aerospace

Article

Task Offloading with Data-Dependent Constraints in Satellite Edge Computing Networks: A Multi-Objective Approach

Ruipeng Zhang [1], Yanxiang Feng [1], Yikang Yang [1,*] and Xiaoling Li [2]

[1] Systems Engineering Institute, School of Automation Science and Engineering, Faculty of Electronic and Information Engineering, Xi'an Jiaotong University (XJTU), Xi'an 710049, China; zrprepeal@stu.xjtu.edu.cn (R.Z.); fengyanxiang@xjtu.edu.cn (Y.F.)

[2] School of Electronic and Control Engineering, Chang'an University (CHD), Xi'an 710049, China; xiaolingli@chd.edu.cn

* Correspondence: yangyk74@mail.xjtu.edu.cn

Abstract: By enabling a satellite network with edge computing capabilities, satellite edge computing(SEC) provides users with a full range of computing service. In this paper, we construct a multi-objective optimization model for task offloading with data-dependent constraints in an SEC network and aim to achieve optimal tradeoffs among energy consumption, cost, and makespan. However, dependency constraints between tasks may lead to unexpected computational delays and even task failures in an SEC network. To solve this, we proposed a Petri-net-based constraint amending method with polynomial complexity and generated offloading results satisfying our constraints. For the multiple optimization objectives, a strengthened dominance relation sort was established to balance the convergence and diversity of nondominated solutions. Based on these, we designed a multi-objective wolf pack search (MOWPS) algorithm. A series of adaptive mechanisms was employed for avoiding additional computational overhead, and a Lamarckian-learning-based multi-neighborhood search prevents MOWPS from becoming trapped in the local optimum. Extensive computational experiments demonstrate the outperformance of MOWPS for solving task offloading with data-dependent constraints in an SEC network.

Keywords: satellite edge computing; mobile edge computing; task offloading; data-dependent constraint; multi-objective optimization

Citation: Zhang, R.; Feng, Y.; Yang, Y.; Li, X. Task Offloading with Data-Dependent Constraints in Satellite Edge Computing Networks: A Multi-Objective Approach. *Aerospace* **2023**, *10*, 804. https://doi.org/10.3390/aerospace10090804

Academic Editor: Guanjun Xu

Received: 1 August 2023
Revised: 4 September 2023
Accepted: 11 September 2023
Published: 14 September 2023

1. Introduction

Satellite networks has recently received increasing attention and been regarded as an important component for future sixth-generation (6G) network architectures [1]. They have global coverage capability and high robustness, providing communication access for IoT devices widely distributed on the ground [2]. These characteristics mean that satellite networks remedy the defects of terrestrial networks in many scenarios [3–5]. Due to limited computing resources, IoT devices usually rely on cloud servers to process generated data beyond local capabilities [6]. Unfortunately, a long distance exists between cloud platforms and devices, leading to high communication latency, making it hard to deal with latency-sensitive applications. Meanwhile, with the rapid growth of IoT devices and data, transferring excessive data poses a challenge to network affordability [7], which also causes troubles in handling massive computation-intensive applications.

Mobile edge computing (MEC) makes up for the above shortcomings, providing a new computing paradigm by deploying computing resources close to the terminal device [8]. In MEC, all undesired transmissions involving long distance and excessive data between the cloud and terminals are avoided [9], which significantly alleviates network congestion and improves response time for latency-sensitive applications. The widespread IoT devices motivate the convergence of satellite networks and terrestrial networks. Furthermore, since embedding MEC servers in satellites can provide a computing service for IoT devices, even in remote and depopulated areas, SEC has received extensive attention [10–14].

Many computation-intensive requirements, such as scientific applications [15], large-scale image mosaicking for reconnaissance [16], and object detection in images [17], represented as applications, can be further decomposed into several tasks with data dependence. Figure 1 illustrates a task offloading scenario with data-dependent constraints in SEC. Processing these tasks can be extremely challenging because tasks are interdependent rather than isolated from each other. Subsequent tasks must wait until the results of all predecessor tasks are available. Thus, the offloading strategy plays a critical role in SEC systems. It determines the assignment of edge servers to specific tasks, improving resource utilization, enhancing coordination, and reducing energy consumption. The dynamic topology of satellite networks [18], combined with the constant changes in routing and intersatellite link (ISL) lengths, amplifies the challenge of this problem. Essentially, task offloading with data-dependent constraints in SEC is an NP-hard combinatorial optimization problem [19]. The complexity of the solution process is affected by the size of the problem, such as the number of satellites, applications, and tasks. Hence, the main challenges of this problem are creating an appropriate mathematical model and discovering an efficient algorithm.

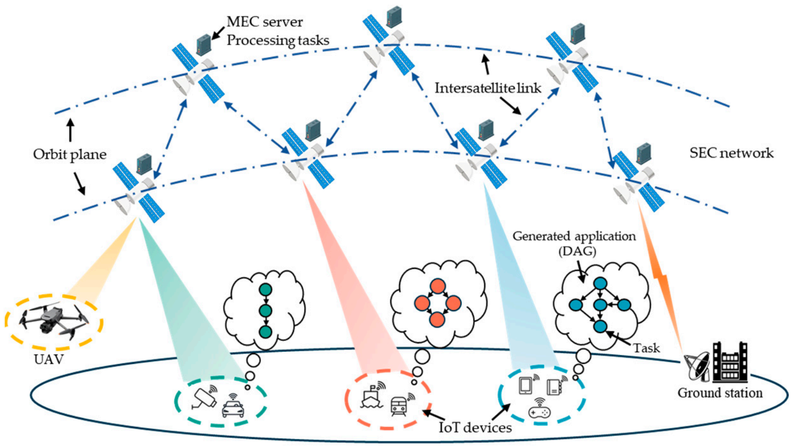


Figure 1. Task offloading in an SEC network with data-dependent constraints.

