An Asynchronous Shuffled Frog-Leaping With Feasible Jaya Algorithm for Uncertain Task Rescheduling Problem in UAV Emergency Networks

Qiuji Luan, Hongyan Cui, *Senior Member, IEEE*, Xi Yu, Lifeng Zhang, and Biplab Sikdar, *Senior Member, IEEE*

***Abstract*— Unmanned aerial vehicles (UAVs) have a great potential for assigning search and rescue operations in emergency scenarios. However, emergency scenarios are complex and unknown, regarding UAVs to reschedule to effectively adapt to the changing environment, and existing literature addressing this challenge is limited. To address this open problem, we consider a task rescheduling problem with uncertainties such as task insertion, edge computing node (ECN) destruction, and parameter fluctuation in UAV-assisted emergency networks. The goal is to minimize the fine-grained makespan, defined as the ratio of makespan to ECNs idle time, that simultaneously characterizes the optimization of rescheduling efficiency and ECNs utilization. To address the problem, we propose an asynchronous shuffled frog-leaping with feasible Jaya (ASFJ) algorithm. In ASFJ, an asynchronous shuffled frog-leaping method independently evolves memeplexes, thereby avoiding forced information coverage. Two feasible local search operators promote the search capability and feasibility of the algorithm. Finally, we verify the advantages of the ASFJ in terms of makespan, effectiveness, and fine-grained makespan. ASFJ can save 3.83ms makespan and outperform 11.2% fine-grain makespan in insertion rescheduling. The effectiveness of destruction rescheduling is improved by at least 16%.**

***Index Terms*— UAV emergency network, uncertain, task rescheduling, fine-grained makespan.**

I. Introduction

TASK scheduling plays an important role in ensuring the efficiency and effectiveness of unmanned aerial vehicle (UAV) communication. It characterizes the process of assigning tasks to suitable edge computing nodes (ECNs), including UAVs or ground devices, and the purpose is to minimize communication costs [1], makespan [2], scheduling costs [3], etc. Scheduling can be divided into static and dynamic. In static scheduling, all tasks are executed sequentially on the assigned ECNs [4], [5]. Dynamic scheduling is not only a response to uncertain events but also a repair to the original scheduling that was adjusted passively.

The complex and unpredictable emergency rescue scenario is a typical application of dynamic scheduling. Rescuers cannot enter the disaster area, which makes task scheduling invisible, intangible, and uncertain. Due to the advantages such as flexible deployment and mobility, UAVs are regarded as relay loads to deliver supplies [6]. In addition, UAVs and ground devices are also regarded as edge servers to provide computing, data analysis, and decision-making services for delay-sensitive and resource-intensive tasks such as image recognition, environmental monitoring, and disaster investigation. However, it has to be considered that computing ability and battery life are constrained by production costs. Therefore, in an uncertain environment, how to reschedule tasks dynamically to respond to requests, while making full use of limited resources still needs to be solved.

Most of the research on task rescheduling focuses on the field of industrial manufacturing(IM-F) [7]. However, in the considered UAV-assisted emergency network, task rescheduling faces more complex, unknown, and uncontrollable challenges, which are quite different from the IM-F as follows:

1) Lower workable time tolerance: In IM-F, considering the actual demand of maximizing output, the supply of energy is generally unconstrained, so that workable time of machines is unlimited. On the contrary, the restricted power of the ECNs constrains the workable time. Rescheduling needs to control energy consumption and improve the utilization of ECN as much as possible.
2) Higher processing node compatibility: In IM-F, to maintain production, the degraded operation will be carried out in case of machine breakdown, which will reduce productivity [8]. In the UAV emergency network, the ECNs are regarded as edge servers with fundamental hardware and software configurations, which can adapt to almost all tasks.

Manuscript received 9 March 2023; revised 19 December 2023 and 7 May 2024; accepted 10 June 2024. Date of publication 16 July 2024; date of current version 4 October 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62171049. The Associate Editor for this article was M. H. Anisi. *(Corresponding author: Hongyan Cui.)*

Qiuji Luan, Hongyan Cui, Xi Yu, and Lifeng Zhang are with the State Key Laboratory of Networking and Switching Technology and Beijing Laboratory of Advanced Information Networks, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: luanqiuji@bupt.edu.cn; cuihy@bupt.edu.cn; YUSY@bupt.edu.com; zhanglf@bjmu.edu.cn).

Biplab Sikdar is with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119077 (e-mail: bsikdar@nus.edu.sg).

3) Different evaluation indicators: Stability is the main optimization indicator for rescheduling problems in IM-F. It is expressed as the deviation between rescheduling and original scheduling with the assumption that the original scheduling is optimal [9]. On the contrary, the emergency environment changes in real-time, and more attention should be paid to rescue efficiencies, such as task makespan and resource utilization.

Therefore, we study a task rescheduling problem tailored to the specific demands and characteristics of UAV-assisted emergency networks. We propose a metric called fine-grained makespan (Fgm) to evaluate rescue efficiency, capturing task execution efficiency and accounting for resource limitations at the ECNs. Fgm is modeled as the ratio of makespan to ECNs' idle time.

In the limited existing literature, Yan et al. [10] estimated uncertain user status and determined the UAV scheduling strategy, which allows UAVs to move across regions to shorten the communication gap. Taking into account the imperfect knowledge of the angle of departure caused by UAV jittering, user location uncertainty, wind speed uncertainty, and polygonal no-fly zones, Xu et al. [11] employed monotonic optimization theory and semidefinite programming relaxation to solve the proposed nonconvex problem. These efforts focus on characterizing uncertainty with a data-driven approach, and naturally, it is assumed that the original scheduling decision will not change once formed, which is not always justified. As a matter of fact, it is necessary to adjust the decision to adapt to the real-time conditions in the emergency network. Therefore, an event-driven rescheduling solution is imperative requirements to respond to uncertain factors and repair the impact on the original scheduling decision.

In UAV-assisted emergency networks, event-driven rescheduling includes but is not limited to task insertion and ECN destruction. Chen et al. [12] characterized the uncertainty of the space-air-ground integrated network as the variability of environment and actual demand, and the amount of arriving tasks. Under the incomplete information regarding users' random arrivals and private service valuations, Wang et al. [13] proposed an optimal dynamic pricing scheme to balance hover time and service capacity. However, these efforts do not consider the possibility of ECNs disappearing or being subject to emergency recall in a hostile environment. This can lead to the failure of pending subtask execution on the broken-down ECNs, as well as potential delays in executing subtasks that depend on the failed subtasks. In addition, they overlook the real-time observation status of tasks and only consider the original input features. As a result, driven by uncertainty, low-competition tasks may be continuously delayed or even exceed their deadlines. Therefore, according to the state of the task in the whole rescheduling process, we establish task insertion and ECN destruction rescheduling models to analyze the impact of uncertain events on task scheduling performance. Furthermore, we design a state phase function to improve task execution satisfaction.

The task rescheduling problem has been proven to be an NP-hard problem [14]. To achieve solutions within a reasonable computational time, heuristic or metaheuristic optimization algorithms have emerged as a trend in recent years, such as genetic algorithm (GA) [15], ant colony algorithm [16], and Differential Evolution (DE) algorithm [17], which exhibit great capabilities in searching for the optimal solution. The shuffled frog-leaping algorithm (SFLA) is a meta-heuristic with the advantages of GA and social behavior-based particle swarm optimization (PSO) [18]. Its main benefits include fast convergence speed and effective algorithm structures containing local search and global information exchanges. However, in SFLA, the sizes and local search counts of all memeplexes always remain equal, and shuffling is enforced synchronously at a fixed frequency. This may cause the information of some memeplexes to be overwritten. Hence, to allow memeplexes that do not meet the local search termination conditions to continue to search for high-quality solutions, or escape from poorer solutions, SFLA needs to be optimized.

The Jaya algorithm is a novel meta-heuristic algorithm proposed by Rao in 2016 [19]. Its key idea is to search in the direction of the best solution and stay away from the worst solution. The Jaya algorithm does not have specific parameters, which can reduce parameter tuning effort. Jaya has only one search operator. The research on using Jaya to solve the rescheduling problem mainly focuses on the flexible job shop problem (FJSP). Reference [20] proposed a self-learning discrete Jaya to address the new job insertion. Reference [21] designed a local search operator and an initializing rule for machine recovery to improve the performance of Jaya. The goal is to minimize the instability and makespan caused by uncertainty. Besides, Jaya has been widely applied to solve various engineering problems, such as electrical discharge machining drilling of MoSi2-SiC composites [22], automotive turbocharger systems [23]. In addition, Rao [24] verified the superiority of Jaya over heuristic algorithms such as GA. Hence, the simple and effective Jaya algorithm is easier to deploy to solve real-life optimization problems [25]. However, when the scale of the problem becomes larger, the Jaya algorithm has the disadvantages of premature convergence and insufficient exploration ability.

Therefore, the properties of SFLA and Jaya motivate us to propose the ASFJ algorithm, in which Jaya searches for the local optimal solution of each memeplex, and the stagnation threshold assists the memeplex to shuffle asynchronously. The advantages of the proposed ASFJ formation include: i) As a local search operator for each memeplex rather than the entire population in SFLA, Jaya can cleverly avoid the limitation of low capability in larger search spaces; ii) The simplicity and efficiency of Jaya reduce the complexity of ASFJ and increase the local search ability of SFLA. In all, ASFJ effectively overcomes the shortcomings of the basic algorithms and enhances their advantages.

The main contributions of this article are as follows.

1) We develop an uncertainty-aware task rescheduling strategy with stochastic tasks insertion, ECNs destruction, and parameter fluctuation in a UAV- assisted emergency network. To evaluate the rescue efficiency of considered emergency scenarios more appropriately, we define a parameter that combines "task execution efficiency"

and "resource utilization of ECNs", which is called fine-grained makespan (Fgm). Among the designed parameter, a state phase function (SPF) is proposed to avoid high-competitive tasks being prioritized continuously. Our goal is to minimize the Fgm.

2) We propose an asynchronous shuffled frog-leaping with a feasible Jaya algorithm (ASFJ), which includes three components: (i) a heuristic population initialization method guided by energy consumption and execution time to obtain high-quality initial solutions; (ii) an asynchronous shuffled frog-leaping method that enhances the global and local search capabilities through an iterative process alternating between information exchange and asynchronous evolution; (iii) two feasible local search operators that not only enable the algorithm to escape local optima but also ensure the feasibility.
3) Extensive experiments on deterministic and uncertain scenarios are conducted to verify the efficiency of the proposed ASFJ. The performance of the ASFJ in terms of makespan, effectiveness, and fine-grained makespan to resist the influence of uncertainty are tested. Simulation results show that ASFJ outperforms the benchmark algorithms.

The rest of this article is organized as follows. Section II reviews relevant work. Section III mainly describes the system modeling and problem formulation in detail. Section IV presents the proposed ASFJ algorithm. Section V verifies the performance and discusses the main advantages of the ASFJ. Finally, Section VI concludes this article.

II. Literature Review

Task rescheduling is characterized by the fact that the original scheduling decision must be modified due to the disturbance of real-time uncertain factors. In the field of industrial manufacturing, there have been many studies on task rescheduling. Sun et al. [26] presented a novel partial repair rescheduling solution that consists of a criterion and a scheduler. Milica et al. [27] proposed a multi-objective grey wolf optimizer methodology to efficiently schedule material transport systems based on an intelligent single mobile robot. An et al. [28] focused on a flexible job-shop rescheduling problem for new job insertion and machine preventive maintenance. However, there is limited work on task rescheduling in the UAV emergency scenario, in most cases, there is more complexity and greater unknown challenges.

