Multitree Genetic Programming With Rule Reconstruction for Dynamic Task Scheduling in Integrated Cloud–Edge Satellite–Terrestrial Networks

Changzhen Zhang, Jun Yang, and Ning Wang

***Abstract*—Satellite–terrestrial networks (STNs) are a promising paradigm for providing Internet services for users globally. Since the dynamics of service resources and the uncertainty of computational requests, how the service resources in STNs can be efficiently exploited to execute differentiated computational tasks is an essential challenge. In this work, we investigate the dynamic task scheduling in the integrated cloud–edge STNs. First, we propose a cloud–edge collaborative computing framework in STNs, where the computational tasks of users can be processed collaboratively by satellite edge servers, terrestrial edge servers, and cloud servers. Based on this framework, a dynamic task scheduling problem is formulated with the objective of maximizing the task success rate. Second, to make effective real-time decisions at decision points in the dynamic scheduling process, we develop a scheduling heuristic with the routing rule and queuing rule, which incorporates dynamic features related to servers, computational tasks, and network environments. Third, to automatically learn the scheduling heuristic, we propose a multitree genetic programming with rule reconstruction (MTGPRR), which introduces a selective reconstruction operator. This operator increases the chance of matching good rules with other rules by recombining common individuals and elites. Experimental results demonstrate that the proposed MTGPRR performs significantly better than the state-of-the-art methods in improving the task success rate. Moreover, the evolved scheduling heuristic has good interpretability, which is important for practical applications.**

***Index Terms*—Dynamic task scheduling, genetic programming, rule reconstruction, satellite–terrestrial networks (STNs), scheduling heuristic.**

I. Introduction

In RECENT years, with the rapid development of low Earth orbit (LEO) satellites, the satellite–terrestrial networks (STNs) are attracting widespread attentions. Significant advances in satellite technology—such as manufacturing, spot-beam antennas, and laser transmission—have facilitated the miniaturization, cost reduction, and high-throughput capabilities of LEO satellites [1]. These developments, coupled with innovations in antenna technology that exploit the Ku and Ka bands, have positioned LEO satellites as vital components in the evolving STNs [2].

With the exponential growth of computation-intensive and delay-sensitive applications, such as virtual reality, autonomous driving, and real-time analytics, STNs face unprecedented computational demands. While cloud computing offers vast computational power, its inherent geographic distance from end users often results in unacceptably high latency for delay-sensitive tasks. Multiaccess edge computing (MEC), which brings computing resources closer to users, effectively reduces latency and bandwidth requirements [3], [4]. Within STNs, MEC servers can be deployed on ground facilities (e.g., base stations and ground stations) as well as onboard LEO satellites, creating a unique hybrid computing architecture that offers both global coverage and reduced latency [5]. However, neither MEC nor cloud computing alone can fully meet the complex and dynamic requirements of STNs. MEC excels at low-latency processing but is constrained by its limited resources and local scope. Conversely, cloud computing provides superior scalability, reliability, and cost-efficiency but struggles with latency-sensitive tasks due to its distance from users [6], [7]. This calls for a collaborative computing paradigm integrating MEC and cloud computing within the STN framework to combine their complementary strengths. Therefore, this study aims to investigate collaborative computing in STNs to optimize resource utilization, enhance computational efficiency, and meet the growing demands of intelligent applications. By bridging the gap between MEC and cloud computing, collaborative computing can empower STNs to provide end users with seamless, low-latency, and reliable services, thus unlocking their full potential in civilian and industrial applications.

The efficient scheduling of computational tasks over computing resources is the key to improving users' Quality of Experience (QoE) and satisfying their needs. Since the heterogeneity of STNs, the dynamics of computing resources, and the stochastic nature of computational tasks, it is a great challenge to develop an efficient task scheduling mechanism [8], [9]. First, STNs include satellite and terrestrial networks, which differ in terms of hardware configuration,

Received 16 December 2024; revised 14 January 2025; accepted 26 February 2025. Date of publication 27 February 2025; date of current version 9 June 2025. This work was supported by the National Natural Science Foundation of China under Grant 72371008 and Grant 72301015. *(Corresponding author: Jun Yang.)*

The authors are with the School of Reliability and Systems Engineering, Beihang University, Beijing 100191, China (e-mail: zcz17853133145@163.com; tomyj2001@buaa.edu.cn; wangning1226@buaa.edu.cn).

Digital Object Identifier 10.1109/JIOT.2025.3546867

coverage, transmission delay, etc. Such diversity complicates the effective allocation and scheduling of computational tasks under different network conditions. Second, the load, energy, etc., of the servers used to process tasks change dynamically over time, resulting in the need for real-time adjustments in task scheduling. Finally, each computational task has its deadline, computational demand, nonperiodic arrival time, etc., which indicates that the computational tasks are stochastic [10]. Therefore, dynamic task scheduling in STNs imposes strict requirements on the scheduling mechanism. On the one hand, the scheduling mechanism needs to be highly adaptive. It should be able to respond in real time to changes in the network environment, the state of computing resources, and the demands of different tasks so that as many tasks as possible can be processed before their deadlines (i.e., to improve the task success rate). On the other hand, the scheduling mechanism enables fast decision-making during the scheduling process to cope with the delay-sensitive requirements of tasks.

Presently, many studies employ heuristic methods to solve the dynamic task scheduling problem in cloud and edge computing. Heuristic methods can be divided into two categories: 1) iterative improvement heuristic methods and 2) constructive heuristic methods. Common iterative improvement heuristic methods for the dynamic task scheduling problem include genetic algorithm [11], particle swarm algorithm [12], [13], simulated annealing [14], artificial bee colony optimization [15], and ant colony optimization [16]. While these methods promise to improve scheduling performance through evolutionary processes, their high computational overhead in dynamic environments limits their applicability for real-time scheduling. Constructive heuristic methods are usually based on extracting state characteristics of servers and computational tasks during the scheduling process to manually designed scheduling heuristics [17]. Compared to iterative improvement heuristics methods, they can make fast, real-time decisions during the dynamic scheduling process. In general, a scheduling heuristic consists of the routing rule determining server selection and the queuing rule determining task processing order. It solves the dynamic scheduling problem by assigning priorities to servers and tasks [18], [19]. While these manually designed scheduling heuristics can cope with the dynamic scheduling problem, they highly rely on domain knowledge [20]. Human experts may need help to capture the complex interactions between state features to design scheduling rules. Moreover, the decisions made by manually designed scheduling heuristics are mostly greedy decisions, which may not be good decisions during long-term scheduling [21]. Therefore, an automated design method is needed to learn the scheduling heuristics.

Genetic programming hyper-heuristic (GPHH) is an automated method for generating scheduling heuristics during a dynamic scheduling process. It is used in many optimization problems, including job shop scheduling [22], [23], cloud computing [10], [24], and routing problems [25]. In these domains, the scheduling heuristics evolved by GPHH have demonstrated significant advantages over manually designed scheduling heuristics, particularly in dynamic and stochastic environments. However, the scheduling challenges in STNs differ fundamentally from those in traditional scheduling problems. STNs exhibit unique characteristics, such as highly dynamic network topologies, heterogeneous computational resources, and stringent latency requirements, compounded by stochastic task arrivals and the need for real-time decision-making. These distinctive challenges render existing GPHH approaches insufficient, as they are not designed to adapt to the heterogeneous and distributed nature of STNs. To address these challenges, this work proposes a novel multitree genetic programming (MTGP) algorithm tailored to the dynamic task scheduling problem in STNs. The proposed algorithm is designed to automatically evolve adaptive scheduling heuristic, including routing rule and queuing rule, that can effectively manage the unique demands of STNs. Specifically, the main contributions of this work are shown as follows.

1) We propose a cloud–edge collaborative computing framework designed explicitly for STNs that enables dynamic task scheduling across satellite edge servers, ground edge servers, and cloud servers to maximize the task success rate.
2) To enable effective real-time decision-making during the dynamic scheduling process, we develop a new scheduling heuristic that integrates the routing rule and queuing rule to adapt to dynamic system features, including real-time load, energy consumption, and remaining deadlines, ensuring high responsiveness and efficiency in complex and evolving STNs environments.
3) We propose an MTGP with rule reconstruction (MTGPRR) to automatically learn the developed scheduling heuristic, which incorporates a selective reconstruction operator that recombines common individuals and elites to improve the likelihood of discovering effective combinations of rules.
4) Experimental results show that the proposed algorithm outperforms the state-of-the-art methods in different scenarios, and the scheduling heuristic evolved by MTGPRR has a sound mathematical interpretation that shows the relationship and importance of features.

The remainder of this work is organized as follows. Section II introduces the related works. Section III describes the system model and dynamic task scheduling problem. Section IV illustrates the proposed MTGPRR algorithm. Section V describes the experiment design, and Section VI analyzes the experiment results. Section VII summarizes this work and introduces future research directions.

II. Related Work

In this section, we review the heuristic methods and GPHH methods commonly employed for the task scheduling. Heuristic methods are broadly categorized into iterative improvement heuristics and constructive heuristics. Meanwhile, GPHH represents a learning-based approach that autonomously generates scheduling heuristics. A novel MTGPRR algorithm is introduced to address the limitations of existing methods.

A. Heuristic Method

1) Iterative Improvement Heuristic Method: Common iterative improvement heuristic methods for the dynamic task scheduling problem include genetic algorithm [11], particle swarm algorithm [12], [13], simulated annealing [14], artificial bee colony optimization [15], ant colony optimization [16], and so on. Pirozmand et al. [11] proposed a two-step hybrid scheduling method for cloud computing systems based on a genetic algorithm that involves prioritizing tasks and assigning tasks. The results of the evaluation show that the proposed method has a better makespan and energy consumption. Fu et al. [12] proposed a hybrid particle swarm optimization (PSO) and genetic algorithm based on the phagocytosis mechanism to optimize task scheduling in cloud environments by improving the search range and population diversity and significantly improving the task completion time and algorithm convergence accuracy. Wu et al. [13] proposed the end–edge–cloud heterogeneous resource scheduling method (EHRSM), combining a recurrent neural network (RNN) and a PSO algorithm, achieving significant reductions in task completion time and waiting time in cloud computing environments. Celik and Dal [14] developed a simulated annealing-based metaheuristic for cluster-based task scheduling, demonstrating its effectiveness through serial and parallel implementations that achieve superior results compared to existing benchmarks. Kishor and Chakarbarty [15] proposed a smart ant colony optimization (SACO) algorithm for task offloading in fog computing, achieving significant latency reductions in IoT-sensor applications compared to baseline algorithms. Zeedan et al. [16] proposed the enhanced binary artificial bee colony-based Pareto front (EBABC-PF) algorithm for workflow scheduling in cloud computing, achieving improved makespan, processing cost, and resource utilization compared to existing methods. Iterative improvement heuristics methods effectively find quality solutions and deal with large-scale problems. However, their reliance on rescheduling processes makes them time-intensive and inefficient for responding promptly to dynamic events, limiting their suitability for dynamic task scheduling problem in STNs.