Numerous researchers have contributed significant progress to SEC. Zhang et al. [10] pioneered the SEC concept and designed a cooperative computation offloading model. Hu et al. [11] proposed a task offloading strategy based on Lyapunov optimization. Qin et al. [12] designed a search matching-based algorithm to solve the dynamic satellite selection problem that improves the load balance of satellite resources. Yu et al. [13] proposed a deep-imitation-learning-driven offloading and caching algorithm to achieve real-time decision making. Zhang et al. [14] presented a greedy-strategy-based task allocation algorithm for LEO satellite networks. Some scholars considered satellite–terrestrial links: Tang et al. [7] proposed a LEO satellite network that combined hybrid cloud and edge computing, taking into account satellite coverage time and computation capabilities. Song et al. [20] designed a framework for terrestrial–satellite IoT and divided computational offloading into ground and space segments. Additionally, some articles have focused on the software-defined networking and network function virtualization in satellite networks, such as [21–23]. To address high dynamics in LEO satellite networks, Wang et al. [24] proposed a time-expanded graph-based model, while Kim et al. [25] developed a model for satellite network topology that considered routing and satellite mobility. Zhang et al. [2] defined a task queue and adopted a multi-hop model to represent the transmission process of ISLs. To achieve a more accurate analysis of SEC characteristics, we developed a mixed-integer linear programming (MILP) model based on [1,2,25] that also considers satellite orbit elements and data dependencies between tasks.

The existing approaches for solving task offloading in SEC can be categorized into two categories: centralized and distributed. Centralized algorithms include heuristics [26], meta-heuristics [15,27,28], game algorithms [29], optimization methods [30], and reinforcement learning [3,31], all of which have demonstrated their effectiveness in various application scenarios. In contrast, distributed approaches, such as dynamic group learning distributed particle swarm optimization [32] and the multi-agent actor-critic reinforcement learning algorithm [33], cannot guarantee optimal results. As metaheuristic approaches can efficiently handle large-scale problems within polynomial time [34], we propose an MOWPS algorithm to address this issue.

To avoid unexpected computation delays and unsuccessful application execution, it is essential to consider the dependency constraints of tasks in SEC [26]. Ahmed et al. [28] proposed two offloading schemes (parallel and sequential) to address task dependencies, while Ma et al. [27] introduced a queue-based method for task offloading based on their allowable execution times. Chai et al. [3] modeled tasks with dependencies as directed acyclic graphs, then proposed an attention mechanism and proximal policy optimization collaborative algorithm to obtain the best offloading strategy. Hu et al. [35] proposed a hybrid genetic binary particle swarm optimization algorithm in which the task sequence is determined by the depth-first algorithm. Liu et al. [26] and Li et al. [15] both used heuristic algorithms to generate task sequences, and Liu et al. [36] proposed a ready queue approach for dynamic applications. However, obtaining the execution status of each satellite at any time in satellite networks can be costly. As a graphical and mathematical modeling tool [37,38], a Petri net [39] can effectively describe the coupling between applications, tasks, and edge servers. However, no studies have used Petri nets to model task offloading in SEC. In this paper, we first present a Petri-net-based constraint amending method to handle dependency constraints.

Additionally, task offloading in SEC involves multiple optimization objectives. When there are three or more objectives, the existing algorithms face challenges. First and foremost, as the number of objectives increases, almost all solutions in population become nondominated [40], making Pareto rank-based methods like NSGA-II invalid [41]. Secondly, solutions in high-dimensional space are typically sparsely distributed in the objective space [42], which makes it harder to maintain diversity. Finally, some metrics such as hypervolume may incur significant computational overhead. Prior work has been performed by a few scholars. For example, Ma et al. [27] used Pareto-optimal relations to obtain an archive set and introduced a grid method to maintain diversity. Li et al. [15] proposed a multi-swarm co-evolutionary mechanism in which each population focuses on different objectives and subsequently performs collaborative optimization. Aravanis et al. [43] devised a two-stage multi-objective optimization approach aimed at striking a balance between transmission rate and power consumption. Dai et al. [44] designed an improved discrete binary particle swarm optimization via jointly considering achievable rate and load balance. Gao et al. [45] used a competition-mechanism-based multi-objective PSO algorithm for satellite systems.

In this paper, we present a comprehensive algorithm for task offloading with data-dependent constraints in SEC. Firstly, we formulated a MILP model and proposed a Petri-net-based amending method to fulfill dependency constraints. Secondly, we introduced a multi-objective wolf pack search algorithm, which balances convergence and diversity, minimizes computational overhead with adaptive mechanisms, and uses a Lamarckian-learning-based multi-neighborhood search to break local optima. Finally, we conducted extensive numerical experiments to evaluate the algorithm's performance. The main contributions of this paper are summarized as follows:

- A MILP model is proposed for task offloading with data-dependent constraints in an SEC network. In addition, we consider a time-varying satellite network associated with orbit elements.
- We construct a Petri-net-based amender from a given candidate solution. This amender effectively describes the coupling between tasks with data dependencies and edge

servers. Furthermore, we introduce a Petri-net-based constraint amending method with polynomial time complexity, ensuring that offloading results conform to the constraints.
- A strengthened dominance relation sort is established to balance the convergence and diversity of nondominated solutions. It does not require a significant increase in computational cost, ensuring that the obtained non-dominated solution set is both close to the actual Pareto front and not overly concentrated.
- An MOWPS algorithm is presented that incorporates adaptive mechanisms to reduce computational overhead and uses Lamarckian-learning-based multi-neighborhood search to avoid local optima. Our experiments demonstrate that MOWPS outperforms existing algorithms in all testing instances.
- The remainder of this paper is organized as follows. Section 2 outlines the MILP model, while Section 3 presents the proposed MOWPS algorithm, including the Petri-net-based amending method, the strengthened dominance relation sort, the adaptive evolution mechanism, and the Lamarckian-learning-based multi-neighborhood search. Section 4 presents the numerical experiments and their results. Section 5 offers a discussion of the findings, and finally, Section 6 concludes this paper.

2. Problem Description and Modeling

In this section, we introduce a system model for task offloading in SEC. The detailed description of the SEC network, task, communication, and computation models is presented as follows. Table 1 provides explanations for the notations used in this paper.

Table 1. Notations.

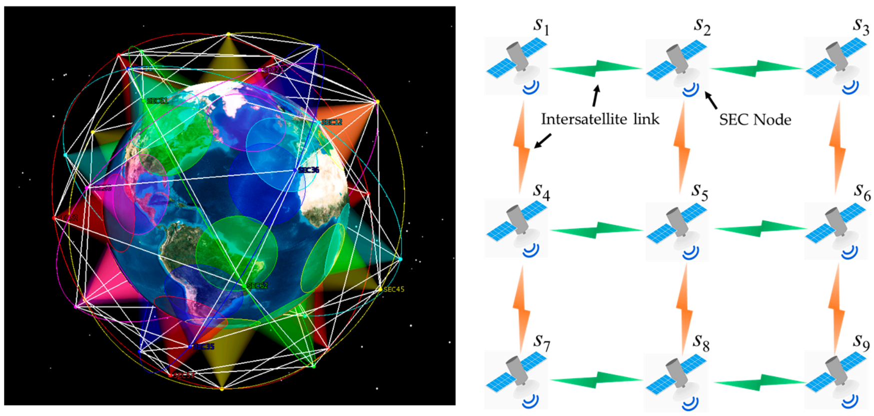
	Problem Descriptions
n	Total number of satellites.
v	Total number of applications.
$\mathbf{S}$	Set of satellites.
$\mathbf{U}$	Set of edge servers.
$l_{i,j}^{a}$	Intersatellite link distance between satellite s_i and s_j at time a.
L_a	Matrix representing satellite communication topology at time a.
$G_{i,j}^{a}$	Route between satellites s_i and s_j at time a.
$GS_{i,j}^{a}$	Shortest route between s_i and s_j at time a.
$\partial_{i,j}^{a}(d)$	Communication delay for transmitting data of size d from satellites s_i to s_j at time a.
$\mathbf{W}$	Applications, each of which can be decomposed into several tasks with dependencies.
w	Application, $w \in \mathbf{W}$.
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Directed acyclic graph representing data-dependent constraints.
m	Task.
T_p^{com}	Computation time for task m_p.
T_p^{avi}	The time that assigned edge server is available for executing m_p.
T_p^{ready}	The time that assigned edge server received all predecessors' results of m_p.
T_p^{start}	The time when edge server starts processing m_p.
T_p^{finish}	Finish time of m_p.

Table 1. Notations.

Solution Components	
$\theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$	Task assignments for all edge servers.
$\Delta = \{\aleph; \Upsilon\}$	Individual of MOWPS, where $\aleph$ implies the execution sequence of tasks and Υ represents assignment results.
Ψ_{all}	Set of individuals.
Ψ_{nd}	Set of nondominated individuals.
N_p	Number of individuals in a population.
(N, M_0, θ)	Petri net amender based on solution θ.
π_Δ	Transition sequence extracted from Δ.
Ψ_e	Set of elite individuals.
ϑ	Strengthened dominance relation sort.
ξ	Proportion of elite individuals.
ε	Proportion of extracted individuals.
$P_{preserve}$	Probability of retaining elements.
P_{bias}	Probability of biased selection.
φ_i	Utility value for Lamarckian learning method.
K_{max}	Maximum number of iterations.

2.1. SEC Network

The SEC network is depicted in Figure 2a; n satellites are represented as $\mathbf{S} = \{s_i, i \in \mathbb{N}_n\}$, which consists of Z_o adjacent orbital planes with Z_s satellites in each orbital plane, and we have $n = Z_o \times Z_s$. Each satellite is equipped with an edge server for executing tasks, and edge servers are represented by $\mathbf{U} = \{u_k, k \in \mathbb{N}_n\}$.



(**a**) (**b**)

Figure 2. (**a**) An SEC network with 36 satellites; (**b**) example of ISL connections.

Furthermore, satellites are continuously connected to their four adjacent satellites via ISLs [46], which comprise the two intra-plane satellites and the two inter-plane satellites. As illustrated in Figure 2b, satellite s_5 is connected to satellites s_2 and s_8 through intra-plane ISLs (represented in orange), while inter-plane ISLs (represented in green) connect satellites s_4 and s_6 to s_5.

The length of the ISLs are continuously changing as satellites move along their orbits. To model the SEC network accurately, we derived an approximate formula detailed in the supplementary file [47], by which the ISL distance $l^a_{i,j}$ between satellites s_i and s_j at time a can be obtained based on the satellite orbit elements. Specially, if no ISL exists between s_i

and s_j, we set $l^a_{I,j} = \infty$. Then, the communicate topology of satellites at time a is represented by a $n \times n$ matrix L_a, where $L_a[i, j] = l^a_{i,j}$.