For the existing research on uncertainty-aware task rescheduling in emergency scenarios, uncertain factors are divided into inherent parameter uncertainty and scenario-based uncertainty. The inherent parameter uncertainty is attributed to the influence of external uncontrollable environmental factors such as airflow, which causes the execution time of the ECNs fluctuate. The ECNs are equipped with multiple functional sensors, which makes it difficult to obtain accurate execution time, because the functions affect each other when an ECN plays many roles at the same time, such as communication, computing, perception, and control. To bring the theoretical knowledge of task rescheduling closer to reality, it is necessary to represent the uncertain ECN parameters as fuzzy variables. There have been many related studies in this area. Fan et al. [29] mapped the channel assignment problem into a fuzzy-logic space, and employed a triangular fuzzy number (TFN) to describe the uncertain channel power gains. Li et al. [30] designed a nonlinear exponential function to integrate multiple heterogeneous conflicting criteria such as fuzzy time.

The scenario-based uncertainty of task scheduling in UAV emergency communication is that the original schedule has to be updated after being corrupted by unpredictable factors, such as the arrival of tasks and the destruction of ECNs. Wang et al. [31] modeled the task rescheduling problem as a dynamic matching problem in an environment where UAVs and tasks arrive stochastically. A multiple-waitlist-based task assignment was proposed to ensure dynamic stability. Considering the uncertain distribution of the transmission numbers and generated data, Chen et al. [32] formulated a robust two-stage stochastic optimization problem for delay minimization. Ngoenriang et al. [33] discussed UAV placement and data delivery schemes under uncertainties of inspection requests, the urgency of reports, and the availability of communication channels. However, the optimization objectives of these literatures tend to focus on scheduling performance, rather than maximizing the rescue efficiency of emergency scenarios. Hence, how to optimize the task rescheduling performance under uncertainty in the UAV emergency network is a significant topic and remains semi-open.

III. Modeling and Problem Formulation

This article studies the task rescheduling problem in uncertainty-aware UAV emergency networks. We consider two uncertainties: inherent parameter uncertainty and scenario-based uncertainty, where the latter includes task insertion and ECN destruction. This section first introduces the research motivation from the scenario requirements, then models uncertain rescheduling problems. Finally, the optimization problem is formulated. Notations used in this paper are given in Table I.

A. Motivation Scenario

We consider a post-disaster (earthquake, fire, etc.) rescue scenario, where obstacles such as mountains prevent rescue equipment from easily approaching the disaster area. The infrastructure is paralyzed without power and backup battery, and the service requests from those trapped cannot be fulfilled. UAVs become an effective means of assisting rescue, and the set of available UAVs is denoted as $\mathbb{U} = \{u_1, u_2, \cdots, u_U\}$. To ensure that the trapped users communicate with the workable infrastructure outside the disaster area, the UAVs can be used as mobile aerial base stations to establish a temporary communication connection, and transmit the rescue request and the monitored information to the emergency command center for better arrangements. In addition, ECNs can be regarded as edge servers, not only performing computing requests, but more importantly, analyzing the load-aware disaster situation. Assume that there are L ground devices (including those held by trapped humans and sensing devices), denoted as $\mathbb{L} = \{l_1, l_2, \cdots, l_L\}$. Suppose the number of ECNs

TABLE I
LIST OF KEY NOTATIONS

Notations	Meaning
$\mathbb{U}$, $\mathbb{L}$	Set of U UAVs and L ground devices
$\mathbb{M}$	Set of M edge computing nodes (ECNs)
$\mathbb{N}$, N^i	Set of $\lvert N\rvert$ tasks, i -th task
m_k , n_j^i	k -th ECN, j -th subtask of the i -th task
$\tilde{Ts}_j^i$, Ts_j^i	Fuzzy /real start time of subtask n_j^i
$\tilde{Tf}_j^i$, Tf_j^i	Fuzzy /real finish time of subtask n_j^i
$\tilde{Te}_{j,m_k}^i$, Te_{j,m_k}^i	Fuzzy /real execution time of subtask n_j^i on the ECN m_k
$\tilde{Rr}$, Rr	Fuzzy /real rescheduling time
$\tilde{Rs}_j^i$, $\tilde{Re}_{j,m_k}^i$ $\tilde{Rf}_j^i$,	Rescheduling start/ execution/ finish time of subtask n_j^i
$\tilde{Tb}_j^i$, $\tilde{Tw}_j^i$,	Released time, waiting time of subtask n_j^i
Ω_j^i , $\tilde{Ts}_{\max j}^{\ i}$	Tolerable maximum time, latest start time.
a_{i,j,m_k}	Offloading indicator to represent if subtask n_j^i is offloaded to the ECN m_k
c_j^i	Number of CPU cycles are required to execute subtask n_j^i
ρ_k	Computing capability of ECN m_k .
y_j^i	The stage phase function value of subtask n_j^i
$\tilde{Fgm}_j^i$, Fgm_j^i	Fuzzy/real fine-grained makespan
$\mathbb{NJ}_{\mathbb{I}}$	Set of subtasks that has started to be executed but not finished
$\mathbb{NH}_{\mathbb{I}}$	Set of subtasks that has not been executed
$\mathbb{NG}_{\mathbb{I}}$	Set of subtasks to be inserted
$\mathbb{D}$	Set of failed ECNs
$\mathbb{NJ}_D$	Set of subtasks assigned to $\mathbb{D}$ in the original scheduling
$\mathbb{NG}_D$	Set of subtasks assigned to workable ECNs $\mathbb{M} \setminus \mathbb{D}$

is $M = U + L$, denoted as $\mathbb{M} = \{m_1, m_2, \cdots, m_M\}$.$\mathbb{N} = \{N^1, N^2, \cdots, N^{|N|}\}$ is the set of $|N|$ tasks and $N^i = \left\{n_1^i, n_2^i, \cdots, n_{|N^i|}^i\right\}$, where $N^i \in \mathbb{N}$ represents the set of subtasks of the i-th task.

Figure 1(a) shows an example of the original scheduling process of 4 tasks (including 8 subtasks n_1^1-n_4^4). The original task list contains the task $N^1 - N^4$, where the tasks N^3 and N^4 consist of dependent subtasks. The subtask n_1^1 is assigned to ECN m_3. The subtask n_1^2 is assigned to ECN m_2. The dependent subtasks n_1^3 and n_2^3 are computed by m_4 and m_5, respectively. The subtask n_2^3 can only be executed when m_5 receives the computing result from m_4, which is the dependency constraint [34].

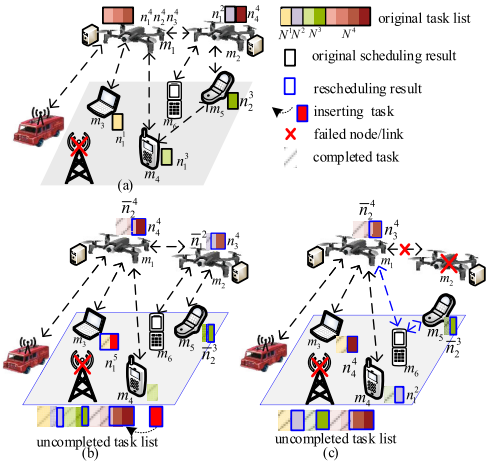


Fig. 1. The system model of the UAV emergency network.

Figure 1(b) and Figure 1(c) show the process of task rescheduling when the original scheduling is interfered with task insertion and ECN destruction, respectively. $\bar{n}_j^i$ indicates that the subtask n_j^i is being processed but not completed when the rescheduling is triggered (the moment called rescheduling time Rr). In Fig. 1(b), due to real-time changes in the actual demands, the task N^5 (including the subtask n_1^5), is inserted at Rr and assigned to ECN m_3. The subtasks n_3^4 and n_4^4 that have not been started yet in the original scheduling need to be rescheduled, and the subtasks $\bar{n}_1^2$, $\bar{n}_2^3$ and $\bar{n}_2^4$ continue to be executed on the originally assigned ECNs. In Fig. 1(c), ECN m_2 breaks down as a result of insufficient power or external factors, and the uncompleted subtasks n_1^2 and n_4^4 on m_2 need to be reassigned. The subtasks $\bar{n}_2^3$ and $\bar{n}_2^4$ on unaffected ECNs m_1 and m_5 continue to be executed.

B. Task Rescheduling Model With Fuzzy Number

Task rescheduling is a response to uncertainty, including inherent parameter uncertainty and scenario-based uncertainty. The former is characterized by the fluctuation of state parameters, and we model the fluctuant parameters as TFNs to enhance the robustness and applicability. The latter is characterized by the tasks being inserted and the ECNs being destroyed at any time. We discuss scenario-based uncertain rescheduling models from the rescheduling time, tasks list, and ECNs list.

1) Fuzzy Scheduling Model: In the original scheduling, the start time Ts_j^i is represented by TFN as $\tilde{Ts}_j^i = \left(\overset{\min}{Ts_j^i}, \overset{most}{Ts_j^i}, \overset{\max}{Ts_j^i}\right)$ of the subtask n_j^i, where $\overset{\min}{Ts_j^i}, \overset{most}{Ts_j^i}, \overset{\max}{Ts_j^i}$ are the best, probable, and worst start time, respectively. Similarly, the execution time Te_{j,m_k}^i and the finish time Tf_j^i are fuzzed with TFN.

Suppose there are two subtasks n_1^i and n_2^i, and their finish times are represented by TFN as $\tilde{T}f_1^i = \left(\overset{\min}{Tf_1^i}, \overset{most}{Tf_1^i}, \overset{\max}{Tf_1^i}\right)$ and $\tilde{T}f_2^i = \left(\overset{\min}{Tf_2^i}, \overset{most}{Tf_2^i}, \overset{\max}{Tf_2^i}\right)$, respectively. The operations

to be used are addition, subtraction, division, maximum, and ranking. These can be defined as follows [35]:

$$\tilde{T}f_1^i + \tilde{T}f_2^i = (\overset{\min}{Tf_1^i} + \overset{\min}{Tf_2^i}, \overset{\text{most}}{Tf_1^i} + \overset{\text{most}}{Tf_2^i}, \overset{\max}{Tf_1^i} + \overset{\max}{Tf_2^i}), \tag{1}$$

$$\tilde{T}f_1^i - \tilde{T}f_2^i = (\overset{\min}{Tf_1^i} - \overset{\max}{Tf_2^i}, \overset{\text{most}}{Tf_1^i} - \overset{\text{most}}{Tf_2^i}, \overset{\max}{Tf_1^i} - \overset{\min}{Tf_2^i}), \tag{2}$$

$$\tilde{T}f_1^i \div \tilde{T}f_2^i = (\overset{\min}{Tf_1^i} \div \overset{\max}{Tf_2^i}, \overset{\text{most}}{Tf_1^i} \div \overset{\text{most}}{Tf_2^i}, \overset{\max}{Tf_1^i} \div \overset{\min}{Tf_2^i}), \tag{3}$$

$$\max\{\tilde{T}f_1^i + \tilde{T}f_2^i\} = (\max\{\overset{\min}{Tf_1^i}, \overset{\min}{Tf_2^i}\}, \max\{\overset{\text{most}}{Tf_1^i}, \overset{\text{most}}{Tf_2^i}\}, \max\{\overset{\max}{Tf_1^i}, \overset{\max}{Tf_2^i}\}). \tag{4}$$

In addition, the defuzzification operation can be expressed as

$$Tf_1^i = \left(\overset{\min}{Tf_1^i} + 2 \times \overset{\text{most}}{Tf_1^i} + \overset{\max}{Tf_1^i}\right)/4. \tag{5}$$

Let Tf_1^i and Tf_2^i be the defuzzification of $\tilde{T}f_1^i$ and $\tilde{T}f_2^i$ according to (5). The ranking operation of $\tilde{T}f_1^i$ and $\tilde{T}f_2^i$ is as follows:

Rule 1: If $Tf_1^i > Tf_2^i$, then $\tilde{T}f_1^i \vee \tilde{T}f_2^i = \tilde{T}f_1^i$; else if $Tf_1^i < Tf_2^i$, then $\tilde{T}f_1^i \vee \tilde{T}f_2^i = \tilde{T}f_2^i$.