2) Constructive Heuristic Method: The constructive heuristic methods solve the dynamic scheduling problem in the form of scheduling heuristics. They support real-time scheduling decisions by assigning priorities to tasks and servers at decision points based on the system state. Wang et al. [26] studied various common scheduling algorithms in cloud computing, including first come first served (FCFS), last come first served (LCFS), random selection for service (RSS), and earliest deadline first (EDF). Li [27] proposed a heuristic algorithm for energy-constrained task scheduling in device–edge–cloud systems, addressing the interleaved challenges of task scheduling and power allocation. Chai et al. [28] proposed a multitask offloading and resource allocation scheme for satellite IoT based on attention mechanism and proximal policy optimization, optimizing cost and efficiency by modeling task dependencies as directed acyclic graphs (DAGs) and leveraging autonomous aerial vehicles-based task collection. Lou et al. [29] proposed the startup-aware dependent task scheduling (SDTS) algorithm to optimize task finish times by coordinating startup, data transmission, and execution in heterogeneous edge computing environments with bandwidth and resource constraints. Shen [30] introduced a hierarchical task scheduling strategy designed to address the issue of significant delay overhead in MEC application scenarios. Zhang and Yang [31] classified computational tasks into delay-sensitive tasks and delay-tolerant tasks based on the deadlines of the tasks and proposed a collaborative service architecture with ground edges, satellite edges, and clouds. Xu et al. [32] proposed an adaptive mechanism for dynamically collaborative computing power and task scheduling (ADCS). It leverages a greedy decision approach to efficiently schedule computing tasks among edge nodes, ensuring they meet deadline requirements. Okegbile et al. [33] proposed a multiuser, multiclass, and multilayer edge computing framework that integrates stochastic geometry, parallel computing, and queuing theory to optimize task offloading and computation, improving performance for delay-sensitive and mission-critical applications. These methods provide real-time decision-making capabilities but rely heavily on domain expertise. The complexity of dynamic scheduling systems often surpasses human capacity to synthesize all feature interactions, limiting scalability.

B. GPHH Method

GPHH has emerged as a powerful tool for dynamic task scheduling, capable of automatically learning scheduling heuristics by interacting with dynamic environments. Yang et al. [24] proposed a GPHH approach to solving the dynamic scheduling problem in cloud computing, aiming to minimize both virtual machine rental fees and service level agreement (SLA) penalties. Sun et al. [10] proposed a new genetic programming with multitree representation to solve dynamic scheduling problem in multiclouds. Xu et al. [21] introduced a new problem model and simulator for dynamic scheduling in fog computing and proposed an MTGP method to evolve scheduling heuristics for effective real-time routing and sequencing decisions across mobile devices, edge, and cloud servers. MTGP means that each individual consists of multiple trees during the evolutionary process, and each tree represents a scheduling rule [34], [35]. Therefore, MTGP enables the simultaneous learning of multiple scheduling rules. An individual's fitness depends on the combination effect of the two trees. Therefore, reconstructing the two trees may achieve better fitness than the original individual. In the classical MTGP algorithm [36], the population can evolve a good batch of rules after reproduction, crossover, and mutation. However, the algorithm only focuses on the independent evolution of the single rules. It neglects to improve the algorithm's population diversity and exploration ability through the reconstruction between different rules. In [21], the crossover operator of MTGP is changed to a swapping crossover operator, i.e., swapping another rule while performing a crossover operation on one rule, to improve the algorithm performance. In [37], a swapping crossover operator is proposed to crossover on more trees and swapping trees between parents. Although the swapping crossover operators focus on the reconstruction

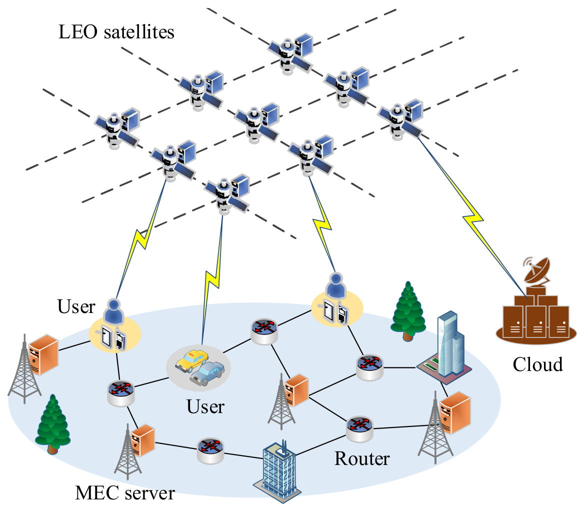

Fig. 1. Cloud–edge collaborative computing framework in STNs.

between rules, their improvements remain limited due to a lack of selectivity, potentially disrupting high-performing individuals. To overcome these shortcomings, it is necessary to design a new rule reconstruction operator to improve the algorithm's population diversity and overall performance. This new operator should be able to perform reconstruction between different rules with some selectivity efficiently.

In summary, scheduling heuristics are effective for real-time decision-making but are limited by dependence on expert knowledge and scalability. Meanwhile, GPHH methods, particularly MTGP, offer a promising alternative. However, existing MTGP approaches inadequately exploit the combination effect of rules to improve population diversity. To address these gaps, this work proposes an enhanced MTGP method with a novel rule reconstruction operator that recombines common individuals and elites to improve the likelihood of discovering effective combinations of rules, improving population diversity and overall algorithm performance.

III. System Model and Problem Description

In this section, we describe the system model of the integrated cloud–edge STNs, including the communication model, the delay model, and the energy model. Based on these, the proposed dynamic task scheduling problem is then formulated.

A. System Model

The system consists of the satellite network and terrestrial network as shown in Fig. 1. The satellite network consists of LEO satellites equipped with MEC servers, which can handle computational tasks from user devices. LEO satellites are connected to each other through intersatellite links (ISLs). The terrestrial network mainly consists of user devices, ground MEC servers, and cloud. Therefore, user devices can offload computational tasks to LEO satellites via satellite–terrestrial links or to the cloud via the LEO satellite backhaul link, and can also access the ground MEC servers via radio access networks [3].

The set of LEO satellites is denoted as $\mathbf{V}^S = \{V_1^S, V_2^S, \dots, V_{N_S}^S\}$, where N_S is the number of LEO satellites. The set of ground MEC servers is denoted as $\mathbf{V}^{\mathrm{G}} = \{V_1^{\mathrm{G}}, V_2^{\mathrm{G}}, \dots, V_{N_{\mathrm{G}}}^{\mathrm{G}}\}$, where N_{G} is the number of ground MEC servers. The set of cloud servers is denoted as $\mathbf{V}^{\mathrm{C}} = \{V_1^{\mathrm{C}}, V_2^{\mathrm{C}}, \dots, V_{N_{\mathrm{C}}}^{\mathrm{C}}\}$, where N_{C} is the number of cloud servers. Moreover, the set of all computational nodes in the system is denoted as $\mathbb{V} = \mathbf{V}^S \cup \mathbf{V}^{\mathrm{G}} \cup \mathbf{V}^{\mathrm{C}}$. At time t, the state of node $V_j (V_j \in \mathbb{V})$ is denoted as $S_j(t) = \{L_j(t), E_j(t), \mu_j, L_{\max}^j\}$, where $L_j(t)$ is the computing load of node V_j at time t, $E_j(t)$ is the remaining energy of node V_j at time t, μ_j is the computing capability of node V_j, and $L_{\max}^j$ is the capacity of node V_j.

The computational tasks $\mathbb{M}$ from user devices are randomly generated and different tasks have different attributes. The ith task $M_i (M_i \in \mathbb{M})$ is modeled as a four tuple $M_i = \{g_i, x_i, \epsilon_i, d_i\}$, where g_i is the generation time of task M_i, x_i is the task size (in bits), ϵ_i is the workload of task M_i (in cycles/bit), and d_i is the deadline of task M_i. If the task is processed before its deadline, it is called a task success, otherwise it is called a task failure [38].

1) Communication Model: The communication model includes the satellite–terrestrial link communication model, the ISLs communication model, and terrestrial link communication model. Let $\mathbb{D} = \{D_1, D_2, \dots, D_{|\mathbb{D}|}\}$ denote the user device set.

a) Satellite–terrestrial link communication model: The complex space environment affects the communication quality between the user device and LEO satellites. Furthermore, it is noted that the communication quality of the link varies with the movement of LEO satellites. Referring to [39], the signal-to-noise ratio (SNR) over the satellite–terrestrial link between the user device D_m and LEO satellite V_n^S at time t can be expressed as

$$\mathrm{SNR}_{m,n}(t) = \frac{P_m^{\mathrm{D}} G_m^{\mathrm{D}} G_n^S L_{m,n}^f(t) L_{m,n}^p(t)}{N_0} \tag{1}$$

where t is presented as a discrete-time slice, which represents the triggering moment of discrete events. P_m^{D} is the transmission power of D_m, G_m^{D} and G_n^S are the antenna gains of D_m and V_n^S, respectively, and N_0 is the noise power. According to Recommendation ITU-R P.618-12 [40], the rain attenuation at time t, $L_{m,n}^p(t)$, is affected by carrier frequency, elevation angle, altitude above the sea level, and rainfall intensity. It is given by

$$L_{m,n}^p(t) = \omega \cdot L_e(t) \cdot \psi^{\sigma} \tag{2}$$

where ω and σ are the frequency-dependent coefficients, and ψ is the rainfall intensity. $L_e(t)$ is the effective path length of the wave in rain at time t and is calculated as $L_e(t) = h_r / \sin\theta(t)$, where h_r is the rain height, and $\theta(t)$ is the elevation angle of D_m with respect to V_n^S at time t. $L_{m,n}^f(t)$ is the free space loss at time t, which can be calculated as

$$L_{m,n}^f(t) = \left(\frac{c}{4\pi d_{m,n}(t) f}\right)^2 \tag{3}$$

where c is the speed of light, and f is the communication center frequency of the link. $d_{m,n}(t)$ is the slope distance between

D_m and V_n^S at time t, which is calculated using the simulation tool to capture the real-time position of the satellite.