2.2. Communication Model

In an SEC network, data transmission between satellites s_i and s_j at time a is accomplished through a *route* $G^a_{i,j}$ = <$s_i \rightarrow s_1{}' \rightarrow s_2{}' \rightarrow \ldots s_k{}' \rightarrow s_j$>, where intermediate satellites $s_i{}'$ and $s_{i+1}{}'$, $i \in \mathbb{N}_{k-1}$, are adjacent. To minimize routing delay, the shortest route $GS^a_{i,j}$ is adopted, whose length is denoted as $\mid GS^a_{i,j} \mid$, determined by the sum of ISL distances. Obviously, $GS^a_{i,j}$ can be obtained from matrix L_a using the Dijkstra algorithm [48]. To simplify, we assume that route in an SEC network remains fixed during the transmission of one task but may change across different tasks.

Then, the communication delay $\partial^a_{i,j}(d)$ can be obtained, which consists of two components, propagation delay and transmission delay:

$$\partial^a_{i,j}(d) = \frac{\left|GS^a_{i,j}\right|}{c} + \frac{d}{r} \tag{1}$$

where c represents the speed of light with a value of 3×10^5 km/s, d is the amount of transmitted data from s_i to s_j, and r is the communication capability of the ISL. Note that the communication delay is negligible when data are transmitted locally, i.e., $\partial^a_{i,j}(d) = 0$ for $i = j$.

2.3. Task Model

A total of v *Applications* is submitted by IoT devices, denoted as $\mathbf{W}$. Each application $w \in \mathbf{W}$ can be decomposed into several *tasks* with data dependencies, represented by a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where node set $\mathcal{V}$ represents tasks, and $\mid \mathcal{V} \mid$ is the number of tasks in application w. The set of directed arcs $\mathcal{E}$ denotes data dependencies between tasks. Each arc (m_p, m_j) in $\mathcal{E}$, $m_p, m_j \in \mathcal{V}$, is weighted by $d_{p,j}$, denoting that task m_j needs an intermediate result with amount $d_{p,j}$ from m_p. For task $m_p \in \mathcal{V}$, let pre_p and $succ_p$ be its predecessor and successor tasks, respectively. Task m_p can be executed only if all results of tasks in pre_p have been received. For completeness, we let d_p be the input data for entry task who has no predecessors, and each node $m_p \in \mathcal{V}$ is labeled with its workload e_p (Kcycles/Byte). Furthermore, we assume that tasks can be executed with any satellites in an SEC network.

Figure 3 shows the DAGs for two applications. Task m_1, labeled as 1, has a workload 1 Kcycles/Byte. Moreover, since d_1 = 13, it implies that 13 Gcycles (13 Mbyte $\times$ 1 Kcycles/Byte = 13 Gcycles) are needed to compute task m_1. The weight of arc (m_1, m_2) is 7, indicating that m_2 can only be executed after receiving a 7 Mbyte result from m_1.

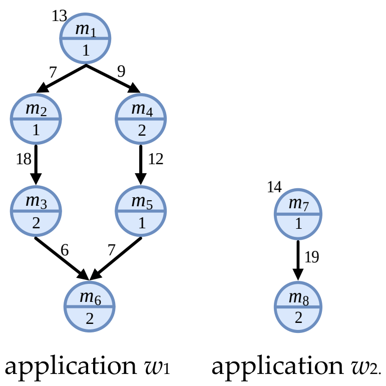


Figure 3. DAGs for applications.

2.4. Computation Model

According to Section 2.3, task can be executed only after receiving all predecessors' results, and each satellite can process only one task at a time. For task m_p, let $u_{\Pi(p)}$ be the assigned edge server; then the computation time T_p^{com} can be calculated as follows:

$$T_p^{com} = \frac{\sum\limits_{m_i \in pre_p} d_{i,p} \times e_p}{f_{\Pi(p)}} \tag{2}$$

where the computing capacity of edge server $u_k \in \mathbf{U}$ is fixed at f_k Gcycles/s. Specifically, for entry task m_p, we have $T_p^{com} = d_p \times e_p / f_{\Pi(p)}$.

Then for any task m_p, there are critical time parameters:

- T_p^{avi}: the time that the assigned edge server is available for executing m_p;
- T_p^{ready}: the time that the assigned edge server has received all predecessors' results of m_p;
- T_p^{start}: the time when the edge server starts processing m_p; and
- T_p^{finish}: the finish time of m_p.
- Since two situations exist when performing task m_p: (i) having received the required results before the assigned server is available, or (ii) waiting for predecessors' results although the edge server is already available, the above parameters are detailed as follows:

First, let $m_{\chi(p)}$ be the preceding task of m_p in the pending queue of assigned server. The edge server is available only after completing task $m_{\chi(p)}$, and we have

$$T_p^{avi} = T_{\chi(p)}^{finish} \tag{3}$$

For completeness, we set $T_p^{avi} = 0$ if $m_{\chi(p)}$ is the first task of edge server.