Rule 2: When $Tf_1^i = Tf_2^i$, If $\overset{\text{most}}{Tf_1^i} > \overset{\text{most}}{Tf_2^i}$, then $\tilde{T}f_1^i \vee \tilde{T}f_2^i = \tilde{T}f_1^i$; else if $\overset{\text{most}}{Tf_1^i} < \overset{\text{most}}{Tf_2^i}$, then $\tilde{T}f_1^i \vee \tilde{T}f_2^i = \tilde{T}f_2^i$.

Rule 3: When $\overset{\text{most}}{Tf_1^i} = \overset{\text{most}}{Tf_2^i}$, if $\left(\overset{\max}{Tf_1^i} - \overset{\min}{Tf_1^i}\right) > \left(\overset{\max}{Tf_2^i} - \overset{\min}{Tf_2^i}\right)$, then $\tilde{T}f_1^i \vee \tilde{T}f_2^i = \tilde{T}f_1^i$; else $\tilde{T}f_1^i \vee \tilde{T}f_2^i = \tilde{T}f_2^i$.

In the above operation, if the previous rules do not yield a conclusion, the latter rule continues to check until Rule 3.

2) Task Insertion Rescheduling Model: In the UAV emergency network where the demand changes in real-time, the original scheduling decision may update when tasks are inserted. We define the time of task insertion as fuzzy rescheduling time $\tilde{R}_r$. In the rescheduling process, the fuzzy start time, the fuzzy execution time, and the fuzzy finish time of subtask n_j^i are expressed as $\tilde{R}s_j^i$, $\tilde{R}e_{j,m_k}^i$, and $\tilde{R}f_j^i$, respectively. If subtask n_j^i is scheduled to ECN m_k, then the scheduling decision vector $a_{i,j,m_k} = 1$, else $a_{i,j,m_k} = 0$.

According to the principle of whether the subtask starts to be executed or completed at $\tilde{R}_r$, the state of subtasks can be divided into four cases as below. It should be noted that, to provide a clearer explanation of the rescheduling model, we assume that all tasks will be offloaded.

Case 1: The subtasks that have been completed. These are not considered in our work since they do not contribute to the rescheduling process.

Case 2: The set of subtasks that has started to be executed but not finished is defined as $\mathbb{NJ}_\mathbb{I}$. The subtask $n_j^i \in \mathbb{NJ}_\mathbb{I}$, $1 \le j \le |NJ_I|$, satisfies the constraint as:

$$\tilde{T}s_j^i + \tilde{T}e_{j,m_k}^i \ge \tilde{R}r, \forall n_j^i \in \mathbb{NJ}_\mathbb{I}, m_k \in \mathbb{M}, \tag{6}$$

where $|NJ_I|$ denotes the number of subtasks being executed.

During the rescheduling process, $n_j^i \in \mathbb{NJ}_\mathbb{I}$ will not change the assigned ECN m_k, which can be formalized as:

$$\tilde{T}s_j^i \le \tilde{R}r, \forall a_{i,j,m_k} = 1, n_j^i \in \mathbb{NJ}_\mathbb{I}, .m_k \in \mathbb{M}. \tag{7}$$

Case 3: The set of subtasks that has not been executed is defined as $\mathbb{NH}_\mathbb{I}$. The subtask $n_h^i \in \mathbb{NH}_\mathbb{I}, 1 \le h \le |NH_I|$ satisfies the constraint as:

$$\tilde{T}s_h^i \ge \tilde{R}r, \forall n_h^i \in \mathbb{NH}_\mathbb{I}, \tag{8}$$

where $|NH_I|$ is the number of subtasks that have not yet started when the rescheduling is triggered. The scheduling results for n_h^i may change or remain unchanged. (9) models the process of decision changing from m_k to $m_{k'}$.

$$\begin{cases} a_{i,h,m_k} = 1 | \tilde{T}s_h^i \ge \tilde{R}r \\ a_{i,h,m_{k'}} = 1 | \tilde{R}s_h^i \ge \tilde{R}r \end{cases}, \forall n_h^i \in \mathbb{NH}_\mathbb{I}, \{m_k, m_{k'}\} \in \mathbb{M}, m_k \ne m_{k'}. \tag{9}$$

Case 4: The set of subtasks to be inserted is defined as $\mathbb{NG}_\mathbb{I}$. The subtask $n_g^i \in \mathbb{NG}_\mathbb{I}, 1 \le g \le |NG_I|$ satisfies the constraint as:

$$\tilde{R}s_g^i \ge \tilde{R}r, \forall n_g^i \in \mathbb{NG}_\mathbb{I}, \tag{10}$$

where $|NG_I|$ is the number of inserted subtasks.

If subtask n_g^i is assigned to ECN m_k, whose task queue contains the uncompleted subtask n_j^i from the original scheduling, then subtask n_g^i must wait until n_j^i is finished before it can start, which can be expressed as:

$$a_{i,j,m_k} = 1, a_{i,g,m_k} = 1 | \tilde{R}s_g^i \ge \tilde{T}f_j^i, \forall n_j^i \in \mathbb{NJ}_\mathbb{I}, n_g^i \in \mathbb{NG}_\mathbb{I}. \tag{11}$$

(11) also indicates that the execution of tasks on an ECN is sequential.

3) ECN Destruction Rescheduling Model: In a dynamic emergency network, interruptions or unpredictable events are inevitable. The representative uncertain event is that ECNs are destroyed, which leads to the disruption of the original optimal plan, and even becomes unfeasible. Therefore, it is necessary to adjust the original scheduling strategy to respond to uncertain events, so as to shorten the reduction of rescue efficiency caused by the failed ECNs. We assume that the number of failed ECNs is $|D|$, and the set of failed ECNs is expressed as $\mathbb{D}$. Thus, the set of $M - |D|$ workable ECNs can be expressed as $\mathbb{M} \backslash \mathbb{D}$. We analyze the status of the task according to the principle of whether the ECNs are destroyed as follows:

Case 1: The set of subtasks assigned to $\mathbb{D}$ in the original scheduling is denoted as $\mathbb{NJ}_D$, which contains $|NJ_D|$ subtasks. If $n_j^i \in \mathbb{NJ}_D, 1 \le j \le |NJ_D|$ is assigned to m_k, as long as n_j^i is not completed, it must be reassigned to a workable ECN $m_{k'}$, which can be formulated as (12) and (13). The subtask n_j^i in (12) has been executed but not completed in the original

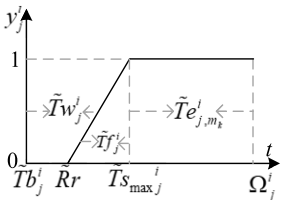


Fig. 2. The stage phase function.

scheduling. The subtask n_h^i in (13) has not been executed until $\tilde{R}r$.

$$a_{i,j,m_k} = 1, a_{i,j,m_{k'}} = 1 | \tilde{T}f_j^i \geq \tilde{R}r, \forall n_j^i \in \mathbb{NJ}_{\mathbb{D}}, m_k \in \mathbb{D}, m_{k'} \in \{\mathbb{M}\backslash\mathbb{D}\}, \tag{12}$$

$$a_{i,h,m_k} = 1, a_{i,h,m_{k'}} = 1 | \tilde{R}s_h^i \geq \tilde{R}r, \forall n_h^i \in \mathbb{NJ}_{\mathbb{D}}, m_k \in \mathbb{D}, m_{k'} \in \{\mathbb{M}\backslash\mathbb{D}\}, \tag{13}$$

where m_k and $m_{k'}$ satisfy $1 \leq k \leq |D|$ and $1 \leq k' \leq M - |D|$, respectively.

Case 2: The set of subtasks assigned to workable ECNs $\mathbb{M}\backslash\mathbb{D}$ can be denoted as $\mathbb{NG}_D$, which contains $|NG_D|$ subtasks. At $\tilde{R}r$ as shown in (14), if $n_h^i \in \mathbb{NG}_{\mathbb{D}}, 1 \leq h \leq |NG_D|$ has started to be executed but is not completed, then it will continue to execute without changing the ECN. If $n_g^i \in \mathbb{NG}_{\mathbb{D}}$, $1 \leq g \leq |NG_D|$ has not started to be executed, its ECN m_k, $\forall 1 \leq k \leq |D|$ may be replaced by another ECN $m_{k'}$, $\forall 1 \leq k' \leq M - |D|$, which is formulated as (15):

$$a_{i,g,m_k} = 1, \forall \tilde{T}s_g^i \leq \tilde{R}r, \tilde{T}f_g^i \geq \tilde{R}r, n_g^i \in \mathbb{NG}_{\mathbb{D}}, m_k \in \{\mathbb{M}\backslash\mathbb{D}\}, \tag{14}$$

$$\begin{cases} a_{i,g,m_k} = 1 | \tilde{T}s_h^i \geq \tilde{R}r \\ a_{i,g,m_{k'}} = 1 | \tilde{R}s_h^i \geq \tilde{R}r \end{cases}, \forall n_g^i \in \mathbb{NG}_{\mathbb{D}}, m_k \in \{\mathbb{M}\backslash\mathbb{D}\}, m_{k'} \in \{\mathbb{M}\backslash\mathbb{D}\backslash m_k\}. \tag{15}$$

In the above task rescheduling model, the index of subtasks n_j^i, n_h^i, n_g^i satisfy $1 \leq i \leq |N|$.

C. Problem Formulation

This section first describes the definition of fine-grained makespan, and then formulates an uncertainty-aware subtask rescheduling problem.

1) Fine-Grained Makespan: The original priority of subtasks represents different competitiveness, and the subtasks with weaker competitiveness will always be delayed, even beyond the deadline. Therefore, we design a SPF that characterizes the real-time state of the subtask n_j^i, which considers the incoming requests and observed states. The former is determined by the size of input data a_j^i, the number of required CPU cycles c_j^i, the released time $\tilde{T}b_j^i$, and the tolerable maximum time Ω_j^i. The latter is determined by the waiting time $\tilde{T}w_j^i$, and the assigned ECN m_k. The change of SPF from released time to deadline is described in Figure 2, where $\tilde{T}s_{\max j}^i$ represents the latest start time. SPT can be formalized as:

$$y_j^i = \frac{\tilde{T}s_j^i - \tilde{R}r}{\Omega_j^i - \tilde{T}e_{j,m_k}^i - \tilde{R}r}, \forall 1 \leq i \leq |N|, 1 \leq j \leq \left|N^i\right|, 1 \leq k \leq M, \tag{16}$$

where the execution time $\tilde{T}e_{j,m_k}^i$ of the subtask n_j^i in m_k is calculated by $\tilde{T}e_{j,m_k}^i = c_j^i \Big/ \rho_k$, and ρ_k represents the computing capability of ECN m_k.

In order to optimize the task rescheduling performance while guaranteeing the utilization of ECNs, we define an evaluation indicator, called fine-grained makespan ($\tilde{F}gm_j^i$), which can be calculated as follows:

$$\tilde{F}gm_j^i = \frac{y_j^i * \tilde{T}f_j^i}{\tilde{T}s_j^i - \tilde{T}f_h^i}, \forall 1 \leq i \leq |N|, 1 \leq j, h \leq \left|N^i\right| \tag{17}$$

where $y_j^i * \tilde{T}f_j^i$ indicates the degree of satisfaction with subtasks. Suppose that subtasks n_j^i and n_h^i are continuously executed on ECN m_k. Then $\tilde{T}s_j^i - \tilde{T}f_h^i$ is the gap time between two subtasks, and meets the following constraints:

$$\tilde{T}s_j^i > \tilde{T}f_h^i, \forall a_{i,j,m_k} = 1, a_{i,h,m_k} = 1, \left\{n_j^i, n_h^i\right\} \in \mathbb{N}, 1 \leq i \leq |N|, 1 \leq j, h \leq \left|N^i\right|, 1 \leq k \leq M. \tag{18}$$

(18) means that the predecessor subtask of n_j^i is n_h^i in the subtask queue of m_k.