Next, the attainable data rate $r_{m,n}^{S\text{-}T}(t)$ for the link from D_m to V_n^S at time t can be formulated as

$$r_{m,n}^{S\text{-}T}(t) = B_{m,n}^{S\text{-}T} \log_2\left(1 + \text{SNR}_{m,n}(t)\right) \tag{4}$$

where $B_{m,n}^{S\text{-}T}$ denotes the available bandwidth of the link.

b) ISLs communication model: The LEO satellites are interconnected via laser ISLs. At time t, the data rate $r_{m,n}^{\text{ISLs}}(t)$ from LEO satellite V_m^S to LEO satellite V_n^S is given by [41]

$$r_{m,n}^{\text{ISLs}}(t) = \frac{P_m^S G_m^S G_n^S L_{m,n}^f(t)}{kT_s(E_b/N_1)M} \tag{5}$$

where P_m^S is the transmission power of V_m^S, G_m^S and G_n^S are the antenna gains of V_m^S and V_n^S, respectively, k is the Boltzmann constant, T_s is the total system noise temperature, E_b/N_1 is the required received energy per bit relative to the noise density, and M is the link margin. The corresponding free space loss $L_{m,n}^f(t)$ is calculated using (3).

c) Terrestrial link communication model: The data rate $r_{m,n}^{T\text{-}T}$ from user device D_m to ground MEC server V_n^{G} via radio access networks can be obtained by [42]

$$r_{m,n}^{T\text{-}T} = B_{m,n}^{T\text{-}T} \log_2\left(1 + \frac{P_m^{\text{D}} g_0 (u_0/u_{m,n})^\rho}{B_{m,n}^{T\text{-}T} N_2}\right) \tag{6}$$

where g_0 is the path loss constant, u_0 is the reference distance, $u_{m,n}$ is the distance between D_m and V_n^{G}, ρ is path loss exponent, P_m^{D} is the transmission power of D_m, and N_2 is noise power.

2) Energy Model: The LEO satellite is small in size and has a limited energy available for processing tasks. The ground MEC server and cloud server have sufficient energy supply. Therefore, as a limiting factor, the real-time residual energy of LEO satellites must be taken into account.

The energy consumption of LEO satellites mainly comes from computation energy consumption for processing computational tasks and communication energy consumption for transmitting data. Similar to [43], the computation energy consumption $E_{\text{com}}^{i,j}$ of LEO satellite V_j^S for processing computational task M_i is given by

$$E_{\text{com}}^{i,j} = \varepsilon\left(\mu_j\right)^3 D_{\text{com}}^{i,j} \tag{7}$$

where ε is the effective capacitance coefficient of computing hardware.

The communication energy consumption of LEO satellite V_m^S for transmitting the computational task M_i is expressed as

$$E_{\text{tra}}^{i,j} = P_j^S D_{\text{tra}}^{i,j,n}. \tag{8}$$

LEO satellites derive their energy primarily from onboard energy harvesting systems, such as solar panels. Let P_h denote the power of the energy harvesting system, given a time period t_p, the energy generated E_{gen} of LEO satellite is calculated as [44]

$$E_{\text{gen}} = P_h t_p. \tag{9}$$

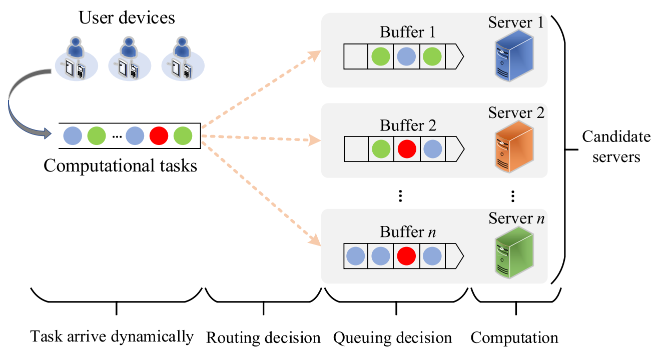

Fig. 2. Dynamic task scheduling process.

B. Problem Description

1) Problem Overview: Each computational task has its deadline and needs to be processed before this deadline, otherwise it is a task failure. Therefore, the dynamic task scheduling in the integrated cloud–edge STNs concentrates on improving the task success rate by allocating computing resources and arranging the task processing order. The computational tasks from user devices are calculation-intensive, and they are independent and indivisible [45]. These tasks are processed by the candidate servers (satellite MEC server, ground MEC server, or cloud server). Moreover, computation-intensive tasks have high resource requirements and they will exclusively occupy the resources of the entire server. Therefore, a server can only process one task at a time, and the rest of the tasks are queued in the buffer. It implies that there are two types of decisions in the dynamic task scheduling problem, i.e., routing decision and queuing decision, as shown in Fig. 2.

Routing Decision: Once a computational task is generated from a user device, it is assigned to a candidate server in a candidate list based on the routing rule. The candidate servers are selected from satellite MEC servers, ground MEC servers, and cloud servers. Since the satellite is moving, the distance between the satellite and the user device is constantly changing. To maintain the transmission continuity, the access satellite needs to be the closest to the user device among the satellites capable of covering the user device when the task is offloaded to the satellite MEC server according to the routing rule. For satellite MEC servers, both capacity and energy requirements for processing the computational task need to be satisfied before entering the candidate list. That is, for the computational task M_i, the LEO satellite V_j^S in the candidate list needs to fulfill the following conditions:

$$\begin{cases} L_{\max}^j - L_j(t) \geq x_i \\ E_j(t) \geq E_{\text{com}}^{i,j} \end{cases} \tag{10}$$

where $L_{\max}^j - L_j(t) \geq x_i$ denotes that the remaining capacity of V_j^S at time t is sufficient for M_i. $E_j(t) \geq E_{\text{com}}^{i,j}$ denotes that the remaining energy of V_j^S at time t is sufficient for the computation energy consumption of M_i. For ground MEC servers and cloud servers, only the capacity requirement for processing the computational task needs to be satisfied to enter the candidate list.

Queuing Decision: After a computation task is assigned to a particular server based on the routing rule, it is processed immediately if the server is idle; otherwise, it is queued in the buffer. The computational tasks in the buffer have multidimensional attributes such as deadline, task size, arrival time, etc. Based on these attributes, the queuing rule assigns priority to each task, and these tasks are queued based on the priority.

Furthermore, the coverage area of LEO satellites is constantly changing over time due to their high mobility speed, resulting in frequent handovers. When a task transmission is interrupted due to the mobility of the satellite, it is necessary to reconnect to a satellite or a ground node capable of covering the target area. In this work, when a handover is triggered, the scheduler uses the routing rule to reschedule the current task according to the real-time states. The handover condition is that the LEO satellite's elevation angle relative to the user device reaches a minimum value [46].

2) Constraints and Assumptions: The dynamic task scheduling in STNs is subject to the following constraints and assumptions.

1) The information of a computational task is unknown until it is generated from a user device, i.e., the computational task is stochastic. This demands that the scheduling algorithm is capable of making decisions in real-time [10].
2) Preemption priority is not permitted, meaning a computational task already in progress cannot be interrupted. This assumption is crucial for maintaining task integrity and stability. In many practical applications, continuity of task processing is critical. For example, in real-time control systems or financial transactions, interruptions can cause significant errors or data inconsistencies. Similarly, any interruption could lead to catastrophic outcomes for safety-critical operations like medical monitoring or autonomous vehicle navigation [31].
3) The computing capability and capacity of a server are constant. Typically, the computing capabilities and resources of servers are established during the design and manufacturing process and remain largely stable throughout their operational lifespan [19]. This consistency allows the scheduling algorithm to optimize and allocate resources based on predictable computational parameters, thereby enhancing efficiency and reliability.
4) Server breakdowns are not considered in the dynamic scheduling process. The work concentrates on improving the efficiency and effectiveness of the scheduling process itself, rather than on fault tolerance or recovery mechanisms.

3) Objective: In this work, the objective is to maximize the task success rate during the dynamic task scheduling. A high task success rate typically reflects not only enhanced user satisfaction but also improved resource utilization and balanced system load. Accordingly, the objective optimization problem can be formulated as

$$\max_{M_i \in \mathbf{M}} P_S = \frac{\sum_{i=1}^{N_T} I_{\{D^i_{\text{total}} \le d_i\}}}{N_T} \tag{11}$$

$$\sum_{j=1}^{N_O} y_{i,j} = 1 \tag{12a}$$

$$x_i y_{i,j} \le L^j_{\max} - L_j(t) \tag{12b}$$

$$E^{i,j}_{\text{com}} y_{i,j} \le E_j(t) \tag{12c}$$

$$y_{i,j} \in \{0, 1\} \tag{12d}$$

$$i = 1, \dots, N_T, j = 1, \dots, N_O. \tag{12e}$$

In (11), N_T is the total number of computational tasks during the simulation time. D^i_{total} is the total delay of computational task M_i, which is composed of transmission delay, propagation delay, computation delay, and queuing delay [47], and it is calculated as

$$D^i_{\text{total}} = D^i_{\text{tra}} + D^i_{\text{pro}} + D^i_{\text{com}} + D^i_{\text{que}}. \tag{13}$$

In (13), the four delays constituting D^i_{total} are specified as follows.

1) D^i_{tra} is the transmission delay of M_i, which is the time required for a task to be transmitted over a communication link and is usually related to the data rate and the task size. It occurs during transmission over satellite–terrestrial links or ISLs. When the computational task M_i is transmitted from user device D_m and LEO satellite V^S_n, the transmission delay $D^{i,m,n}_{\text{tra}}$ is calculated as $D^{i,m,n}_{\text{tra}} = x_i / r^{S\text{-}T}_{m,n}$, where x_i is the task size of M_i, and $r^{S\text{-}T}_{m,n}$ is the data rate of the link from D_m to V^S_n. Since the task transmission time is very short (typically on the millisecond level) and channel condition does not change significantly during this period, for simplicity, we approximate $r^{S\text{-}T}_{m,n}$ as a constant over the duration of task transmission, and it is calculated as the data rate at the beginning of the transmission.
2) D^i_{pro} is the propagation delay of M_i, which refers to the signal propagation time between the user device and the satellite or between satellites. For user device D_m and LEO satellite V^S_n, the propagation delay $D^{i,m,n}_{\text{pro}}$ of M_i between the two nodes is calculated as $D^{i,m,n}_{\text{pro}} = d_{m,n}/c$, where $d_{m,n}$ is the slope distance between D_m and V^S_n, it is consider as a constant during propagation. c is the speed of light.
3) D^i_{com} is the computation delay of M_i. If the computational task M_i is processed on LEO satellite V^S_n, the computation delay $D^{i,n}_{\text{com}}$ is calculated as $D^{i,n}_{\text{com}} = x_i \epsilon_i / \mu_n$.
4) D^i_{que} is the queuing delay of M_i, which refers to the time a task waits in the queue of the server. If the computational task M_i is offloaded to LEO satellite V^S_n, the queuing delay $D^{i,n}_{\text{que}}$ is calculated as $D^{i,n}_{\text{que}} = \sum_{k=1}^{Z} D^{k,n}_{\text{com}}$, where Z denotes the number of tasks that rank ahead of M_i.