According to DAG, let $m_{\gamma(p)}$ be a predecessor task of m_p, i.e., $m_{\gamma(p)} \in pre_p$. After completing task $m_{\gamma(p)}$, the result of $m_{\gamma(p)}$ can be transmitted. Then the time of transferring $m_{\gamma(p)}$'s result for processing task m_p is

$$T_{\gamma(p),p}^{trans} = \partial_{\Pi(\gamma(p)),\Pi(p)}^{T_{\gamma(p)}^{finish}}(d_{\gamma(p),p}) \tag{4}$$

Given that task m_p can be executed only after receiving all results of pre_p, then the ready time T_p^{ready} can be calculated by (5).

$$T_p^{ready} = \max\left\{T_i^{finish} + T_{i,p}^{trans} \middle| \forall m_i \in pre_p\right\} \tag{5}$$

When m_p is an entry task, let $s_{I(p)}$ be m_p's access satellite, and we have $T_p^{ready} = \partial^0_{I(p),\Pi(p)}(d_p)$.

Afterward, Equation (6) is used to calculate the start time T_p^{start}.

$$T_p^{start} = \max\left\{T_p^{avi}, T_p^{ready}\right\} \tag{6}$$

Finally, the finish time T_p^{finish} is:

$$T_p^{finsih} = T_p^{start} + T_p^{com} \tag{7}$$

2.5. Optimization Objectives

There are three objectives considered in this paper.

2.5.1. Makespan

The makespan represents the maximum completion time of all tasks in **W**. A smaller makespan means more tasks can be processed in less time, resulting in a higher throughput.

$$Makespan = \max\left\{T_p^{finish} \middle| \forall m_p \in \mathbf{W}\right\} \tag{8}$$

2.5.2. User Cost

We assumed the SEC network uses AWS pricing [49], which is billed every second. The user cost has three components: (1) cost of using edge servers, (2) cost of transmitting data, and (3) cost of network occupancy. Then, we have

$$Cost = \sum_{m_p \in \mathbf{W}} T_p^{com} h_e^{\Pi(p)} + h_d \sum_{m_p \in \mathbf{W}} \sum_{m_i \in pre_p} d_{i,p} + h_o \sum_{m_p \in \mathbf{W}} \sum_{m_i \in pre_p} T_{i,p}^{trans} \tag{9}$$

where h_e^k is the price per unit time (\$/s) for server u_k; h_d is the price per unit of data (\$/MB) for transmission; and h_o is the network occupancy fee per unit time (\$/s).

2.5.3. Energy Consumption

The energy consumption of an SEC network includes three parts: communication, task processing, and standby energy consumption.

Firstly, the communication energy consumption can be expressed as:

$$Energy_1 = \sum_{m_p \in \mathbf{W}} \sum_{m_i \in pre_p} g_c T_{i,p}^{trans} \tag{10}$$

where g_c is the transmission power (W/s) of the ISLs.

Let g_w represent the energy consumption coefficient of the edge service's chip architecture, and recall that the computing capacity of server u_k is f_k Gcycles/s. Then the task processing energy consumption is

$$Energy_2 = \sum_{m_p \in \mathbf{W}} g_w {f_{\Pi(p)}}^2 \sum_{m_i \in pre_p} d_{i,p} e_p \tag{11}$$

Since edge servers in standby mode still consume energy, and g_s is the standby power (W/s) for an edge server, we have

$$Energy_3 = g_s \left(nMakespan - \sum_{m_p \in \mathbf{W}} T_p^{com} \right) \tag{12}$$

Finally, the total energy consumption can be expressed as

$$Energy = Energy_1 + Energy_2 + Energy_3 \tag{13}$$

2.6. Mathematical Formulation

Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be the decision vectors representing task assignment and task order on edge servers, respectively. The variable x_p^k in $\boldsymbol{x}$ is a decision variable such that $x_p^k = 1$ if task m_p is assigned to server u_k and $x_p^k = 0$ otherwise. Another variable $y_{p,j}^k$ in $\boldsymbol{y}$, equal to 1 if m_p is assigned to sever u_k before m_j, and $y_{p,j}^k = 0$ otherwise.

The mathematical model for task offloading with data-dependent constraints in SEC is presented below:

$$minimize\ \boldsymbol{F} = (Makespan, Cost, Energy) \tag{14}$$

$$\text{s.t.} \sum_{u_k \in \mathbf{U}} x_p^k = 1, \forall m_p \in \mathbf{W} \tag{15}$$

$$\sum_{m_p \in \mathbf{W}} x_p^k \geq 0, \forall u_k \in \mathbf{U} \tag{16}$$

$$\prod_{m_p \in \mathbf{W}, x_p^k = 1} T_p^{avi} = 0, \forall u_k \in \mathbf{U} \tag{17}$$

$$\sum_{u_k \in \mathbf{U}} (T_j^{avi} - T_p^{start} - T_p^{com}) y_{p,j}^k = 0, \forall m_p, m_j \in \mathbf{W} \tag{18}$$

$$y_{p,j}^k \left[T_j^{start} - T_p^{start} - T_p^{com} \right] \geq 0, \forall m_p, m_j \in \mathbf{W}, m_p \in pre_j, u_k \in \mathbf{U} \tag{19}$$

$$z_p - z_j + N_m \sum_{u_k \in \mathbf{U}} y_{p,j}^k + (N_m - 2) \sum_{u_k \in \mathbf{U}} y_{j,p}^k \leq N_m - 1, \forall m_p, m_j \in \mathbf{W}, m_p \neq m_j \tag{20}$$

$$x_p^k \in \{0,1\}, y_{p,j}^k \in \{0,1\}, \forall m_p, m_j \in \mathbf{W}, u_k \in \mathbf{U} \tag{21}$$

$$x_p^k x_j^k = y_{p,j}^k, \forall m_p, m_j \in \mathbf{W}, u_k \in \mathbf{U} \tag{22}$$

where Equation (14) is the objective function, and *Makespan*, *Cost*, and *Energy* can be calculated by Equations (8), (9), and (13), respectively. Equation (15) indicates that each task is executed by a specific edge server. Equation (16) indicates that some edge servers may have no assigned tasks. Equations (17) state that all edge servers are available simultaneously at time 0, and Equations (18) dictate that an edge server will be ready for the next task immediately upon completing the current one. The task precedence constraints are specified in Equation (19). Let z_p be the number of tasks that an edge server has performed before task m_p. Equation (20) is the traditional subtour elimination constraints. Equations (21) and (22) specify the domains of the involved variables.