2) Objective Function: Uncertainty-aware task rescheduling problem is the process of integrating uncertain factors into assigning $|N|$ tasks to M ECNs. The uncertain factors include task insertion, ECN destruction, and parameter fluctuation. The goal of rescheduling is to minimize the fine-grained makespan, which maps the optimization of makespan and ECNs' utilization. The mathematical model is formulated as follows:

$$\begin{aligned}
&\min \sum_{i=1}^{|N|} \sum_{j=1}^{|N^i|} \tilde{F}gm_j^i, \\
&\text{Subject to:} \quad C1: (6) - (15), \\
&\qquad C2: \sum_{k=1}^{M} a_{i,j,m_k} \leq 1, \\
&\qquad \forall m_k \in \mathbb{M}, 1 \leq i \leq |N|, 1 \leq j \leq \left|N^i\right|, \\
&\qquad C3: \sum_{i=1}^{|N|} \sum_{j=1}^{|N^i|} a_{i,j,m_k} \leq 1, \\
&\qquad \forall \mathbb{N}^i \in \mathbb{N}, n_j^i \in \mathbb{N}^i, 1 \leq k \leq M, \\
&\qquad C4: a_{i,j,m_k} \in [0,1], \\
&\qquad \forall \mathbb{N}^i \in \mathbb{N}, n_j^i \in \mathbb{N}^i, m_k \in \mathbb{M}, \\
&\qquad 1 \leq i \leq |N|, 1 \leq j \leq \left|N^i\right|, 1 \leq k \leq M \\
&\qquad C5: \sum_{k=1}^{M} Te_{j,m_k}^i \leq \tilde{T}^{\max}, \forall n_j^i \in \mathbb{N}^i, m_k \in \mathbb{M}, \\
&\qquad 1 \leq i \leq |N|, 1 \leq j \leq \left|N^i\right|, 1 \leq k \leq M,
\end{aligned} \tag{19}$$

where $C1$ are constraints when uncertain events occur. $C2$ indicates that each subtask can only be executed by one ECN. $C3$ ensures that each ECN can only execute one subtask at the same time. $C4$ is the binary constraint of decision vector a_{i,j,m_k}.$C5$ indicates that the service time of the ECN cannot exceed the maximum workable time $\tilde{T}^{\max}$.

IV. Proposed Algorithm for Task Rescheduling

In this section, for an uncertain UAV emergency communication system, we develop an asynchronous shuffled frog-leaping with feasible Jaya algorithm to solve the task rescheduling problem, aiming to minimize the fine-grained makespan. The ASFJ integrates the simplicity and efficiency of the Jaya algorithm into the asynchronous shuffled frog-leaping algorithm. In this section, two preliminary algorithms are presented first. Then the ASFJ is introduced.

A. Preliminaries

1) Jaya Algorithm: Jaya algorithm is a new meta-heuristic algorithm. First of all, Jaya algorithm forms the initial population randomly, then evolves in the principle of being close to the optimal individual and away from the worst individual. Moreover, the involved parameters are only population size and the number of iterations, who are not specific to the performance of the Jaya algorithm. Hence, the Jaya algorithm is a simple, effective optimization tool, and the new solution can be updated as follows [36]:

$$o_i' = o_i + r_1 * (o_B - |o_i|) - r_2 * (o_W - |o_i|), \tag{20}$$

where o_i and o_i' represent the current solution and the new solution of the i-th individual, respectively. o_B and o_W are the best and worst solutions. r_1 and r_2 are random values in the range of [0, 1], respectively. The term $o_B - |o_i|$ indicates that the candidate solution is closer to o_B and the term $-(o_W - |o_i|)$ indicates that it is farther away from o_W.

2) The Shuffled Frog-Leaping Algorithm: SFLA is a novel bionic swarm intelligence algorithm, that imitates the cooperative behavior of frogs in finding the locations with the most food. It combines the advantages of the genetic-based meme algorithm and the social behavior-based PSO algorithm. Through the information interaction between different individuals (solutions) and different subgroups (memeplexes), the global search performance can be improved, which can be divided into the following four stages:

a) Initializing the population stage: Generate a population P_{O*Q} consisting of O solutions (frogs). Q is the dimension of the solution, and each solution o_i is expressed as $o_i = \{o_{i,1}, o_{i,2} \cdots, o_{i,Q}\}$.

b) Sorting and grouping stage: Calculate the fitness values of the O solutions and arrange them in ascending order. Divide the population P_{O*Q} into S memeplexes, where each memeplex contains O/S solutions. In the process of grouping, the first solution is divided into the 1st memeplex, the S-th solution is divided into the S-th memeplex, and the $S+1$-th solution is also divided into the 1st memeplex. Therefore, the division rule of memeplexes is:

$$s = o\%S, \ \forall o = 1, 2, \cdots, O\,, \tag{21}$$

where s is the index of the s-th memeplex.

c) Local search stage: First, the global optimal solution o_G, the local optimal solution o_B and the worst solution o_W are obtained. Second, at each iteration of the memeplex s, the same method as PSO is used to improve the quality of the worst solution o_W. That is to say, according to (22), o_W is calculated to obtain the updated solution o_W', and if o_W' is smaller than o_W, then o_W will be replaced by o_W'; otherwise, o_B in (22) will be replaced by the global optimal solution o_G and then continue to calculate and compare in the same way. If the worst solution o_W is still not improved, a new worst solution o_W' will be randomly generated in the search space.

$$o_W' = o_W + rand(0, 1) * (o_B - o_W). \tag{22}$$

d) Global information interaction stage: All solutions are reshuffled and reordered according to step b) when memeplexes evolve to a certain degree. Then, the new population is re-divided into multiple new memeplexes, which enables a global exchange of local information between memeplexes. The local search stage and the global information interaction stage are run repeatedly until the convergence condition is satisfied.

All in all, although Jaya algorithm is simple and efficient, it has the problems of premature convergence and insufficient search ability when the population size is larger. Besides, even if the SFLA has good convergence, the fixed size of memeplexes makes the algorithm easy to fall into the local optimum, and it is difficult to obtain an accurate solution in the local search stage. Therefore, we propose an ASFJ algorithm, which is a discrete optimization algorithm with: 1) a heuristic population initialization method; 2) an asynchronous shuffled frog-leaping method; 3) two feasible local search operators.

B. The Asynchronous Shuffled Frog-Leaping With Feasible Jaya Algorithm

To solve the task rescheduling problem effectively in a UAV emergency network, we develop the ASFJ algorithm. Firstly, the heuristic population initialization method (Algorithm 1) is presented, as well as the energy consumption and execution time guide to develop a subtask sequence vector and an ECN assignment vector. Then, the asynchronous shuffled frog-leaping method (Algorithm 2) divides the population into multiple memeplexes that evolve independently. The number of memeplex stagnations is an indicator for judging whether to process local search or information interaction through fusion memeplexes. This asynchronous manner enables the evolution of memeplexes to avoid the information overlap caused by the forced shuffling of the benchmark SFLA. Finally, for the local search of each memeplex, the feasible Jaya-based local search operator and the absorbing and swapping operator are designed, and the global optimal solution, local optimal solution, and local worst solution are utilized to improve the quality of the solution and ensure that the formed solution is feasible. On one hand, the advantages of simplicity and efficiency of Jaya make the proposed ASFJ improve the local search ability while controlling the complexity. On the other hand, as a local search operator, it can appropriately alleviate the limitations of the Jaya in large-scale instances. Figure 3 shows a flowchart of the ASFJ.

Algorithm 1 The heuristic population initialization method

Input: the population size O, the set of tasks $\mathbb{N}$ with $|N| * \left|N^i\right|$ subtasks, and the set of available ECNs $\mathbb{M}$.
Output:initial population P_{O*Q}

%Encoding representation
generating subtask sequence vector SR and ECN assignment vector CA
for $o = 1 : O$ **do**
  % Initialize SR
  for $i = 1 : |N|$ **do**
    for $j = 1 : \left|N^i\right|$ **do**
      $SR(1, j) = i$
    end for
  end for
  shuffling SR
  % Initialize CA
  if $0 < o \leq 0.2 * O$ %Rule 1
    for $j = 0.5 * Q + 1 : Q, \forall j \in SR$ **do**
      $m_k = \arg\min_{m_k \in \mathbb{M}} Te^i_{j,m}$
    end for
  end if
  if $0.2 * O < o \leq 0.4 * O$ %Rule 2
  for $j = 0.5 * Q + 1 : Q, \forall j \in SR$ **do**
    $m_k = \arg\min_{m_k \in \mathbb{M}} W^i_{j,m}$
    end for
  end if
  if $0.4 * O < o \leq O$ %Rule 3
    for $j = 0.5 * Q + 1 : Q, \forall j \in SR$ **do**
      $m_k = random(m_k \in \mathbb{M})$
    end for
  end if
end for

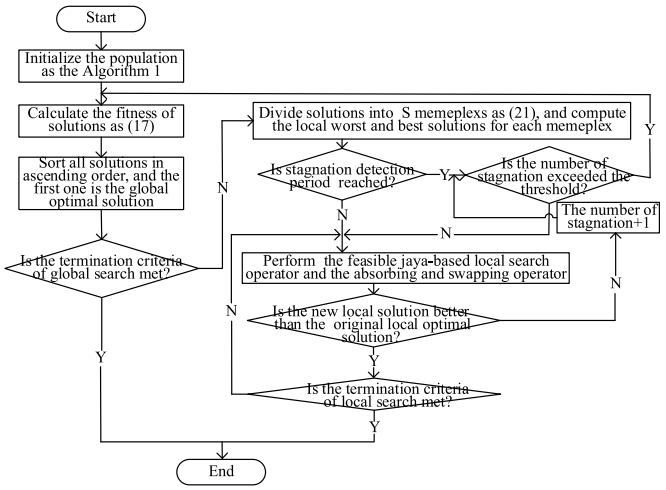


Fig. 3. The flowchart of ASFJ.

1) Encoding, Decoding and Initialization:

a) The encoding and decoding scheme: The purpose of the encoding and decoding scheme of the task rescheduling problem is to optimize the subtask sequence vector and ECN assignment vector. The subtask sequence vector represents the execution order of all subtasks, and its length denotes the number of subtasks. We express the sequence vector of $|N|$ tasks as $SR = \left\{1, \cdots, i_j, \cdots, |N|_{|N|*|N^i|}\right\}$, where $i = 1, \cdots, |N|$ expresses $|N|$ tasks, $j = 1, \cdots, \left|N^i\right|$ expresses

Algorithm 2 The asynchronous shuffled frog-leaping method

Input: the population P_{O*Q}, the number of population iterations T, the number of memeplex iterations K, stagnation detection period st_{period}.
Initialization:$|S| = O/2, st_{num} = 0, O\_me = 2, st_{period} = K/O\_me$.
Output:the optimal solution $o_{optimal}$ with SR and CA

while $t \leq T$
  % Global Exploration
  for $o = 1 : O$ **do**
    compute fine-grained makespan $\tilde{F}gm^i_j$ as (17)
  end for
  sort all solutions in ascending order of $\tilde{F}gm^i_j$, and divide them into S memeplexes as (21)
  find the global optimum o_G with $\min(\tilde{F}gm^i_j)$
  % Local exploitation parallelly
  for $s = 1 : S$ **do**
    for $k = 1 : K$ **do**
      perform a local search as Section IV-B-III, find the local best solution o_B and worst solution o_W
      if $k = st_{period}$ and $st_{num} \geq O\_me/2$
        merge $s \in s\_st$
      end if
    end for
  end for
end while

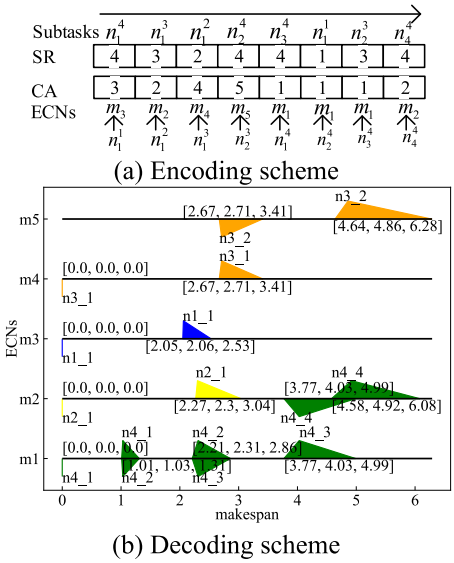


Fig. 4. Example for encoding and decoding scheme.

the number of occurrences of the task N_i and also the subtask n^i_j. The ECN assignment vector represents the index of ECNs that executes each subtask sequentially, which is denoted as $CA = \{1, \cdots, m, \cdots, M\}$.