$I_{\{D^i_{\text{total}} \le d_i\}}$ denotes an indicator function with the expression

$$I_{\{D^i_{\text{total}} \le d_i\}} = \begin{cases} 1, & D^i_{\text{total}} \le d_i \\ 0, & D^i_{\text{total}} > d_i. \end{cases} \tag{14}$$

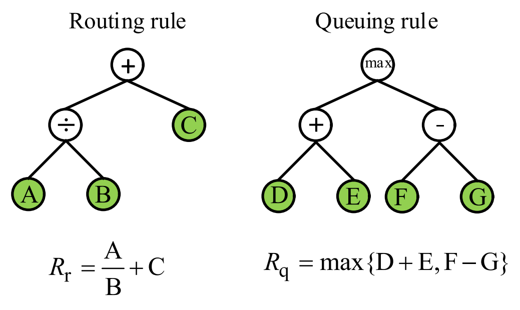

Fig. 3. Example of a scheduling heuristic.

In (12a), $y_{i,j} = 1$ indicates that M_i will be processed on the server V_j, while $y_{i,j} = 0$ indicates that M_i will not be processed on the server V_j. N_O is the total number of servers that $N_O = N_S + N_G + N_C$. Constraint (12b) denotes that the remaining capacity of server V_j at time t is sufficient for M_i if M_i is processed on V_j. Constraint (12c) denotes that the remaining energy of V_j at time t is sufficient for the computation energy consumption of M_i if M_i is processed on V_j. Constraint (12c) acts primarily on LEO satellites because the ground servers have sufficient energy access.

IV. Algorithm Design

This section describes the newly proposed MTGPRR algorithm for solving the dynamic scheduling problem in the integrated cloud–edge STNs. The representation of routing rules and queuing rules is first introduced. Terminal sets for describing computing resources and computational tasks are then designed. Finally, the evolution of the scheduling heuristic is described in detail.

A. Representation

MTGP is capable of evolving a scheduling heuristic with routing rule and queuing rule to solve dynamic scheduling problem in the integrated cloud–edge STNs. In MTGP, an individual consists of a routing rule and a queuing rule. Each rule is represented as a tree-based priority function. As shown in Fig. 3, the priority function determined by the routing rule is $R_r = (A/B) + C$. The priority function determined by the queuing rule is $R_q = \max\{D+E, F-G\}$. In the priority function, variables are the key features that affect the scheduling decision and they constitute the terminal set of the algorithm. All operators constitute the function set of the algorithm.

B. Terminal Set and Function Set

Considering the influences on routing decisions and queuing decisions, we designed nine terminals. They reflect the characteristics related to computational tasks, computing resources, and the network environment, describing the state of the integrated cloud–edge STNs. The terminal set is shown in Table I and described in detail below.

DT is the delay for transferring data to the server, which reflects the distance of different computing resources relative to the user device. CC is the computing capability of the server. NT is the total number of computational tasks in

TABLE I
Terminals and Functions of MTGP

Notation	Description	Rout	Queue
DT	The delay for transferring data to the server	✓	
CC	The computing capability of the server	✓	
NT	The total number of computational tasks in the server	✓	
CL	The total computing load in the server	✓	
RE	The remaining energy of the server	✓	
AT	The arrival time of a computational task		✓
TS	The task size of a computational task		✓
IDDL	The initial deadline of a computational task		✓
RDDL	The remaining deadline of a computational task		✓
Function set	+, -, ×, ÷, max, min	✓	✓

the server, which measures the congestion level of a given server. CL is the total computing load in the server, which reflects the remaining computational time of a given server. RE is the remaining energy of the server, which is the access limit of the computational tasks. AT is the arrival time of a computational task, on which the classical FCFS heuristic is based. TS is the task size of a computational task, on which the EDF heuristic is based. IDDL is the initial deadline of a computational task, which reflects the overall urgency of the computational task. RDDL is the remaining deadline of a computational task, which reflects the real-time urgency of the computational task.

The routing rules and queuing rules have different terminals. We use "✓" to mark the terminals that form the corresponding rules. The function set is $+$, $-$, $\times$, $\div$, max, min, and each operator acts on two terminals. The $\div$ operator refers to protected division, i.e., returning 1 when the denominator is 0. The function set is the same for two rules.

C. Evolutionary Process

MTGPRR aims to evolve scheduling heuristics to make routing decisions and queuing decisions for the dynamic task scheduling. The workflow of MTGPRR is shown in Fig. 4. Different from the traditional MTGP method, the main innovation of this work is the introduction of a new rule reconstruction operator to enhance the effectiveness of the algorithm. The details of each step of the proposed algorithm are described as follows.

1) *Initialization:* A prescribed number of individuals are randomly initialized by the ramped-half-and-half method based on predefined terminal and function sets. Half of the individuals are initialized with a predefined maximum depth, while the other half are initialized randomly within this depth. In our problem, each individual consists of two tree structures: a) the routing rule and b) the queuing rule.
2) *Evaluation:* To evaluate the fitness of an individual, it is applied to the dynamic scheduling process given a set of instances. After a simulation period, the output task success rate is taken as the fitness of this individual. The discrete event-driven simulation is used to describe

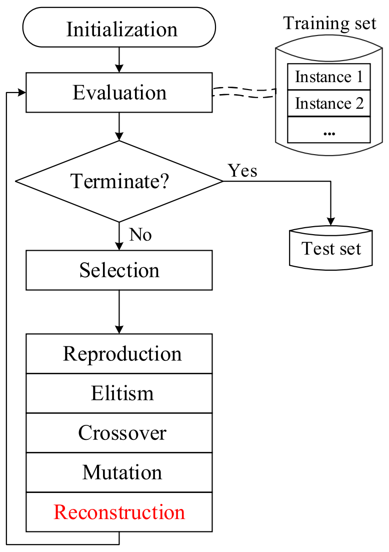

Fig. 4. Workflow of MTGPRR.

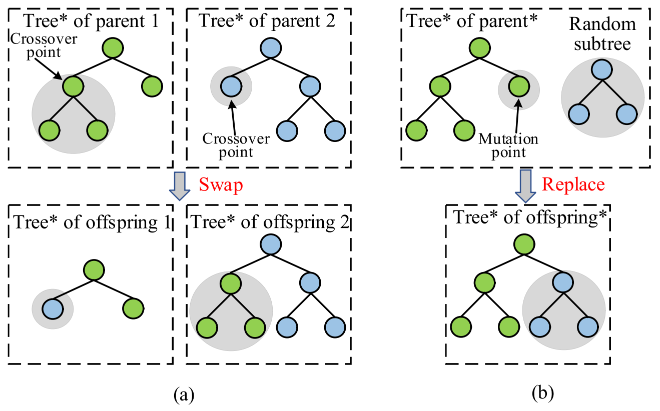

Fig. 5. Crossover and mutation processes. (a) Process of the crossover. (b) Process of the mutation.

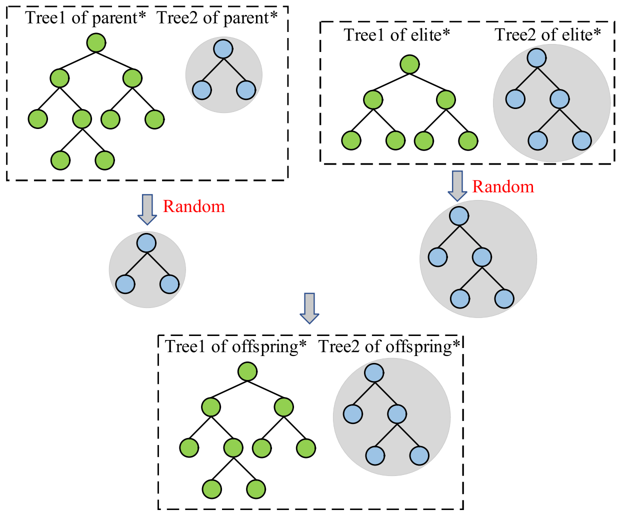

Fig. 6. Reconstruction process.

the system's dynamic scheduling process. During the simulation, the computational tasks arrive according to a Poisson process. The task arrival and task processing completion are considered basic events. The dynamic simulation process is executed by updating the event list and the system state.

3) *Selection:* The termination condition of the algorithm is a certain number of generations, and once it is met, the optimal individual is output; otherwise, tournament selection is used to select parents for generating offspring.
4) *Reproduction, Elitism, Crossover, and Mutation:* Reproduction, elitism, crossover, and mutation are key operations for evolving scheduling heuristics. We copy a part of the selected parent directly to the next generation for reproduction. For elitism, the few individuals with the highest fitness in the current population enter the Hall of Fame and are retained into the next generation. Two parents are randomly selected for crossover, and one of the trees (routing rule or queuing rule) is chosen, a subtree crossover operation is performed, and a swap operation is performed on the other tree. For mutation, a new subtree is randomly generated and replaces the subtree of the selected parent. The specific crossover and mutation processes are shown in Fig. 5.
5) *Reconstruction:* Individuals in a population inherit a tree from a random elite in the Hall of Fame with some probability. That is, replacing the corresponding tree of this common individual with a tree of an elite. The specific reconstruction process is shown in Fig. 6.

The reconstruction operator is proposed based on the following considerations. On the one hand, the common individual is likely to achieve high fitness by selecting the excellent individual for swapping rather than the common individual. Whereas elites happen to be the best individuals evaluated in a given generation. Therefore, the swapping of common individuals with elites reflects the selectivity of the proposed reconstruction operator. On the other hand, from the elite's point of view, this operation increases the chances of matching the elite's good tree with other trees, which facilitates the exploration of better scheduling heuristics.

The reconstruction may cause some individuals in the offspring to inherit the same tree from the same elite. This increases the probability that these individuals have the same behavior, i.e., make the same decisions. To avoid this problem, the behavioral repetition test is performed on the reconstructed individuals. For individuals that make the same decisions, perform the mutation operation once. An example of the repeated behavior of individuals is given in Table II. Table II gives five servers and four tasks. Each individual is used as a scheduling heuristic to compute the priority values of the servers and tasks. The server or task with a smaller priority value is ranked higher. Individuals are behavioral repetitive if they compute the same combined rank. As shown in the table, individuals 1 and 3 are behavioral repetitive.

