies Technical Committee, and a member of Intelligent Systems Applications Technical Committee.

Ya-Hui Jia (M'20) received the B.Eng. and Eng.D. degrees from Sun Yat-sen University, China, in 2013 and 2019, respectively. He is currently a postdoctoral research fellow at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research interests include evolutionary computation, combinatorial optimization, software engineering, cloud computing, and intelligent transportation.

Will Browne Will N. Browne's research interests are in developing Artificial Cognitive Systems. His background is in Mechanical Engineering (B.Eng. degree (Hons.)) from the University of Bath, U.K., 1993, and both the M.Sc. degree in Energy and the EngD degree (Engineering Doctorate Scheme) from the University of Wales, Cardiff, U.K., in 1994 and 1999, respectively. After lecturing for eight years at the Department of Cybernetics, University of Reading, U.K., he is now an Associate Professor with the Evolutionary Computation Research Group, School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. He has published over 100 academic papers in books, refereed international journals, and conferences.

Bin Xin (Member, IEEE) received the B.S. degree in information engineering and the Ph.D. degree in control science and engineering, both from the Beijing Institute of Technology, Beijing, China, in 2004 and 2012, respectively. He was an academic visitor at the Decision and Cognitive Sciences Research Centre, the University of Manchester, from 2011 to 2012. He serves as an associate editor of the journal Unmanned Systems and many other international journals. He is currently a Professor with the School of Automation, Beijing Institute of Technology. His current research interests include search and optimization, evolutionary computation, unmanned systems, and multiagent systems.