Figure 4(a) shows the two encoding vectors of the 5 ECNs and 4 tasks in Fig. 1(a). $SR = \{4, 3, 2, 4, 4, 1, 3, 4\}$ represents that the execution sequence of the subtask is $n^4_1 \rightarrow n^3_1 \rightarrow n^2_1 \rightarrow n^4_2 \rightarrow n^4_3 \rightarrow n^1_1 \rightarrow n^3_2 \rightarrow n^4_4$. $CA = \{3, 2, 4, 5, 1, 1, 1, 2\}$ denotes that the subtask n^1_1 is executed at ECN m_3, the subtask n^2_1 is assigned to ECN m_2, and so on.

This paper utilizes an active decoding scheme to minimize the maximum fine-grained makespan, which has been proven to be feasible [37]. The scheduling decision of the subtask to

be executed on the ECN with the earliest fine-grained start time is made by comparing two parameters, which are the fine-grained makespan of the dependent predecessor and the predecessor assigned to the same ECN. The decoding result is shown using a Gantt chart in Fig. 4(b). The TFNs below and above the line denote the start time and completed time, respectively.

b) The heuristic population initialization method: The makespan of all tasks is expressed as the time gap between the start time of the first subtask and the completed time of the last subtask, and the execution time of each subtask is an important factor affecting the makespan. Energy consumption determines the utilization of ECNs. Therefore, the generation of initial solutions is guided by the principles of minimizing the execution time and energy consumption. To ensure the diversity of solutions, it is necessary to generate feasible initial solutions randomly.

To sum up, we propose a heuristic population initialization method guided by energy consumption and execution time to allocate ECNs initially. The details are given in Algorithm 1. The rules are described as follows:

Rule 1: For each subtask n_j^i, globally search for the ECN m_k with the shortest execution time Te_{j,m_k}^i.

Rule 2: For each subtask n_j^i, globally search for the ECN m_k with the lowest energy consumption W_{j,m_k}^i.

Rule 3: Randomly assign subtasks to available ECNs.

The population is formed according to the proportion of 20% for each of rule 1 and rule 2, and the proportion of 60% for rule 3, which ensures that the solution has a high quality, and also guarantees the diversity of the population, preventing similar individuals from reducing the search efficiency of the algorithm.

2) The Asynchronous Shuffled Frog-Leaping Method: The main idea of ASFJ is that the feasible Jaya algorithm asynchronously evolves the memeplexes of the shuffled frog-leaping method. In the initial evolution stage, according to the quality of the fitness (fine-grained makespan), the population is divided into multiple memeplexes, which accelerates the global search ability and diversity. As the number of iterations increases, the memeplexes satisfying the stagnation threshold of the local optimal solution are adaptively shuffled, which makes the information interaction more frequent, representing the transition from global exploration to local exploitation. The pseudocode is shown in Algorithm 2.

The details of the asynchronous shuffled frog-leaping method are as follows:

Step 1: For the O solutions of the population P_{O*Q} generated in Section IV-B-I, we calculate the fine-grained makespan according to (17), and sort in ascending order.

Step 2: According to the rule of (21), divide P_{O*Q} into S memeplexes. In the first iteration, we set the initial number of memeplexes $|S|$ to be $O/2$ to make the global search more diverse.

Step 3: For the memeplex $s \in s\_st$, where $s\_st \subseteq S$ is the set of stagnant memeplexes, the current local optimal solution has been found according to Section IV-B-III. If the number of stagnations st_{num} is greater than $O\_me/2$, then merge

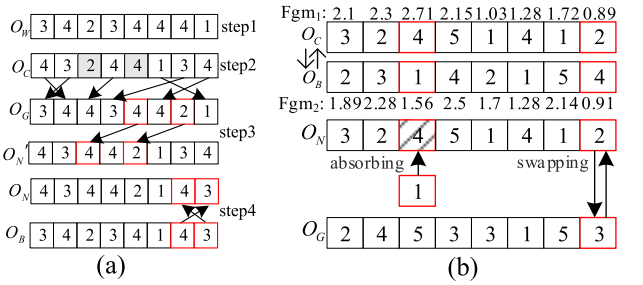


Fig. 5. Two local search operators (a) the feasible Jaya-based local search operator; (b) the absorbing and swapping operator.

memeplexes s_st until the number of memeplexes $|S| = 1$, where O_me is the number of solutions in the memeplex s; Otherwise, continue the local search. In general, the period of stagnation detection st_{period} is set as the ratio of the number of local iterations to the number of solutions within the memeplex.

Step 4: Continue to run step 1-step 3 until the size of S is equal to O, that is, all memeplexes are merged into one.

Following step 1-step 4, the adaptive scale of memeplexes avoids the assimilation of solutions. On the one hand, the solution evolves towards the local optimal solution of the memeplex. On the other hand, the solution is guided by the global optimal solution. Therefore, both global and local search capabilities are confirmed.

3) Two Feasible Local Search Operators: For memeplex $s \in S$, as the number of iterations increases, the population quality becomes higher and the local optimum tends to stagnate. In order to get a better local optimal solution, we design two feasible local search operators, including the feasible Jaya-based local search operator and the absorbing and swapping operator.

The feasible Jaya-based local search operator is proposed for the subtask sequence vector, and it draws on the advantages of the Jaya algorithm that are close to the global optimal solution and far from the local worst solution, and takes advantage of the local optimal solution to make the candidate solutions feasible. The optimization process is illustrated with an example in Figure 5(a). The details are as:

Step1: Obtain the global optimal solution o_G. For the memeplex s where the candidate solution o_C is located, obtain the local optimal solution o_B and the local worst solution o_W.

Step 2: Compare the candidate solution o_C with the local worst solution o_W and remove memes at the same position.

Step 3: Map the global optimal solution o_G to the empty position to form the new solution o'_N.

Step 4: Evaluate the feasibility of the new solution o'_N, for example, whether the dependency constraint is satisfied. For infeasible memes, the local optimal solution o_B is used to amend it to the feasible solution o_N. In Fig. 5(a), it is assumed that subtask n_2^3 and subtask n_4^4 are dependent, that is, n_2^3 must obtain the computing result of n_4^4 before it can be executed. Hence, o_N is not feasible, and must be adjusted.

Therefore, the solution obtained from Fig. 5(a) is $o_N = [4, 3, 4, 4, 2, 1, 3, 4]$. The proposed operator avoids the same information as the worst solution, inherits the information

of the global optimal solution, and effectively repairs the infeasible solution with the local optimal solution.

We present the absorbing and swapping operator for the ECN assignment vector, which is inspired by the principle of head-on collision [38]. After the collision, object 1 and object 2 may change their running direction based on their respective masses and speeds: i) Faster objects will continue to move in the original direction; ii) When the speed is similar, it is reflected in the opposite direction or static movement patterns.

We map the collision scenario described above to the task rescheduling problem, where candidate solution o_C is equivalent to object 1, local optimal solution o_B is equivalent to object 2, and the execution time of each meme in a solution is equivalent to mass. (17) and (5) calculate the Fgm_j^i of each meme, which is equivalent to speed. The following two cases may occur after the two solutions collide:

Case 1 (absorbing operator): If $Fgm_1 > Fgm_2$, indicating that the speed of meme $o_{C,i}$ in candidate solution o_C, is greater than that of meme $o_{B,i}$ in local optimal solution o_B, then keep moving in the original direction and move away from o_C. Subsequently, the vacant position absorbs meme $o_{B,i}$ to bring o_C closer to o_B. As shown in Fig.5 (b), in the collision crossover between o_C and o_B, task is assigned to ECNs m_4 and m_1, resulting in Fgm of 2.71 and 1.56, respectively. Consequently, the meme "4" with a larger Fgm moves away from o_C, and the vacant position absorbs meme "1" from the same position in the o_B. When $Fgm_1 < Fgm_2$, a similar approach is applied as in Case 1.

Case 2 (swapping operator): If $Fgm_1 \approx Fgm_2$, indicating that the meme $o_{C,i}$ and the meme $o_{B,i}$ have similar speeds, then they are reflected or become stationary, which indicates that the assignment decision is efficient and close to a local optimum. Furthermore, to escape the local optimum, we swap the meme $o_{G,i}$ at the same position as the global optimum with the meme $o_{C,i}$, which ensures that the global optimal solution o_G does not degenerate over iterations. In Fig. 5 (b), memes "2" and "4" exhibit similar speeds. We guide the evolution of the o_C by swapping meme "3" from the same position in the global optimal solution o_G with meme "2" in the o_C.

Therefore, the solution obtained from Fig. 5(b) is $o_N = [3, 2, 1, 5, 1, 4, 1, 3]$. In addition, feasibility detection is performed for the above two cases, and the infeasible meme is swapped by the randomly selected one from the list of workable ECNs.

V. Performance Analysis

In this section, we evaluate the efficiency and effectiveness of the proposed ASFJ algorithm in deterministic and uncertain UAV emergency networks. All algorithms are coded in Python 3.8.8 and run on a 1.6-GHz Intel Core i5 processor with 16 GB RAM.

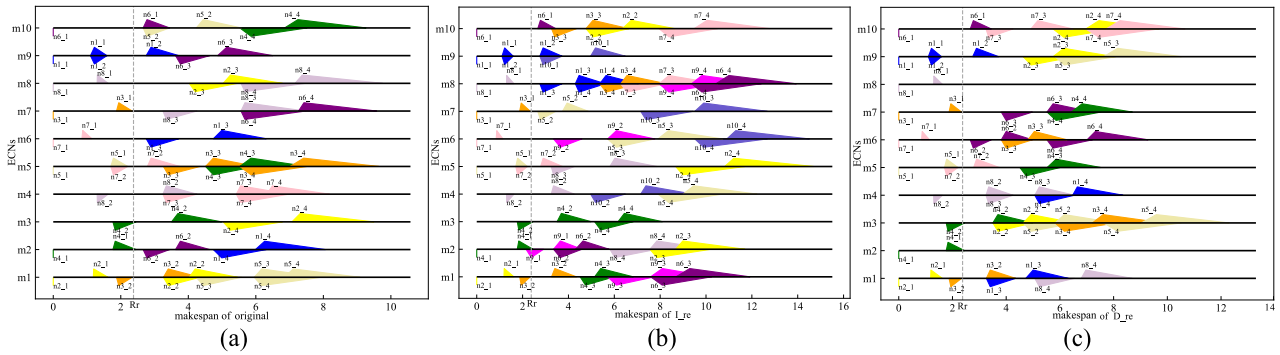
We assume that, with the help of the disaster command center, the current network status and task information are known at the beginning [34]. In the UAV emergency rescue scenario, UAVs and ground devices provide computing services in an area of 400 × 400 m^2, and the ratio of the number is 1:4. Each task has 4 dependent subtasks. The average time between the arrivals of two consecutive tasks follows an exponential distribution, with an average of 5 to 25 units of time. All subtasks are potentially processed by workable ECNs if the constraints are satisfied. UAVs serve as edge servers and have higher computing capacity than ground devices, which are 3.6*10^9Hz and 1*10^9Hz, respectively. The size of input data u_j is generated randomly within [2], [8] Mbit. We assume orthogonal frequency division multiplexing access is used. In the process of data transmission, the channel is orthogonal and there is no co-channel interference. For uncertainty, we consider two kinds of uncertain events, task insertion and ECN destruction. Assume that the number of inserted tasks is an integer randomly generated from [1, $|N|$] [39]. Similarly, the times at which ECNs are destroyed obey an exponential distribution, and the number of failed ECNs is an integer randomly generated from [1,1/5*M], where M is the number of ECNs. The parameter settings are listed in Table II.