3. Algorithm Description

In this section, we propose the MOWPS algorithm, which simulates the hunting process of wolves in nature and retains the mechanisms of "winner is king" and "survival of the strong" in wolf packs.

3.1. Encoding and Initialization

We use $\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots, \theta_n\}$ to denote a solution for task assignment, where $\theta_k \in \boldsymbol{\theta}$ contains all tasks need to be performed by edge server u_k. To obtain $\boldsymbol{\theta}$, we first give the definition of individual $\Delta = \{\aleph; \Upsilon\}$.

We define $\aleph = <\aleph[1], \aleph[2], \ldots, \aleph[v]>$ as a permutation of all tasks, which represents their execution sequence. We also define $\Upsilon = <\Upsilon[1], \Upsilon[2], \ldots, \Upsilon[v]>$ as a sequence of edge servers, indicating which edge server is assigned to each task. For example, an individual Δ can be represented in Figure 4, where $\aleph[1] = 1$ and $\Upsilon[1] = 3$ indicate that task m_1 is assigned to edge server u_3. We can also see that tasks m_8 and m_4 are both assigned to u_2, and m_8 should be executed before m_4. We can similarly derive the mapping relationships for other tasks.

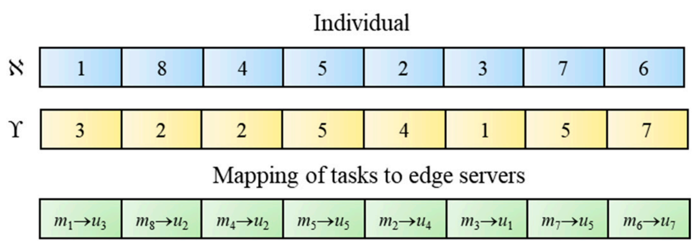


Figure 4. An individual for MOWPS with $v = 8$ and $n = 7$.

To obtain better guidance in the start-up phase, a heuristic-based initialization mechanism is used, which inserts four predefined individuals into the initial population. Specifically, individual Δ_1 is obtained by performing Horae [26]. Then, individuals $\Delta_2 - \Delta_4$ are obtained by a random mutation of Δ_1. Other individuals in the initial population are randomly generated. The resulting population, denoted as Ψ_{all}, consists of N_p individuals, and all non-dominated solutions in Ψ_{all} comprise the set Ψ_{nd}, which is updated after each iteration.

3.2. Amending

For any individual $\Delta = \{\aleph; \Upsilon\}$ in Ψ_{all}, Algorithm BD can be used to obtain a solution θ. However, it is possible that the solution does not comply with the precedence constraints.

Algorithm BD (Basic decoding method)

Input: an individual $\Delta = \{\aleph; \Upsilon\}$;
Output: a solution $\theta = \{\theta_k \mid k \in \mathbb{N}_n\}$;
1: Let each $\theta_k = \emptyset$, $k \in \mathbb{N}_n$;
2: **for** $i \in \mathbb{N}_v$
3: Insert task $m_{\aleph[i]}$ to the end of task sequence $\theta_{\Upsilon[i]}$.
4: **end**
5: Output solution θ

Example 1. *Consider the individual in Figure 4; the solution* $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_7\}$ *is obtained by Algorithm BD, where* $\theta_1 = \{m_3\}$, $\theta_2 = \{m_8, m_4\}$, $\theta_3 = \{m_1\}$, $\theta_4 = \{m_2\}$, $\theta_5 = \{m_5, m_7\}$, $\theta_6 = \emptyset$, *and* $\theta_7 = \{m_6\}$. *Then, we use the Wait-For Graph (WFG) [50]* $\mathcal{W} = (\mathcal{T}, \mathcal{A})$ *in Figure 5 to represent* $\boldsymbol{\theta}$, *where* $\mathcal{T}$ *contains all task nodes, the dotted arcs represent data-dependent constraints in Figure 3, and the task sequence of edge servers is indicated by colored solid arcs. According to the assigned results, task* m_8 *should be performed before* m_4, *task* m_7 *is before* m_8, *and* m_5 *is before* m_7; *thus, task* m_5 *must be executed before* m_4, *which contradicts the precedence constraint between tasks* m_4 *and* m_5. *It can be characterized by a loop composed of nodes* $\{m_4, m_5, m_7, m_8\}$, *which can be detected by the DFS algorithm [51].*

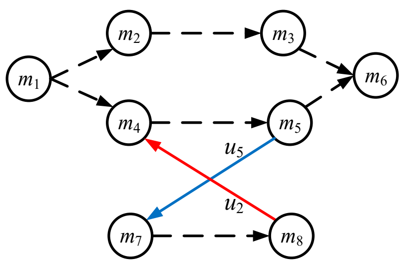


Figure 5. WFG of a constraint-violating solution.