D. Complexity Analysis

Suppose g is the number of evolutionary generations, p is the population size, n is the number of tasks in the decision

TABLE II
EXAMPLE OF REPEATED BEHAVIOR OF INDIVIDUALS

Individual	Server priority value	Task priority value	Server rank	Task rank	Combined rank
1	[10,20,15,8,30]	[100,130,90,110]	②④③①⑤	②④①③	②④③①⑤②④①③
2	[25,10,12,16,20]	[80,100,120,130]	⑤①②③④	①②③④	⑤①②③④①②③④
3	[15,22,18,10,25]	[100,130,90,110]	②④③①⑤	②④①③	②④③①⑤②④①③
4	[20,10,24,30,18]	[100,130,90,110]	③①④⑤②	②④①③	③①④⑤②②④①③

point, m is the number of servers. C_r and C_q are the node numbers of the routing rule and the queuing rule, respectively. The number of decisions in each of these two decision points does not exceed the total number of tasks submitted in the training phase, denoted as N.

The fitness evaluation characterizes the complexity of the training phase. The complexity in decision points is $O(n \cdot C_r) + O(m \cdot C_q) = O(n \cdot C_r + m \cdot C_q)$. The complexity of fitness evaluation is $O(N \cdot n \cdot C_r + N \cdot m \cdot C_q)$. Accordingly, the complexity of MTGPRR in the training phase is $O(g \cdot p \cdot (N \cdot n \cdot C_r + N \cdot m \cdot C_q))$. In the testing phase, the complexity of MTGPRR is equal to the complexity of the fitness evaluation, that is $O(N \cdot n \cdot C_r + N \cdot m \cdot C_q)$.

V. Experimental Design

This section presents the experimental design information used to evaluate the proposed MTGPRR algorithm. It includes the simulation configuration, baseline approaches, and parameter settings.

A. Simulation Configuration

For LEO satellite networks in our simulations, the real-time positions of LEO satellites are captured based on the two-line element set (TLE) data for Starlink. The elevation angle and distance of the LEO satellite relative to the ground are calculated using the SkyField library. The user devices are located near 33.9°S, 151.2°E. The selected satellites operate at altitudes between 540–570 km, with an inclination of around 53.1°. Similar to [48], the minimum elevation angle of LEO satellites is set as 5°. Due to the limitation of the elevation angle, handover occurs when the satellite moves out of service range over time. In terrestrial networks, the user devices can access three cloud servers and two MEC servers. The task requests are generated from the user devices over time according to the Poisson process. The computational tasks are heterogeneous, i.e., the computational tasks have different task sizes, workloads, and required deadlines. The task size follows a uniform distribution between 1 to 3 Mb. The workload follows a uniform distribution between 2 to 3 K cycles/bit. The deadline follows a uniform distribution between 5 to 11.

To conduct the performance testing and comparative analysis of MTGPRR, we set up twelve experimental scenarios. The names of scenarios are set to ⟨SC, UN, SN⟩, where SC denotes the simulation clock; the larger it is, the more tasks are generated during the simulation. UN denotes the user number, and SN denotes the satellite number. The experimental scenarios are set up as ⟨100, 800, 100⟩, ⟨100, 800, 120⟩, ⟨100, 800, 140⟩, ⟨100, 1000, 100⟩, ⟨100, 1000, 120⟩, ⟨100, 1000, 140⟩, ⟨150, 800, 100⟩, ⟨150, 800, 120⟩, ⟨150, 800, 140⟩, ⟨150, 1000, 100⟩, ⟨150, 1000, 120⟩, ⟨150, 1000, 140⟩. The simulation is run on Python 3.11 with a computer configuration of a 2.30 GHz Intel Xeon Gold 5118 CPU processor.

B. Baseline Approaches

Eight baseline approaches are listed below for comparison.

1) *FCFS [26]:* At the routing decision point, FCFS selects the server with the least computational tasks. At the queuing decision point, the computational task that arrives first is prioritized.
2) *EDF [26]:* EDF selects the server with the least computational tasks at the routing decision point. At the queuing decision point, it prioritizes the computational task with the earliest deadline.
3) *SOF [49]:* At the routing decision point, the shortest operation duration first (SOF) selects the server with the least computational tasks. At the queuing decision point, it prioritizes the computational task with the shortest processing duration.
4) *MOF [49]:* Maximum operation duration first (MOF) selects the server with the least computational tasks at the routing decision point. At the queuing decision point, it prioritizes the computational task with the longest processing duration.
5) *WMDC [28]:* Weighted mean of delay and consumption (WMDC) prioritizes low-cost computational tasks, where cost is a weighted mean of computational delay and energy consumption. The weighting factor is set as 0.5. At the routing decision point, we set WMDC to select the server with the least computing load.
6) *MTGP [36]:* MTGP evolves the routing rule and queuing rule via standard crossover.
7) *MTGP-CS [21]:* MTGP-CS changed the crossover operator of MTGP to a swapping crossover operator, i.e., swapping another rule while performing a crossover operation on one rule.
8) *MTGP-C^{2}S [37]:* MTGP-C^{2}S selects both the routing rule and the queuing rule for crossover, and then either the routing rule or the queuing rule is selected for swap.

For all the above algorithms, 30 independent runs are performed for each scenario. The routing rules and queuing rules evolved by MTGP, MTGP-CS, MTGP-C^{2}S, and proposed MTGPRR are tested on 30 unseen instances. The mean task success rate across the 30 test instances is reported as the test performance of the rules. Friedman's test with a significance level of 0.05 is used to rank the algorithms based on their performance. If Friedman's test gives significant results, we

TABLE III
PARAMETERS FOR DYNAMIC SCHEDULING SIMULATION

Parameters	Value	Parameters	Value
P^{D}	2 W	ρ	4
P^{S}	5 W	$B^{\text{T-T}}$	10 MHz
G^{D}	20 dB	u_0	1 m
G^{S}	20 dB	N_0	-203 dBm
g_0	-40 dB	k	1.3805×10^{-23} J/K
f	20 GHz	T_s	316.2278 K
$B^{\text{S-T}}$	500 MHz	E_b/N_1	9.6 dB
σ	0.9392	M	1.4125 dB
ω	0.2588	P_h	30 W
ψ	5 mm/h	ε	10^{-28}

TABLE IV
PARAMETERS FOR MTGPRR

Parameter	Value
Population size	200
Number of generations	51
Method for initializing population	Ramped-half-and-half
Initial minimum/maximum depth	1/4
Maximal depth	7
Crossover rate	0.8
Mutation rate	0.15
Reproduction rate	0.05
Reconstruction rate	0.15
Parent selection	Tournament selection with size 3

conduct the Wilcoxon rank-sum test with a significance level of 0.05 to determine which pairs of algorithms are significantly different. "+," "−," and "≈" indicate that the test performance is significantly better than, worse than, or similar to another algorithm, respectively. "Win, Draw, Lose" refers to the number of scenarios in which the comparison algorithm is statistically better, similar, or worse than MTGPRR.

C. Parameter Settings

The parameters covered in this work include the dynamic scheduling simulation related and MTGPRR algorithm related. The specific parameter settings are shown in Tables III and IV.

VI. RESULTS AND DISCUSSION

In this section, the advantages of MTGPRR over baseline methods are demonstrated by comparisons. Then, we specifically analyze the features, structure, and mathematical significance of the scheduling heuristics evolved by MTGPRR.

A. Performance of MTGPRR

The task success rate indicates the likelihood of processing computational tasks from the user devices before their deadlines. A high task success rate implies good performance in the scheduling heuristic.

Table V shows the mean task success rate of the scheduling heuristic evolved by the proposed MTGPRR and five manually designed scheduling heuristics in different scenarios. The proposed MTGPRR is significantly better than other methods in all scenarios. Among the five manually designed scheduling heuristics, WMDC has the best performance because it considers both the computational delay and energy consumption of

TABLE V
MEAN TASK SUCCESS RATE OF MTGPRR AND MANUALLY DESIGNED SCHEDULING HEURISTICS OVER 30 INDEPENDENT RUNS FOR DIFFERENT SCENARIOS

⟨SC,UN,SN⟩	FCFS	EDF	SOF	MOF	WMDC	MTGPRR
⟨100,800,100⟩	0.7225	0.7591	0.7448	0.7033	0.7639	0.9582
⟨100,800,120⟩	0.8421	0.8554	0.8692	0.8219	0.8722	0.9595
⟨100,800,140⟩	8939	0.9036	0.9133	0.8774	0.9209	0.9611
⟨100,1000,100⟩	0.5395	0.6433	0.6671	0.6213	0.6554	0.9530
⟨100,1000,120⟩	0.6939	0.7235	0.7359	0.7004	0.7491	0.9540
⟨100,1000,140⟩	0.8122	0.8440	0.8585	0.8154	0.8557	0.9553
⟨150,800,100⟩	0.6635	0.7039	0.7239	0.6697	0.7530	0.9350
⟨150,800,120⟩	0.7991	0.8145	0.8331	0.7804	0.8422	0.9368
⟨150,800,140⟩	0.8533	0.8539	0.8720	0.8155	0.8990	0.9403
⟨150,1000,100⟩	0.4030	0.5939	0.6342	0.5874	0.6355	0.9324
⟨150,1000,120⟩	0.5874	0.6482	0.6799	0.6509	0.6759	0.9335
⟨150,1000,140⟩	0.7002	0.7937	0.8032	0.8032	0.8185	0.9352

the task. Moreover, manually designed scheduling heuristics are susceptible to scenarios. For example, the mean task success rates of the manually designed scheduling heuristics all decrease as the simulation clock increases and increase as the number of LEO satellites increases. In contrast, MTGPRR can achieve high task success rates regardless of scenario variations. This indicates that MTGPRR has better stability than manually designed scheduling heuristics.

Table VI shows the test performance of MTGP, MTGP-CS, MTGP-C^{2}S, and MTGPRR over 30 independent runs. Based on the results of the Wilcoxon rank-sum test, it can be observed that MTGPRR significantly outperforms MTGP, MTGP-CS, and MTGP-C^{2}S in most scenarios. Regarding the mean task success rate, MTGPRR outperforms MTGP, MTGP-CS, and MTGP-C^{2}S in all scenarios. Furthermore, MTGP-CS and MTGP-C^{2}S outperform MTGP in most scenarios, although the advantage is insignificant. From the point of view of performance stability, the standard deviation of MTGPRR is smaller than that of MTGP, MTGP-CS, and MTGP-C^{2}S in most scenarios. This indicates that the scheduling heuristic evolved by MTGPRR has superior and stable performance. Moreover, MTGPRR runs slightly longer in all scenarios than MTGP, MTGP-CS, and MTGP-C^{2}S due to behavioral repetition test and rule reconstruction in MTGPRR. Overall, MTGPRR outperforms MTGP, MTGP-CS, and MTGP-C^{2}S, considering its significant advantages in test performance and insignificant additional computational cost.