TABLE II
Parameter Setting

Parameters	Values
Number of ECNs M at beginning	[10,50]
Number of tasks $\|N\|$ at beginning	[10,50]
Sub-carrier frequency b^c	2GHz
Channel bandwidth α	5MHz
Maximum power of UAVs (ground devices)	0.1W
Noise variance σ^2	1
Input data size u_j	[2-8] Mbit
Workload d_j	$1.2\,u_j$
Computing ability of UAVs	3.6×10^9 Hz
Computing ability of ground devices	1.0×10^8 Hz
The average value of exponential distribution between two successive new task arrivals	[5,25]

To verify the performance of ASFJ, we quantitatively compare ASFJ with three benchmark algorithms, which are briefly described as follows:

1) GA: GA is a classic evolutionary algorithm. Its encoding and decoding representation and population initialization method are the same as those in Section IV-B-I. The crossover and mutation rate are 0.9 and 0.5.
2) CF [3]: CF models the heterogeneous task allocation as a potential game between players and agents, and adopts a greedy strategy to search for a solution that maximizes the benefit within the limited budget. In order to ensure that ECN is available, we set the budget to 90% of the workable time of each ECN.
3) SFDE: SFDE is a discrete self-adaptive differential evolution algorithm with a shuffled frog-leaping strategy [17], which uses DE as the local search operator to perform unique mutation, crossover, and selection operations for each solution in different memeplexes. This experiment follows the same parameter setting as ASFJ.



(a) (b) (c)

Fig. 6. The Gantt chart of (a) original scheduling; (b) insertion rescheduling; (c) destruction rescheduling.

TABLE III

FGM VALUES AND MAKESPAN GAP BETWEEN PROPOSED ASFJ AND SFDE

Instances	Insertion				Destruction			
	ASFJ		SFDE		ASFJ		SFDE	
	best/avg(ms)	*gap(ms)*	*best/avg(ms)*	*gap(ms)*	*best/avg(ms)*	*gap(ms)*	*best/avg(ms)*	*gap(ms)*
8*4*10	0.362/0.428	2.577	0.392/0.450	3.320	0.425/0.489	1.735	0.478/0.532	1.812
16*4*10	0.417/0.478	2.941	0.521/0.620	3.243	0.460/0.521	2.983	0.536/0.623	5.492
8*4*20	0.354/0.407	1.865	0.412/0.488	2.880	0.337/0.375	0.351	0.408/0.459	0.879
16*4*20	0.205/0.251	1.253	0.363/0.423	2.205	0.219/0.255	0.770	0.320/0.358	1.099

The evaluation metrics include makespan, effectiveness, and fine-grained makespan, which can be described as:

1) Makespan represents the time gap from the beginning of the first subtask being executed to the end of the last one, which is the most intuitive and common evaluation index for task scheduling performance.
2) To evaluate the adaptability to uncertain events, we employ an evaluation metric that characterizes the repair ability of the algorithm, called effectiveness, which is calculated as:

$$eff = \sum_{m_k, m'_k=1}^{M} \frac{Rm'_k - Tm_k}{Tm_k}, \forall \{m_k, m_{k'}\} \in \mathbb{M}. \quad (23)$$

where $Rm'_k - Tm_k$ denotes the gap between the rescheduling makespan Rm'_k and the original scheduling makespan Tm_k on ECN m_k. The larger eff represents the greater proportion of the makespan gap to the original makespan, indicating that the algorithm has made greater efforts to repair the uncertainty and has greater repair ability.
3) As shown in (17), the fine-grained makespan is determined by the real-time status of the task, the makespan, and the ECNs' idle time. It is the fitness in the population iteration process.

Therefore, we evaluate the performance of AFSJ in deterministic original scheduling and uncertain rescheduling when the number of ECNs and tasks are fixed and varied. We set the number of individuals to be 100, the maximum global generation number T is 1000, and the maximum local generation number K is 500.

A. Performance With a Fixed Number of Tasks and ECNs

We evaluate the performance of the ASFJ with 10 ECNs and 8 tasks (including 32 subtasks). Figure 6(a)-(c) visualize the results of the original scheduling, insertion and destruction rescheduling, respectively. The horizontal axis is time, and the vertical axis is ECN. The same color represents different subtasks of the same task. ni_j denotes the j-th subtask of the i-th task. The area below the line represents the fuzzy start time of the subtask, and the area above represents the fuzzy finish time. Figure 6(b) exhibits rescheduling decisions after inserting tasks $n9$ and $n10$ (including 8 subtasks $n9\_1$-$n10\_4$) at rescheduling time Rr. It can be seen that the subtasks that have already started execution are not affected, e.g., subtask $n3\_2$. However, unstarted subtasks may be reassigned, e.g. the subtask $n1\_4$. Fig. 6(c) demonstrates the repair of the original scheduling after ECNs $m2$ and $m8$ were destroyed. In Fig. 6(c), all unfinished subtasks on the subtask queue of the damaged ECNs need to be reassigned, e.g., subtask $n6\_2$. Subtasks queued on non-destroyed ECNs are scheduled the same as insertion rescheduling.

Table III compares the minimum and average values of the fine-grained makespan ("best/avg") and the makespan gap ("*gap*") between rescheduling and original scheduling of ASFJ and SFDE for different problem instances. The problem sizes are the combination k*4*m of k tasks, and each task has 4 subtasks, m ECNs, e.g., 8 *4*10. We can see that the performance of ASFJ is better than SFDE in all the instances since the local search operator of ASFJ has the advantage of simplicity and efficiency, while SFDE involves multiple parameters and it is difficult to obtain appropriate values.

Figure 7 evaluates the generation process of the proposed ASFJ under the original scheduling, insertion and destruction rescheduling. ASFJ reaches a basic convergence when the population evolves to 500 generations. This indicates that the

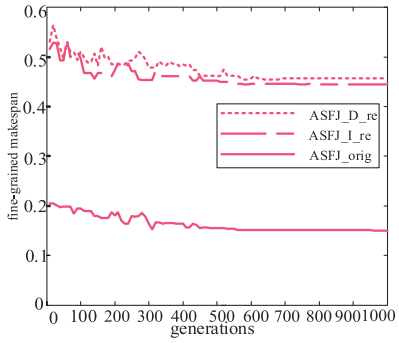


Fig. 7. ASFJ convergence performance.

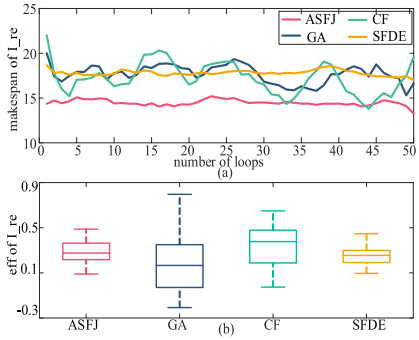


Fig. 8. Insertion rescheduling performance.

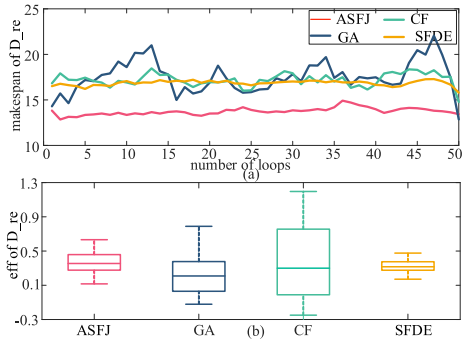


Fig. 9. Destruction rescheduling performance.

ASFJ achieves a stable state and searches for a relatively optimal solution within this number of generations.

Figure 8 and Figure 9 depict the makespan and effectiveness under insertion and destruction rescheduling. Figure 8(a) and Figure 9(a) show the makespan as the number of loops changes. In each loop, the population has evolved the optimal rescheduling solution as shown in Fig.7. The purpose of Fig. 8 (a) and Fig. 9 (a) is to evaluate the performance of the ASFJ algorithm when the number of ECNs and the number of tasks are fixed, while demonstrating stability. It is obvious that ASFJ is superior to other benchmark algorithms. ASFJ is able to save makespan by 10.2% and 20.9% on average than SFDE, respectively. The main reason is that the two local search operators of AFSJ search jump out of the local optimum, while SFDE converges prematurely.

Figure 8(b) and Fig. 9(b) demonstrate the effectiveness with box plots. It can be seen that the InterQuartile Range of AFSJ and SFDE is smaller, indicating that their effectiveness is more stable than the others. In Fig. 8(b), the median of CF is about 8.4% higher than AFSJ. This is because in small- scale

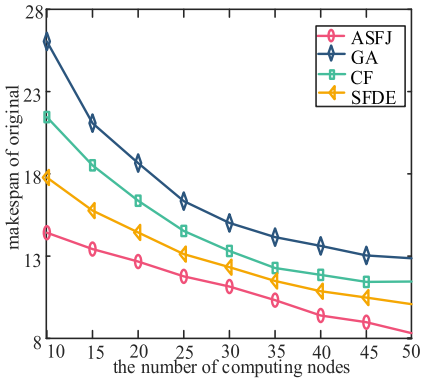


Fig. 10. Makespan of original.

instances, CF can iteratively obtain task-scheduling coalition pairs that minimize fine-grained makespan. However, the search space in ASFJ is smaller, which is more susceptible to the influence of local optimal solutions and ignores the wider search space. Therefore, Fig. 8 (b) shows the performance boundary of ASFJ in small-scale instances, and also provides guidance for further optimization of the scalability of ASFJ. The median of CF is about 3.5% lower than AFSJ in Fig. 9(b). The goal of CF is to allocate tasks in the process of balancing cost and benefit, but the budget will be reduced after the ECNs fail, which hinders the generation of better alliance pairs and dilutes the benefits. GA searches for the optimal solution randomly, and the effectiveness and stability are also the worst.

B. Performance When the Number of ECNs Changes

We evaluate the trend of the makespan, the effectiveness, and the fine-grained makespan versus the number of ECNs. The original number of tasks $|N|$ is set to 20 (including 80 subtasks). In order to intuitively compare the performance of rescheduling, the makespan of the original scheduling is depicted in Figure 10, and it can be seen that the makespan of all algorithms shows a downward trend. The reason is that more ECNs provide more resources, which can respond to requests more quickly. In particular, compared with benchmarks, ASFJ is able to save makespan by about 18% on average, which confirms the advantage of ASFJs' original scheduling. GA has the maximum makespan, and the difference between GA and ASFJ is at least 19.2% from 10 ECNs to 50 ECNs, which can be explained by the lack of local search ability of GA, which often obtains the suboptimal solution rather than the optimal solution. The makespan of ASFJ can save up to 35.2% compared with CF. It can be concluded that the constraints of CF are static, but the task scheduling strategy is the result of the dynamic coupling of the tasks and the ECNs' state. The core of the exploitation process of SFDE is DE, and the information differentiation among solutions of the DE becomes weaker as the number of iterations increases, which reduces the exploitation ability. Therefore, the makespan of SFDE is about 8.84% higher on average than ASFJ.