To address this undesired phenomenon, we propose a Petri-net-based amending method to ensure that the precedence constraints are satisfied between any two consecutive tasks. Readers are expected to be familiar with the properties and definition of Petri nets, which can be found in the supplemental file [47,52].

For any individual Δ, we obtain solution θ using Algorithm BD; then, the Petri-net-based amender (N, M_0, θ) can be established as follows:

Step 1: For each application $w_q \in \mathbf{W}$, let Petri net $(\alpha_q, M_q) = (P_q, T_q, F_q, M_q)$, where $P_q = \{p_{q,e}\} \cup \{p_{q,h} \mid m_h \in w_q\}$, $T_q = \{t_{q,h} \mid m_h \in w_q\}$, $F_q = \{(p_{q,h}, t_{q,h}),(t_{q,i}, p_{q,h}) \mid m_h \in w_q, m_i \in pre(m_h)\} \cup \{(t_{q,h}, p_{q,e}) \mid m_h \in w_q \wedge succ(m_h) = \emptyset\}$, $M_q(p_{q,h}) = 1$ if $pre(m_h) = \emptyset$, and $M_q(p) = 0$ otherwise.

Step 2: For each satellite $s_k \in \mathbf{S}$, we define Petri net $(\beta_k, M_k) = (P_k, T_k, F_k, M_k)$, where $P_k = \{p_j \mid \forall sat(u_j)=s_k\}$ denotes the *resource* place of s_k, $T_k = \{t_{q,h} \mid m_h \in w_q, \forall m_h \in \theta_j, p_j \in P_k\}$, $F_k = \{(t, p), (p, t) \mid t \in T_k, p \in P_k\}$, $M_k(p) = 1, \forall p \in P_k$.

Step 3: The amender (N, M_0, θ) can be constructed by combining all (α_q, M_q) and (β_k, M_k):

$$(N, M_0, \boldsymbol{\theta}) = \oplus_{w_q \in \mathbf{W}}(\alpha_q, M_q) \oplus_{s_k \in \mathbf{S}} (\beta_k, M_k) \tag{23}$$

where operator $\oplus$ indicates the combination of two Petri nets via their common places and transitions.

The subnet (α_q, M_q) generated by Step 1 represents the data dependencies of tasks in w_q. Each transition $t_{q,h} \in T_q$ corresponds to a specific task m_h in w_q. The *sink* place is represented by $p_{q,e}$, while the intermediate states are represented by other places in P_q. Initially, *source* place $p_{q,s}$ (where m_s is the entry task of w_q) is marked by a unique token. When a token flows into the sink place $p_{q,e}$, it signifies the complication of each task in w_q.

Step 2 constructs the subnet (β_k, M_k), which represents the task assignments of edge servers in s_k. Each transition $t_{q,h} \in T_k$ is associated with a resource place $p_j \in P_k$ through a pair of directed arcs ($t_{q,h}$, p_j) and (p_j, $t_{q,h}$). This connection indicates that task m_h is assigned to edge server u_j. Each resource places p_j in P_k always contains one token.

Example 2. *Given the individual in Figure 4 with solution* $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_7\}$*, where* $\theta_1 = \{m_3\}$*,* $\theta_2 = \{m_8, m_4\}$*,* $\theta_3 = \{m_1\}$*,* $\theta_4 = \{m_2\}$*,* $\theta_5 = \{m_5, m_7\}$*,* $\theta_6 = \emptyset$*, and* $\theta_7 = \{m_6\}$*. Figure 6a shows the Petri net* (α_q, M_q) *for each* $w_q \in \mathbf{W}$*. Then we have* $T_1 = \{t_{1,3}\}$*,* $T_2 = \{t_{2,8}, t_{1,4}\}$*,* $T_3 = \{t_{1,1}\}$*,* $T_4 = \{t_{1,2}\}$*,* $T_5 = \{t_{1,5}, t_{2,7}\}$*, and* $T_7 = \{t_{1,6}\}$ *according to* $\boldsymbol{\theta}$*, and the Petri net* (β_k, M_k) *for each satellite* $s_k \in \mathbf{S}$ *is shown in Figure 6b. After composing all above* (α_q, M_q) *and* (β_k, M_k)*, the Petri-net-based amender* (N, M_0, $\boldsymbol{\theta}$) *is shown in Figure 6c.*