Table VII shows the mean delay of MTGP, MTGP-CS, MTGP-C^{2}S, and MTGPRR over 30 independent runs. It can be seen that the mean delay of MTGPRR is significantly lower than that of MTGP, MTGP-CS, and MTGP-C^{2}S, except in scenarios ⟨100,1000,120⟩, ⟨100,1000,140⟩, ⟨150,800,120⟩, ⟨150,1000,100⟩, and ⟨150,1000,140⟩ where the mean delay of MTGPRR is not all significantly lower than that of MTGP, MTGP-CS, and MTGP-C^{2}S. Besides, in most scenarios, the mean delay shows an ascending trend as the number of users increases. Whereas, as the number of LEO satellites increases, the mean delay shows a decreasing trend.

Fig. 7 shows the violin plot of the mean task success rate of MTGP, MTGP-CS, MTGP-C^{2}S, and MTGPRR. The data comes from 30 independent runs of the four methods, where the best scheduling heuristic obtained from each evolution is

TABLE VI
MEAN (STANDARD DEVIATION) TASK SUCCESS RATE OF MTGP, MTGP-CS, MTGP-C^{2}S, AND MTGPRR OVER 30 INDEPENDENT RUNS FOR DIFFERENT SCENARIOS

⟨SC,UN,SN⟩	Task success rate				Runtime (s)			
	MTGP	MTGP-CS	MTGP-C^{2}S	MTGPRR	MTGP	MTGP-CS	MTGP-C^{2}S	MTGPRR
⟨100,800,100⟩	0.9522(0.0026)	0.9530(0.0020)(≈)	0.9533(0.0023)(≈)(≈)	0.9582(0.0022)(+)(+)(+)	23231	23755	23830	24921
⟨100,800,120⟩	0.9532(0.0019)	0.9555(0.0022)(+)	0.9556(0.0021)(+)(≈)	0.9595(0.0020)(+)(+)(+)	23644	24115	24033	25871
⟨100,800,140⟩	0.9556(0.0024)	0.9570(0.0018)(+)	0.9587(0.0025)(+)(+)	0.9611(0.0014)(+)(+)(+)	24202	24439	24521	26003
⟨100,1000,100⟩	0.9494(0.0023)	0.9500(0.0017)(≈)	0.9505(0.0027)(≈)(≈)	0.9530(0.0015)(+)(+)(+)	26392	27035	26877	28982
⟨100,1000,120⟩	0.9508(0.0015)	0.9516(0.0023)(≈)	0.9519(0.0023)(+)(≈)	0.9540(0.0020)(+)(+)(+)	26885	27544	27371	29822
⟨100,1000,140⟩	0.9524(0.0024)	0.9538(0.0021)(+)	0.9541(0.0016)(+)(≈)	0.9553(0.0015)(+)(+)(+)	27159	27734	27694	29996
⟨150,800,100⟩	0.9323(0.0018)	0.9342(0.0023)(+)	0.9343(0.0024)(+)(≈)	0.9350(0.0014)(+)(≈)(≈)	27935	28521	28439	30388
⟨150,800,120⟩	0.9334(0.0020)	0.9361(0.0019)(+)	0.9357(0.0018)(+)(≈)	0.9368(0.0031)(+)(≈)(≈)	28314	29051	28877	30747
⟨150,800,140⟩	0.9343(0.0018)	0.9382(0.0022)(+)	0.9381(0.0023)(+)(≈)	0.9403(0.0017)(+)(+)(+)	28981	29632	29599	31310
⟨150,1000,100⟩	0.9289(0.0021)	0.9299(0.0020)(≈)	0.9303(0.0018)(+)(≈)	0.9324(0.0018)(+)(+)(+)	30391	30787	30809	33392
⟨150,1000,120⟩	0.9299(0.0022)	0.9309(0.0023)(≈)	0.9307(0.0018)(≈)(≈)	0.9335(0.0018)(+)(+)(+)	30735	31306	31433	33933
⟨150,1000,140⟩	0.9309(0.0022)	0.9318(0.0025)(≈)	0.9324(0.0023)(+)(≈)	0.9352(0.0022)(+)(+)(+)	31355	31881	32039	34202
Win/Draw/Lose	0/0/12	0/2/10	0/2/10	N/A	N/A	N/A	N/A	N/A

TABLE VII
MEAN DELAY OF MTGP, MTGP-CS, MTGP-C^{2}S, AND MTGPRR OVER 30 INDEPENDENT RUNS FOR DIFFERENT SCENARIOS

⟨SC,UN,SN⟩	MTGP	MTGP-CS	MTGP-C^{2}S	MTGPRR
⟨100,800,100⟩	5.223	5.218(≈)	5.219(≈)(≈)	5.205(+)(+)(+)
⟨100,800,120⟩	5.183	5.172(+)	5.170(+)(+)	5.152(+)(+)(+)
⟨100,800,140⟩	5.105	5.103(≈)	5.100(≈)(≈)	5.086(+)(+)(+)
⟨100,1000,100⟩	5.375	5.369(≈)	5.374(≈)(≈)	5.348(+)(+)(+)
⟨100,1000,120⟩	5.306	5.305(≈)	5.300(≈)(≈)	5.295(+)(≈)(≈)
⟨100,1000,140⟩	5.279	5.269(+)	5.263(+)(+)	5.255(+)(+)(≈)
⟨150,800,100⟩	5.617	5.604(+)	5.603(+)(≈)	5.594(+)(+)(+)
⟨150,800,120⟩	5.542	5.530(+)	5.533(+)(≈)	5.523(+)(≈)(+)
⟨150,800,140⟩	5.520	5.518(≈)	5.520(≈)(≈)	5.500(+)(+)(+)
⟨150,1000,100⟩	5.774	5.770(≈)	5.760(+)(+)	5.756(+)(+)(≈)
⟨150,1000,120⟩	5.733	5.732(≈)	5.727(≈)(≈)	5.705(+)(+)(+)
⟨150,1000,140⟩	5.637	5.638(≈)	5.630(≈)(≈)	5.622(+)(+)(≈)
Win/Draw/Lose	0/0/12	0/2/10	0/4/8	N/A

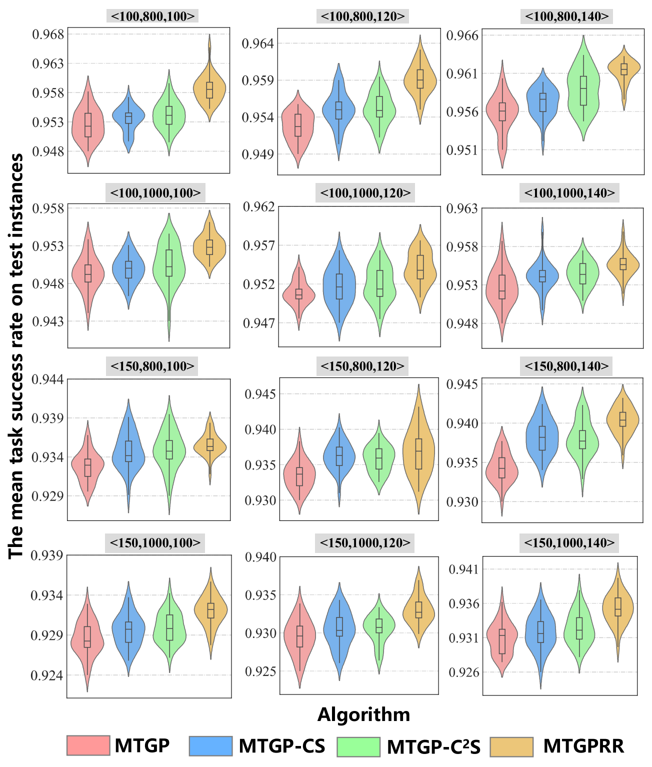

Fig. 7. Violin plot of the mean task success rate of MTGP, MTGP-CS, MTGP-C^{2}S, and MTGPRR on 30 unseen instances over 30 independent runs.

tested on 30 unseen instances for the mean task success rate. It demonstrates that the task success rates obtained by MTGPRR are distributed at a higher position than those obtained by MTGP, MTGP-CS, and MTGP-C^{2}S in all scenarios. In addition, the graphs of MTGPRR are more concentrated than those of MTGP, MTGP-CS, and MTGP-C^{2}S, which confirms the stable performance of MTGPRR.

Fig. 8 illustrates the mean processing time of each satellite versus the number of LEO satellites under MTGPRR. In the figure, the horizontal axis represents the number of different LEO satellites and the vertical axis represents the mean processing time of the satellites in different scenarios. The height of the bar in the figure is the mean of the processing time for all the satellites, the error bar is the standard deviation of the processing time, and the legend represents ⟨SC, UN⟩. It is easy to see that the mean processing time of each satellite decreases as the number of LEO satellites increases. In addition, the standard deviation of the processing time for satellites in all scenarios is small. This suggests that the MTGPRR method is able to efficiently allocate computational tasks to all computing resources in different scenarios, thus promoting load balancing.

The above results indicate that MTGPRR significantly outperforms MTGP, MTGP-CS, and MTGP-C^{2}S, and the stability of MTGPRR is better, which verifies the effectiveness of the introduction of the rule reconstruction operator.

B. Further Analysis

For the scheduling heuristics evolved by MTGPRR, we further analyze the feature frequency and the mathematical significance of the tree-based rules.

The greater the frequency of occurrence of a terminal in a scheduling rule, the more important this terminal is to the scheduling decision. Figs. 9 and 10 show the mean frequency of terminals in the routing rules and queuing rules evolved by MTGPRR in 30 independent runs, respectively. For the routing

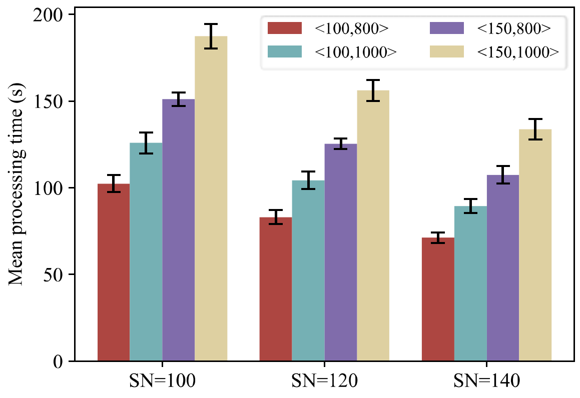

Fig. 8. Mean processing time of each LEO satellite for different SN.