Figure 11(a) and Figure 11(b) consider uncertainties of task insertion and ECN destruction. To quantify the impact of uncertainty, the number of inserted tasks list in Fig. 11(a) is set to [3], [5], [8], [10], [13], [18], [20], [23], and [25] from

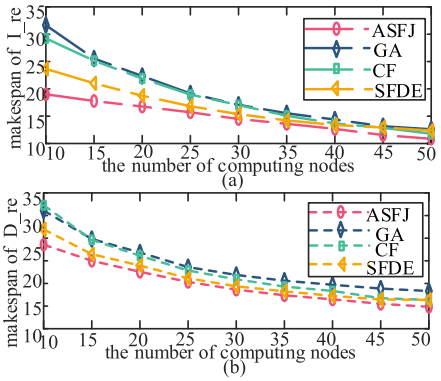


Fig. 11. Makespan of rescheduling.

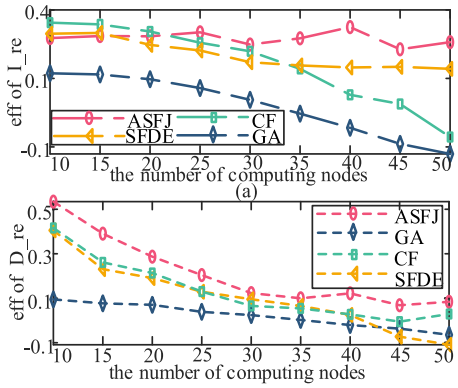


Fig. 12. Effectiveness of original.

10 ECNs to 50 ECNs, and the number of destroyed ECNs list in Fig. 11(b) is set to [2], [3], [5], [7], [8], [10], [12], [13], and [15]. Figure 11(a) and Fig. 11(b) show that the makespan decreases as the number of ECNs increases. Compared with Fig. 10, the makespan of ASFJ, GA, CF, and SFDE in Fig. 11(a) increased on average by 3.55ms, 2.29ms, 3.83ms, and 3.57ms, respectively. The makespan of GA is the largest in Fig. 11(a), but the gap between original and insertion is the smallest because the result is obtained by sacrificing some tasks, whose finished time exceeds maximum tolerable time in suboptimal solutions. The makespan increment of ASFJ is smaller than CF and SFDE, which can be explained by the fact that our proposed insertion rescheduling policy can adjust the assigning strategy of the affected original tasks in a timely manner, and adapt to the arrived tasks. Furthermore, the makespan gap between original scheduling and insertion rescheduling of ASFJ is reduced by 2.1% between 10 ECNs and 50 ECNs, at least 4.6% smoother than benchmarks. Especially, the makespan gap of CF is reduced by 19.8% between 10 ECNs and 50 ECNs, which is the largest. The performance gain of ASFJ over CF decreases from 40.9% with 10 ECNs to 3.7% with 50 ECNs in Fig. 11(a). This is because the cost of each task-ECN pair in CF is fixed, and inserted tasks will increase the execution cost of the original tasks, resulting in the distance from the optimal solution. ASFJ outperforms SFDE by 7.3% on average, which demonstrates that the two feasible local search operators can make the algorithm jump out of the local optimum. When the number of ECNs is 10 and 15, the average gap between the makespan in Fig. 11(b) and the makespan in Fig. 10 is about 8.56ms and 5.53ms, respectively. In addition, the makespan of ASFJ is the smallest and decreases by 3.77ms from 10 ECNs to 50 ECNs, which reflects that ASFJ has better robustness when the system fluctuates.

Figure 12 compares the effectiveness versus the number of ECNs. As shown in (23), effectiveness is defined as the sum of the ratios of the makespan gap to the original makespan on all ECNs. A larger effectiveness represents a greater change in the finish time of the last task on each ECN during rescheduling, which reflects the algorithm's effort in responding to uncertain factors, i.e., repair ability. In Fig.12, the effectiveness decreases as the number of ECNs increases, indicating that this is due to the fact that the original scheduling is less affected when more ECNs are proficient in dealing with fluctuations caused by uncertainty.

Figure 12(a) illustrates the effectiveness of insertion rescheduling. ASFJ has greater effectiveness than others. Combined with the results of the minimum insertion rescheduling and original makespan in Fig. 10 and Fig. 11(a) for ASFJ, we can infer that larger effectiveness implies that ASFJ more actively adjusts the execution decisions on each ECN to respond to the inserted tasks to minimize the fine-grained makespan. The average effectiveness difference between ASFJ and CF is nearly 11.8 %. This is because the increase in the number of tasks represents a higher requirement for the capacity of the alliance. However, in order to greedily search for the optimal task-ECN pairs under constraints, some tasks of CF cannot be assigned to any coalition.

Figure 12(b) plots the effect of the number of ECNs on effectiveness under destruction rescheduling. The slope becomes smaller as the number of ECNs increases, which can be explained by the fact that when the number of tasks is constant and more ECNs provide rescue services, ECNs' failure has less impact on the original scheduling with effective rescheduling, and the remedial effect on the original scheduling is smaller. As we expected, the effectiveness of ASFJ was on average about 38.4%, 21.2% and 16% higher than GA, CF, and SFDE, respectively. For SFDE, the imbalance between exploration and exploitation reduces the effectiveness. The performance difference between SFDE and CF increases from 2.2% at 10 ECNs to 27.6% at 50 ECNs, which proves the scalability of CF and it is more beneficial in large-scale applications. Effectiveness of GA is almost always the lowest, showing that GA is weaker than others in adapting to system changes.

In Fig. 13, the fine-grained makespan decreases as the number of ECNs increases, which implies that more resources are allocated to support task rescheduling, to ensure higher ECNs' utilization and lower makespan. ASFJ is able to decrease fine-grained makespan by 3.5%, 11.2% and 3.46% than SFDE in the original, insertion, and destruction scenarios, respectively, which indicates that the proposed two Jaya-based feasible local search operators are superior to the DE algorithm. For the original scheduling, when the number of ECNs is greater than 35, almost all tasks are effectively dispatched, and as a result, the fine-grained makespan tends to be stable. The performance gap of insertion rescheduling is the largest. This is because the increase in the number of tasks maps the increase in the population. The concise and

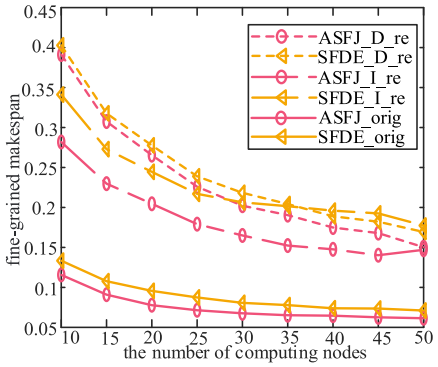


Fig. 13. Fine-grained makespan of original and rescheduling.

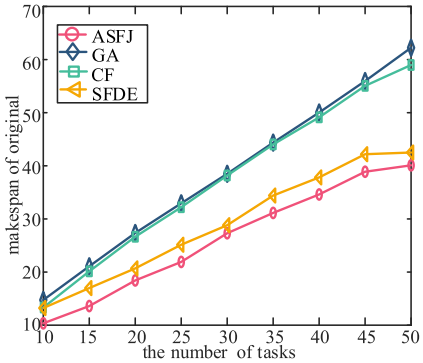


Fig. 14. Makespan under original.

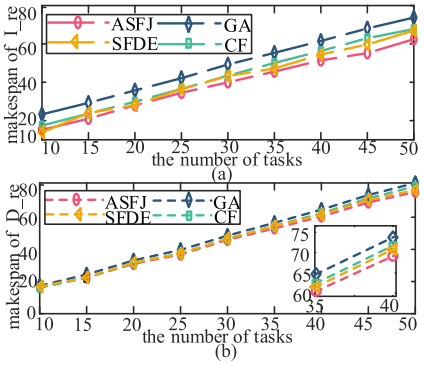


Fig. 15. Makespan under rescheduling.

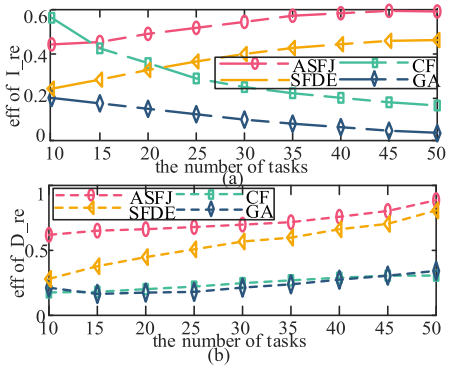


Fig. 16. Effectiveness under original.

fast advantages of Jaya make ASFJ efficiently search for high-quality solutions. Besides, the reason for the large fine-grained makespan of SFDE in destruction rescheduling is that it does not repair the infeasible solution, while ASFJ guarantees the generation of feasible solutions through the global optimal solution and local optimal solution.

C. Performance When the Number of Tasks Changes

Considering deterministic and uncertain emergency scenarios, we evaluate the effect of the number of tasks on makespan, effectiveness, and fine-grained makespan when the original number of ECNs is 10. For insertion rescheduling, the number of tasks arriving from 10 tasks to 50 tasks is 3, 5, 8, 10, 13, 18, 20, 23, and 25, respectively. The number of failed ECNs is 5 in destruction rescheduling. Similarly, the original makespan of the four algorithms is specified in Figure 14. It can be seen that the makespan becomes larger as the number of tasks increases. This is because the next subtask can only be executed after the successive predecessor subtask is released, and more tasks make the waiting time longer. ASFJ saves at least 4.2% on average compared with other algorithms, reflecting the scheduling benefits of ASFJ.

Figure 15 evaluates the makespan of task insertion and ECN destruction rescheduling based on the optimal original scheduling shown in Fig. 14. We can see that the makespan becomes larger as the number of tasks increases. In Fig. 15(a), the smallest makespan is achieved by ASFJ compared to the other approaches, i.e., around 16%, 7.5% and 6.4% lower than GA, CF and SFDE when the number of tasks is 50, respectively, reflecting the ability to quickly respond to uncertain factors. As shown in Fig. 15(b), the makespan of the four algorithms increases by at least 72.4% from 10 tasks to 50 tasks, because dependency constraints and resource competition expand the waiting time. When the number of tasks is 10, the makespan of the destruction rescheduling increases by 3.64ms on average compared with the original scheduling, which exhibits the service capability gap between 10 ECNs and 5 ECNs.

Figure 16(a) shows the change in the effectiveness of insertion rescheduling under different task numbers. The effectiveness of ASFJ and SFDE increases when the number of tasks increases, indicating that the increase in the proportion of the makespan gap to the original makespan, and the rate of increase in the makespan gap is greater than the rate of increase in the original makespan. ASFJ has the greatest effectiveness because it actively adjusts the scheduling strategy to respond to the inserted tasks while satisfying the constraints in the rescheduling process, rather than simply deploying the tasks on idle ECNs to maintain the original scheduling decision. However, the effectiveness of GA and CF is reduced, indicating that either the ECNs are always occupied, there are no free ECNs to provide services for other subtasks, or the order of subtasks does not perfectly adapt to constraints such as dependencies. It is worth noting that after the number of tasks is greater than 40, the performance of the four algorithms converges, which can be explained by the fact that each ECN is saturated to process the arrived tasks. In particular, when the number of tasks is equal to 50, the effectiveness of the proposed ASFJ reaches 0.59 and exceeds other algorithms by 21.5%.

Figure 16(b) analyzes the results on the effectiveness of destruction rescheduling when the number of tasks changes.

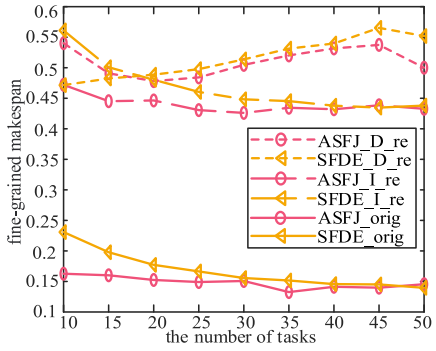


Fig. 17. Fine-grained makespan of original and rescheduling.