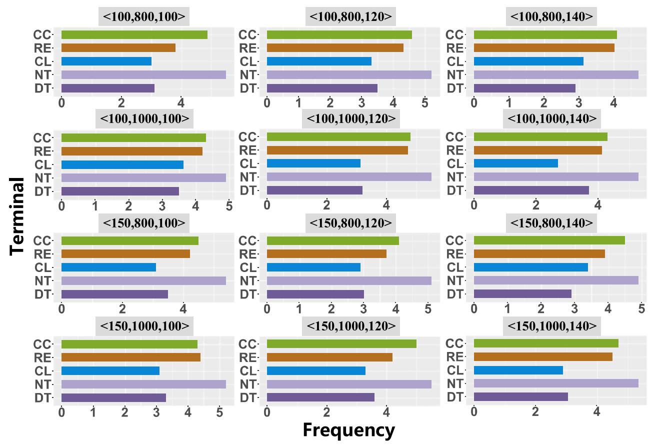

Fig. 9. Mean frequency of terminals in the routing rules.

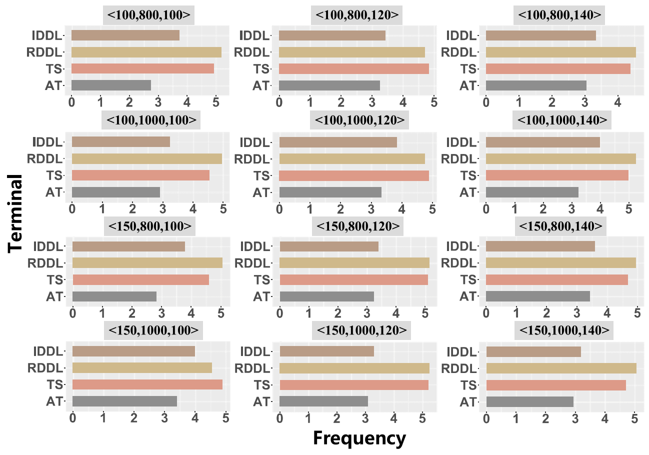

Fig. 10. Mean frequency of terminals in the queuing rules.

rules, the mean frequency of terminal NT is the largest in most scenarios, indicating that the number of tasks in a server plays a key role in selecting the server. In addition, the mean frequency of CC, DT, CL, and RE is also large and their role is not negligible.

For the queuing rules, the terminals TS and RDDL are frequently used among the all terminals. The mean frequency of terminal IDDL is smaller than RDDL, indicating more reference to remaining deadline when scheduling the processing order of tasks. The mean frequency of terminal AT is the smallest in all scenarios, indicating that the task arrival time is a minor factor, which echoes the result that the FCFS rule has the worst performance.

Both routing rules and queuing rules guide scheduling decisions in the form of priority functions. The priority function is represented as a tree structure. Therefore, by mining the mathematical significance of tree-based scheduling rules, one can better understand the logic of their decision and obtain the scheduling experience.

Fig. 11 shows the structures of two rules evolved by MTGPRR. For the routing rule, the expression for the priority function is given by

$$R_{\mathrm{r}} = \min\left\{\min\left\{\mathrm{RE{+}NT}, \frac{1}{\mathrm{CC}^2}\right\}\mathrm{CC}, \mathrm{NT}\right\}. \tag{15}$$

Since $1/\mathrm{CC}^2$ is smaller than RE+NT, the expression can be further written as $\mathrm{R_r} = \min\{1/\mathrm{CC}^2, \mathrm{NT}\}$. We can know that the routing rule mainly depends on the computational capability and the number of tasks of the server to perform the routing decisions. Moreover, the greater the computational capability or the smaller the number of tasks of the server, the greater its priority (the smaller the function value the greater the priority).

For the queuing rule, the expression for the priority function is given by

$$R_{\mathrm{q}} = \frac{\max\left\{\mathrm{TS}, \max\left\{\mathrm{IDDL} + \min\left\{\mathrm{TS\text{-}AT}, \frac{\mathrm{AT}}{\mathrm{IDDL}}\right\} - \mathrm{IDDL}, \mathrm{RDDL}\right\}\right\}}{\frac{\mathrm{RDDL}}{\max\{\min\{\max\{\mathrm{TS},\mathrm{IDDL}\},\mathrm{IDDL}\},\min\{\min\{\mathrm{AT},\mathrm{IDDL}\},1\}\}} - \mathrm{IDDL}}. \tag{16}$$

Since TS-AT is smaller than AT/IDDL, TS is smaller than IDDL, and 1 is smaller than both AT and IDDL, the expression can be further written as $R_q = [(\max\{\mathrm{TS}, \mathrm{RDDL}\})/(\mathrm{RDDL}/\mathrm{IDDL} - \mathrm{IDDL})]$. The expression shows that the queuing rule is related to TS, RDDL, and IDDL. The smaller TS and IDDL are, the smaller the function value is. Therefore, this queuing rule ranks the tasks with short computation time and short deadlines first.

In summary, the MTGPRR proposed in this work significantly outperforms the manually designed scheduling heuristics and the GP-based algorithms. The performance of the scheduling heuristic evolved by MTGPRR is more stable. In addition, the evolved scheduling heuristic has good interpretability, which is important for practical applications.

VII. Conclusion

In this work, we investigate the dynamic task scheduling problem in the integrated cloud–edge STNs, which aims to improve the task success rate of heterogeneous tasks. To solve this problem, we consider two types of decision points in the scheduling process at the same time, i.e., routing decision and queuing decision. Then, the scheduling heuristic with routing rule and queuing rule is developed to make efficient real-time decisions at these decision points, incorporating dynamic features related to multitier servers, computational tasks, and network environments. To simultaneously learn both routing rule and queuing rule from the scheduling process, we propose

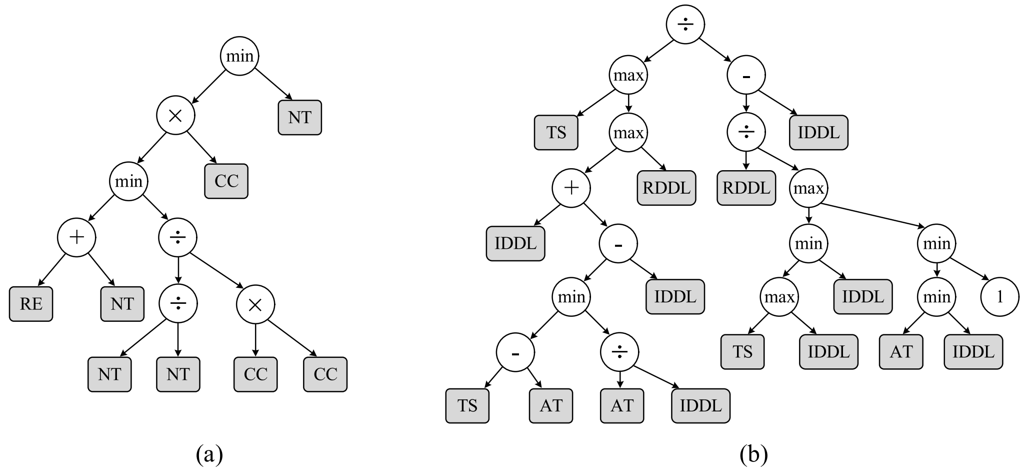

Fig. 11. Structures of two rules evolved by MTGPRR. (a) Routing rule. (b) Queuing rule.

the MTGPRR hyper-heuristic method, which introduces a selective reconstruction operator to increase the chance of matching good rules with other rules. Experimental results show that the test performance of the proposed MTGPRR is significantly better than that of existing state-of-the-art methods. In addition, the tree-based scheduling heuristic evolved by MTGPRR has good interpretability, which can tell us which features play a key role and the potential relationship between the features.

In the future, we will continue to explore other methods of rule reconstruction to further improve the performance of the algorithm. In addition, for multiclass computational tasks in STNs, we will develop the dynamic environment-based task classification method and propose the corresponding MTGP method to achieve parallel task scheduling. Meanwhile, we will consider the adaptive service pricing problem for operators and use the multiobjective GPHH to simultaneously maximize the operational revenue and task success rate.

References

[1] T. Kim, J. Kwak, and J. P. Choi, "Satellite edge computing architecture and network slice scheduling for IoT support," *IEEE Internet Things J.*, vol. 9, no. 16, pp. 14938–14951, Aug. 2022.

[2] L. Zhao, Y. Liu, A. Hawbani, N. Lin, W. Zhao, and K. Yu, "QoS-aware multihop task offloading in satellite–terrestrial edge networks," *IEEE Internet Things J.*, vol. 11, no. 19, pp. 31453–31466, Oct. 2024.

[3] X. Zhu and C. Jiang, "Delay optimization for cooperative multi-tier computing in integrated satellite-terrestrial networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 366–380, Feb. 2023.

[4] H. Huang, Q. Ye, and Y. Zhou, "6G-empowered offloading for realtime applications in multi-access edge computing," *IEEE Trans. Network Sci. Eng.*, vol. 10, no. 3, pp. 1311–1325, May/Jun. 2023.

[5] S. Wang and Q. Li, "Satellite computing: Vision and challenges," *IEEE Internet Things J.*, vol. 10, no. 24, pp. 22514–22529, Dec. 2023.

[6] R. Xie, Q. Tang, Q. Wang, X. Liu, R. Yu, and T. Huang, "Satellite-terrestrial integrated edge computing networks: Architecture, challenges, and open issues," *IEEE Netw.*, vol. 34, no. 3, pp. 224–231, May/Jun. 2020.

[7] Y. Gao, Z. Yan, K. Zhao, T. de Cola, and W. Li, "Joint optimization of server and service selection in satellite-terrestrial integrated edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 73, no. 2, pp. 2740–2754, Feb. 2024.

[8] H. H. Esmat, B. Lorenzo, and W. Shi, "Toward resilient network slicing for satellite–terrestrial edge computing IoT," *IEEE Internet Things J.*, vol. 10, no. 16, pp. 14621–14645, Aug. 2023.

[9] M. Zhao, X. Chang, Z. Wang, Q. Sun, G. Lv, and Y. Jin, "A space-air-ground enabled edge computing architecture for the Internet of Things," in *Proc. IEEE 4th Int. Conf. Electron. Technol. (ICET)*, 2021, pp. 752–757.

[10] Z. Sun, Y. Mei, F. Zhang, H. Huang, C. Gu, and M. Zhang, "Multi-tree genetic programming hyper-heuristic for dynamic flexible workflow scheduling in multi-clouds," *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2687–2703, Sep./Oct. 2024.

[11] P. Pirozmand, A. A. R. Hosseinabadi, M. Farrokhzad, M. Sadeghilalimi, S. Mirkamali, and A. Slowik, "Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing," *Neural Comput. Appl.*, vol. 33, pp. 13075–13088, Oct. 2021.

[12] X. Fu, Y. Sun, H. Wang, and H. Li, "Task scheduling of cloud computing based on hybrid particle swarm algorithm and genetic algorithm," *Cluster Comput.*, vol. 26, pp. 2479–2488, Oct. 2023.

[13] H. Wu, W. Shen, W. Lin, W. Li, and K. Li, "End-edge-cloud heterogeneous resources scheduling method based on RNN and particle swarm optimization," *IEEE Trans. Netw. Serv. Manag.*, early access, Nov. 27, 2024, doi: 10.1109/TNSM.2024.3507017.

[14] E. Celik and D. Dal, "A novel simulated annealing-based optimization approach for cluster-based task scheduling," *Cluster Comput.*, vol. 24, pp. 2927–2956, Dec. 2021.

[15] A. Kishor and C. Chakarbarty, "Task offloading in fog computing for using smart ant colony optimization," *Wireless Pers. Commun.*, vol. 127, no. 2, pp. 1683–1704, Nov. 2022.

[16] M. Zeedan, G. Attiya, and N. El-Fishawy, "Enhanced hybrid multi-objective workflow scheduling approach based artificial bee colony in cloud computing," *Computing*, vol. 105, no. 1, pp. 217–247, Jan. 2023.

[17] J. Pang, Y. Mei, and M. Zhang, "Multi-objective genetic-programming hyper-heuristic for evolving interpretable flexible job shop scheduling rules," in *Proc. IEEE Congr. Evolut. Comput. (CEC)*, 2024, pp. 1–8.

[18] Y. Yang, G. Chen, H. Ma, S. Hartmann, and M. Zhang, "Dual-tree genetic programming with adaptive mutation for dynamic workflow scheduling in cloud computing," *IEEE Trans. Evol. Comput.*, early access, Apr. 24, 2024, doi: 10.1109/TEVC.2024.3392968.

[19] B. Xu, K. Xu, B. Fei, D. Huang, L. Tao, and Y. Wang, "Automatic design of energy-efficient dispatching rules for multi-objective dynamic flexible job shop scheduling based on dual feature weight sets," *Mathematics*, vol. 12, no. 10, p. 1463, May 2024.

[20] S. Shady, T. Kaihara, N. Fujii, and D. Kokuryo, "A novel feature selection for evolving compact dispatching rules using genetic programming for dynamic job shop scheduling," *Int. J. Prod. Res.*, vol. 60, no. 13, pp. 4025–4048, Jul. 2022.

[21] M. Xu, Y. Mei, S. Zhu, B. Zhang, T. Xiang, F. Xiang, and M. Zhang, "Genetic programming for dynamic workflow scheduling in fog computing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 4, pp. 2657–2671, Jul./Aug. 2023.

[22] S. Shady, T. Kaihara, N. Fujii, and D. Kokuryo, "Automatic design of dispatching rules with genetic programming for dynamic job shop scheduling," in *Proc. IFIP Int. Conf. Adv. Prod. Manag. Syst.*, 2020, pp. 399–407.

[23] R. Braune, F. Benda, K. F. Doerner, and R. F. Hartl, "A genetic programming learning approach to generate dispatching rules for flexible shop scheduling problems," *Int. J. Prod. Econ.*, vol. 243, Jan. 2022, Art. no. 108342.

[24] Y. Yang, G. Chen, H. Ma, M. Zhang, and V. Huang, "Budget and SLA aware dynamic workflow scheduling in cloud computing with heterogeneous resources," in *Proc. IEEE Congr. Evolut. Comput.*, 2021, pp. 2141–2148.

[25] S. Wang, Y. Mei, and M. Zhang, "Explaining genetic programming-evolved routing policies for uncertain capacitated arc routing problems," *IEEE Trans. Evol. Comput.*, vol. 28, no. 4, pp. 918–932, Aug. 2024.

[26] S. Wang, X. Li, Q. Z. Sheng, and A. Beheshti, "Performance analysis and optimization on scheduling stochastic cloud service requests: A survey," *IEEE Trans. Netw. Services Manage.*, vol. 19, no. 3, pp. 3587–3602, Sep. 2022.

[27] K. Li, "Design and analysis of heuristic algorithms for energy-constrained task scheduling with device-edge-cloud fusion," *IEEE Trans. Sustain. Comput.*, vol. 8, no. 2, pp. 208–221, Apr.–Jun. 2023.

[28] F. Chai, Q. Zhang, H. Yao, X. Xin, R. Gao, and M. Guizani, "Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite IoT," *IEEE Trans. Veh. Technol.*, vol. 72, no. 6, pp. 7783–7795, Jun. 2023.

[29] J. Lou, Z. Tang, W. Jia, W. Zhao, and J. Li, "Startup-aware dependent task scheduling with bandwidth constraints in edge computing," *IEEE Trans. Mob. Comput.*, vol. 23, no. 2, pp. 1586–1600, Feb. 2024.

[30] X. Shen, "A hierarchical task scheduling strategy in mobile edge computing," *Internet Technol. Lett.*, vol. 4, no. 5, pp. 1–6, Sep. 2021.

[31] C. Zhang and J. Yang, "An energy-efficient collaborative offloading scheme with heterogeneous tasks for satellite edge computing," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 6, pp. 6396–6407, Nov./Dec. 2024.

[32] Y. Xu, L. Chen, Z. Lu, X. Du, J. Wu, and P. C. K. Hung, "An adaptive mechanism for dynamically collaborative computing power and task scheduling in edge environment," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3118–3129, Feb. 2023.

[33] S. D. Okegbile, B. T. Maharaj, and A. S. Alfa, "A multi-user tasks offloading scheme for integrated edge-fog-cloud computing environments," *IEEE Trans. Veh. Technol.*, vol. 71, no. 7, pp. 7487–7502, Jul. 2022.

[34] D. P. Muni, N. R. Pal, and J. Das, "A novel approach to design classifiers using genetic programming," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 183–196, Apr. 2004.

[35] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Learning strategies on scheduling heuristics of genetic programming in dynamic flexible job shop scheduling," in *Proc. IEEE Congr. Evolut. Comput. (CEC)*, 2022, pp. 1–8.

[36] Y. Yang, G. Chen, H. Ma, and M. Zhang, "Dual-tree genetic programming for deadline-constrained dynamic workflow scheduling in cloud," in *Proc. Int. Conf. Service-Oriented Comput.*, 2022, pp. 433–448.

[37] L. Zhu, F. Zhang, M. Feng, K. Chen, X. Zhu, and M. Zhang, "Crossover operators between multiple scheduling heuristics with genetic programming for dynamic flexible job shop scheduling," in *Proc. IEEE Congr. Evolut. Comput. (CEC)*, 2024, pp. 1–8.

[38] S. K. U. Zaman, T. Maqsood, A. Ramzan, F. Rehman, S. Mustafa, and J. Shuja, "Deadline-aware heuristics for reliability optimization in ubiquitous mobile edge computing," *Int. J. Data. Sci. Anal.*, 2023, to be published.

[39] Q. Tang, R. Xie, Z. Fang, T. Huang, T. Chen, R. Zhang, and F. R. Yu, "Joint service deployment and task scheduling for satellite edge computing: A two-timescale hierarchical approach," *IEEE J. Sel. Areas Commun.*, vol. 42, no. 5, pp. 1063–1079, May 2024.

[40] *Propagation Data and Prediction Methods Required for the Design of Earth-Space Telecommunication Systems Recommendation*, Rec. ITU-Rec. P. 618-12, Int. Telecommun. Union, Geneva, Switzerland, 2015.

[41] D. Zhou, M. Sheng, R. Liu, Y. Wang, and J. Li, "Channel-aware mission scheduling in broadband data relay satellite networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 5, pp. 1052–1064, May 2018.

[42] Y. Zhang et al., "Slow-movement particle swarm optimization algorithms for scheduling security-critical tasks in resource-limited mobile edge computing," *Future Gener. Comput. Syst.*, vol. 112, pp. 148–161, Nov. 2020.

[43] M. Merluzzi, N. d. Pietro, P. Di Lorenzo, E. C. Strinati, and S. Barbarossa, "Discontinuous computation offloading for energy-efficient mobile edge computing," *IEEE Trans. Green Commun. Netw.*, vol. 6, no. 2, pp. 1242–1257, Jun. 2022.

[44] B. Zhu, S. Lin, Y. Zhu, and X. Wang, "Collaborative hyperspectral image processing using satellite edge computing," *IEEE Trans. Mob. Comput.*, vol. 23, no. 3, pp. 2241–2253, Mar. 2024.

[45] H. Zhang, R. Liu, A. Kaushik, and X. Gao, "Satellite edge computing with collaborative computation offloading: An intelligent deep deterministic policy gradient approach," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 9092–9107, May 2023.

[46] T. Chen et al., "Learning-based computation offloading for IoRT through Ka/Q-Band satellite–terrestrial integrated networks," *IEEE Internet Things J.*, vol. 9, no. 14, pp. 12056–12070, Jul. 2022.

[47] J. Cao, S. Zhang, Q. Chen, H. Wang, M. Wang, and N. Liu, "Computing-aware routing for LEO satellite networks: A transmission and computation integration approach," *IEEE Trans. Veh. Technol.*, vol. 72, no. 12, pp. 16607–16623, Dec. 2023.

[48] Y. Zhou, J. Liu, R. Zhang, F. Liu, T. Huang, and T. Chen, "A congestion-aware handover scheme for LEO satellite networks," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, 2022, pp. 896–901.

[49] H. Chen, X. Li, and L. Gao, "A surrogate-assisted dual-tree genetic programming framework for dynamic resource constrained multi-project scheduling problem," *Int. J. Prod. Res.*, vol. 62, no. 16, pp. 5631–5653, 2024.

Changzhen Zhang received the M.S. degree from Yanshan University, Qinhuangdao, China, in 2022. He is currently pursuing the Ph.D. degree with the School of Reliability and Systems Engineering, Beihang University, Beijing, China.

His research interest includes wireless communication systems, reliability modeling, and queueing systems.

Jun Yang received the B.S. degree from Yantai University, Yantai, China, in 1998, and the Ph.D. degree from Chinese Academy of Sciences, Beijing, China, in 2006.

He is a Professor with the School of Reliability and Systems Engineering, Beihang University, Beijing. His scientific interests include wireless communication systems, reliability modeling, and applied statistics.

Ning Wang received the B.S. degree from China University of Geosciences, Beijing, China, in 2019. He is currently pursuing the Ph.D. degree with the School of Reliability and Systems Engineering, Beihang University, Beijing.

His research interests include network reliability, wireless sensor network optimization, and natural language processing.