It can be seen that ASFJ maintains maximum effectiveness because when the number of ECNs is less, the diversity of the ECN assignment vector in the solution is reduced, while ASFJ ensures the diversity of solutions through the proposed population initialization and dynamic division and fusion of memeplexes. Therefore, a more effective solution not only makes full use of the remaining workable ECNs but also obtains beneficial task-ECN pairs to repair the impact of the destroyed ECNs on the original scheduling. The shuffled frog-leaping prevents the solution of SFDE from being assimilated due to the reduction of the number of ECNs, and the advantage is more obvious when the number of tasks increases. Therefore, as the number of tasks ranges from 10 to 50, the effectiveness gap between ASFJ and SFDE decreases from 0.34 to 0.08.

Figure 17 shows the fine-grained makespan under uncertain rescheduling and original scheduling. The performance of destruction rescheduling is the worst, which is due to the conflict between the larger number of tasks and the limited computing resources. The fine-grained makespan of ASFJ hardly fluctuates with the number of tasks, that is to say, the ratio of the makespan based on the task status value to the ECNs' idle time is basically stable. This can be explained by the fact that when the number of tasks increases, ASFJ maximizes the utilization of ECNs, taking into account the original priority and observation priority of the task at the same time to try to let the task be released when the state value is low.

VI. Conclusion

In this article, we considered an uncertainty-aware UAV emergency network, in which several new tasks arrive, some ECNs fail unpredictably, or ECNs' state parameters fluctuate. To improve rescue response efficiency, we formulated the task rescheduling problem to minimize the fine-grained makespan, which is designed to simultaneously characterize makespan and ECN utilization. To solve the optimization problem, we proposed the ASFJ algorithm. Firstly, we designed the heuristic population initialization method, in which energy consumption and execution time lead to high-quality initial solutions. Secondly, we presented the asynchronous shuffled frog-leaping method to divide the population into multiple memeplexes that evolve independently. Finally, to balance the exploration and exploitation process, we designed the feasible Jaya-based local search operator and the absorbing and swapping operator. Through a large number of simulation experiments under deterministic and uncertain scenarios, we evaluated the performance of ASFJ. Compared with other algorithms, ASFJ has better performance in terms of makespan, effectiveness, and fine-grained makespan. In future work, we will further investigate the task rescheduling scheme driven by the coupling relationship between energy consumption, execution time, and stability in uncertainty-aware UAV emergency networks, to improve the adaptability to different scale problems.

References

[1] N. Qi, Z. Huang, F. Zhou, Q. Shi, Q. Wu, and M. Xiao, "A task-driven sequential overlapping coalition formation game for resource allocation in heterogeneous UAV networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 8, pp. 4439–4455, Aug. 2023.

[2] J. Zhu, X. Wang, H. Huang, S. Cheng, and M. Wu, "A NSGA-II algorithm for task scheduling in UAV-enabled MEC system," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 9414–9429, Jul. 2022.

[3] Q. Li, M. Li, B. Q. Vo, and R. Kowalczyk, "An efficient algorithm for task allocation with the budget constraint," *Exp. Syst. Appl.*, vol. 210, Dec. 2022, Art. no. 118279.

[4] Z. Ning et al., "5G-enabled UAV-to-community offloading: Joint trajectory design and task scheduling," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 11, pp. 3306–3320, Nov. 2021.

[5] Z. Wang, B. Cao, C. Liu, C. Xu, and L. Zhang, "Blockchain-based fog radio access networks: Architecture, key technologies, and challenges," *Digit. Commun. Netw.*, vol. 8, no. 5, pp. 720–726, Oct. 2022.

[6] H. Huang, C. Hu, J. Zhu, M. Wu, and R. Malekian, "Stochastic task scheduling in UAV-based intelligent on-demand meal delivery system," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 13040–13054, Aug. 2022.

[7] Y. Cheng, Y. Xie, D. Wang, F. Tao, and P. Ji, "Manufacturing services scheduling with supply–demand dual dynamic uncertainties toward industrial internet platforms," *IEEE Trans. Ind. Informat.*, vol. 17, no. 5, pp. 2997–3010, May 2021.

[8] K. Lei et al., "A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem," *Exp. Syst. Appl.*, vol. 205, Nov. 2022, Art. no. 117796.

[9] F. Qiao, Y. Ma, M. Zhou, and Q. Wu, "A novel rescheduling method for dynamic semiconductor manufacturing systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 5, pp. 1679–1689, May 2020.

[10] H. Yan, W. Bao, X. Zhu, J. Wang, and L. Liu, "Data offloading enabled by heterogeneous UAVs for IoT applications under uncertain environments," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 3928–3943, Mar. 2023.

[11] D. Xu, Y. Sun, D. W. K. Ng, and R. Schober, "Multiuser MISO UAV communications in uncertain environments with no-fly zones: Robust trajectory and resource allocation design," *IEEE Trans. Commun.*, vol. 68, no. 5, pp. 3153–3172, May 2020.

[12] Y. Chen, B. Ai, Y. Niu, H. Zhang, and Z. Han, "Energy-constrained computation offloading in space-air-ground integrated networks using distributionally robust optimization," *IEEE Trans. Veh. Technol.*, vol. 70, no. 11, pp. 12113–12125, Nov. 2021.

[13] X. Wang and L. Duan, "Dynamic pricing and capacity allocation of UAV-provided mobile services," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Paris, France, Apr. 2019, pp. 1855–1863.

[14] H. Liu et al., "An iterative two-phase optimization method based on divide and conquer framework for integrated scheduling of multiple UAVs," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 9, pp. 5926–5938, Sep. 2021.

[15] J. Ding, Y. Wang, S. Zhang, W. Zhang, and Z. Xiong, "Robust and stable multi-task manufacturing scheduling with uncertainties using a two-stage extended genetic algorithm," *Enterprise Inf. Syst.*, vol. 13, no. 10, pp. 1442–1470, Nov. 2019.

[16] Y.-H. Jia et al., "An intelligent cloud workflow scheduling system with time estimation and adaptive ant colony optimization," *IEEE Trans. Syst. Man, Cybern., Syst.*, vol. 51, no. 1, pp. 634–649, Jan. 2021.

[17] Q. Pan, J. Tang, H. Wang, H. Li, X. Chen, and S. Lao, "SFSADE: An improved self-adaptive differential evolution algorithm with a shuffled frog-leaping strategy," *Artif. Intell. Rev.*, vol. 55, no. 5, pp. 3937–3978, Jun. 2022.

[18] D. Lei and X. Guo, "A shuffled frog-leaping algorithm for hybrid flow shop scheduling with two agents," *Exp. Syst. Appl.*, vol. 42, no. 23, pp. 9333–9339, Dec. 2015.

[19] R. V. Rao, D. P. Rai, J. Ramkumar, and J. Balic, "A new multi-objective Jaya algorithm for optimization of modern machining processes," *Adv. Prod. Eng. Manag.*, vol. 11, no. 4, pp. 271–286, Dec. 2016.

[20] F. Zhao, R. Ma, and L. Wang, "A self-learning discrete Jaya algorithm for multiobjective energy-efficient distributed no-idle flow-shop scheduling problem in heterogeneous factory system," *IEEE Trans. Cybern.*, vol. 52, no. 12, pp. 12675–12686, Dec. 2022.

[21] K. Z. Gao, M. C. Zhou, and Y. X. Pan, "Jaya algorithm for rescheduling flexible job shop problem with machine recovery," in *Proc. IEEE Int. Conf. Syst., Man Cybern. (SMC)*, Bari, Italy, Oct. 2019, pp. 3660–3664.

[22] L. Selvarajan, K. Venkataramanan, A. Nair, and V. P. Srinivasan, "Simultaneous multi-response Jaya optimization and Pareto front visualization in EDM drilling of MoSi2-SiC composites," *Exp. Syst. Appl.*, vol. 230, Nov. 2023, Art. no. 120669.

[23] I. Mlaouhi, N. Ben Guedria, and C. Bouraoui, "An efficient hybrid differential evolution-Jaya algorithm for enhancing vibration behaviour in automotive turbocharger systems," *Eng. Optim.*, Oct. 2023.

[24] R. Rao, "Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems," *Int. J. Ind. Eng. Comput.*, vol. 7, no. 1, pp. 19–34, 2016.

[25] Y. Pan, K. Gao, Z. Li, and N. Wu, "Solving biobjective distributed flow-shop scheduling problems with lot-streaming using an improved Jaya algorithm," *IEEE Trans. Cybern.*, vol. 53, no. 6, pp. 3818–3828, Jun. 2023.

[26] L. Sun, L. Lin, M. Gen, and H. Li, "A hybrid cooperative coevolution algorithm for fuzzy flexible job shop scheduling," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 5, pp. 1008–1022, May 2019.

[27] P. Milica et al., "Multi-objective scheduling of a single mobile robot based on the grey wolf optimization algorithm," *Appl. Soft Comput.*, vol. 131, Dec. 2022, Art. no. 109784.

[28] Y. An, X. Chen, K. Gao, Y. Li, and L. Zhang, "Multiobjective flexible job-shop rescheduling with new job insertion and machine preventive maintenance," *IEEE Trans. Cybern.*, vol. 53, no. 5, pp. 3101–3113, May 2023.

[29] C. Fan, B. Li, J. Hou, Y. Wu, W. Guo, and C. Zhao, "Robust fuzzy learning for partially overlapping channels allocation in UAV communication networks," *IEEE Trans. Mobile Comput.*, vol. 21, no. 4, pp. 1388–1401, Apr. 2022.

[30] F. Li, T. W. Liao, W. Cai, and L. Zhang, "Multitask scheduling in consideration of fuzzy uncertainty of multiple criteria in service-oriented manufacturing," *IEEE Trans. Fuzzy Syst.*, vol. 28, no. 11, pp. 2759–2771, Nov. 2020.

[31] B. Wang, Y. Sun, D. Liu, H. M. Nguyen, and T. Q. Duong, "Social-aware UAV-assisted mobile crowd sensing in stochastic and dynamic environments for disaster relief networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 1, pp. 1070–1074, Jan. 2020.

[32] Q. Chen, W. Meng, S. Han, C. Li, and H. Chen, "Robust task scheduling for delay-aware IoT applications in civil aircraft-augmented SAGIN," *IEEE Trans. Commun.*, vol. 70, no. 8, pp. 5368–5385, Aug. 2022.

[33] N. Ngoenriang, S. J. Turner, D. Niyato, and S. Supittayapornpong, "Joint UAV placement and data delivery in aerial inspection under uncertainties," *IEEE Internet Things J.*, vol. 9, no. 9, pp. 6389–6403, May 2022.

[34] Q. Luan, H. Cui, L. Zhang, and Z. Lv, "A hierarchical hybrid subtask scheduling algorithm in UAV-assisted MEC emergency network," *IEEE Internet Things J.*, vol. 9, no. 14, pp. 12737–12753, Jul. 2022.

[35] D. Lei, "Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling," *Appl. Soft Comput.*, vol. 12, no. 8, pp. 2237–2245, Aug. 2012.

[36] R. V. Rao and H. S. Keesari, "Multi-team perturbation guiding Jaya algorithm for optimization of wind farm layout," *Appl. Soft Comput.*, vol. 71, pp. 800–815, Oct. 2018.

[37] J. Fan, W. Shen, L. Gao, C. Zhang, and Z. Zhang, "A hybrid Jaya algorithm for solving flexible job shop scheduling problem considering multiple critical paths," *J. Manuf. Syst.*, vol. 60, pp. 298–311, Jul. 2021.

[38] E. O. Alkafaween, "Novel methods for enhancing the performance of genetic algorithms," 2018, *arXiv:1801.02827*.

[39] Y. An, X. Chen, K. Gao, L. Zhang, Y. Li, and Z. Zhao, "A hybrid multi-objective evolutionary algorithm for solving an adaptive flexible job-shop rescheduling problem with real-time order acceptance and condition-based preventive maintenance," *Exp. Syst. Appl.*, vol. 212, Feb. 2023, Art. no. 118711.