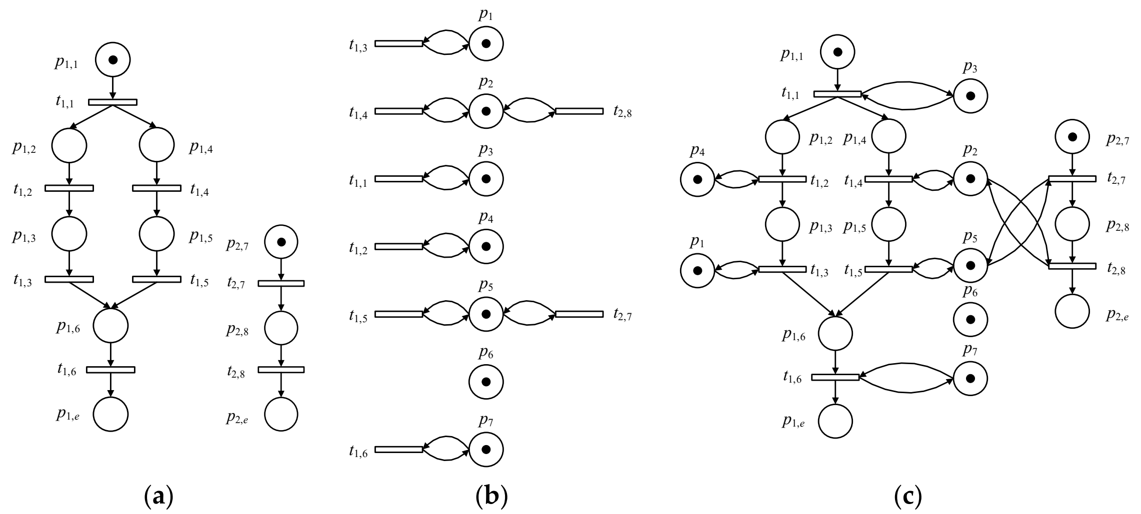


Figure 6. Amender construction process. (**a**) Petri nets (α_1, M_1) and (α_2, M_2); (**b**) Petri net (β_k, M_k) for each satellite $s_k \in \mathbf{S}$; (**c**) Petri-net-based amender (N, M_0, θ).

In (N, M_0, θ), each transition can be fired exactly once under initial marking M_0 or the state M reached from M_0. The eventually marking, where all tokens are collected in $p_{q,e}$ for all $\forall w_q \in \mathbf{W}$, is denoted as M_E. From M_0 to M_E, each transition in (N, M_0, θ) is fired once, resulting in the completion of all tasks.

For an individual $\Delta = \{\aleph; \Upsilon\}$ and its amender (N, M_0, θ), transition $t_{q,h}$ corresponds to task m_h, where $m_h \in w_q$. The transition sequence π_Δ can be obtained by replacing each task in $\aleph$ with the corresponding transition. For instance, the transition sequence for individual in Figure 4 is $\pi_\Delta = t_{1,1}t_{2,8}t_{1,4}t_{1,5}t_{1,2}t_{1,3}t_{2,7}t_{1,6}$. The sequence π_Δ is *feasible* if M_E can be reached from M_0 through π_Δ, which is represented as $M_0[\pi_\Delta > M_E)$. Then, firing transitions in the sequence π_Δ sequentially can accomplish all tasks in $\mathbf{W}$, and the solution is feasible. Thus, a feasible individual can be obtained by finding a transition sequence π_Δ that satisfies $M_0[\pi_\Delta > M_E)$.

After the aforementioned analysis, we can propose an algorithm to amend individuals that violate precedence constraints using amender (N, M_0, θ).

Algorithm AM (Amending method)

Input: a candidate individual Δ = {ℵ; ϒ};
Output: a feasible individual Δ* = {ℵ*; ϒ*};
1: Obtain solution θ from Δ by algorithm BD;
2: Construct Petri-net-based amender (N, M_0, θ);
3: Generate the transition sequence π_Δ from Δ;
4: **for** n = 1 to v
5: **while** ($\pi_\Delta[n]$ is disabled under M_{n-1})
6: Move $\pi_\Delta[n]$ to the end of π_Δ;
7: Move ϒ[n] to the end of ϒ;
8: **end**
9: Let $M_{n-1}[\pi_\Delta[n] > M_n$;
10: **end**
11: Let π_Δ* = π_Δ and ϒ* = ϒ;
12: Obtain a permutation of tasks ℵ* from π_Δ*;
13: Output Δ* = {ℵ*; ϒ*};

First, we create amender (N, M_0, θ) and transition sequence π_Δ for candidate individual Δ = {ℵ; ϒ} and its corresponding solution θ. Next, we check whether $\pi_\Delta[n]$ is disabled under M_{n-1} in sequential order. If $\pi_\Delta[n]$ is disabled, we move $\pi_\Delta[n]$ and ϒ[n] to the end of π_Δ and ϒ, respectively. We repeat this process until we find an enabled transition. Then, we fire the enabled transition, generate a new marking M_n (i.e., $M_{n-1}[\pi_\Delta[n] > M_n$), and move on to the next detection. After performing the iteration in Lines 4−10, we obtain a feasible π_Δ* and a new ϒ*, and we have $M_0[\pi_\Delta{}^* > M_E$. After converting π_Δ* to a task permutation ℵ*, we obtain a feasible individual Δ* = {ℵ*; ϒ*} corresponding to a solution satisfying the precedence constraints. The effectiveness and computational complexity of Algorithm AM are established by the following proposition.

Proposition 1. *Algorithm AM is effective and has polynomial time complexity.*

Proof of Proposition 1. In (N, M_0, θ), each transition $t_{q,h}$ can be fired exactly once under some state M before reaching the final marking M_E. This means that there is at least one enabled transition under M. Algorithm AM can always find an enabled transition through its iteration. Finally, M_E is reached, and a feasible transition sequence π_Δ* is obtained (i.e., $M_0[\pi_\Delta{}^* > M_E$). Thus, the solution Δ* generated from π_Δ* is feasible.

The entire algorithm repeats v times. In the n-th iteration, at most ($v - n$) transitions are checked. Thus, the complexity of Algorithm AM is $O(v^2)$, i.e., Algorithm AM is polynomial. □

Example 3. *Figure 7 depicts the amending process of individual Δ = {ℵ; ϒ} in Figure 4. The amender (N, M_0, **θ**) and transition sequence $\pi_\Delta = t_{1,1}t_{2,8}t_{1,4}t_{1,5}t_{1,2}t_{1,3}t_{2,7}t_{1,6}$ are shown in Figure 7a. Starting with M_0, we fire the enabled transition $t_{1,1}$ and obtain a new marking M_1 (i.e., $M_0[t_{1,1} > M_1$), as shown in Figure 7b. The fired transitions are marked in green. However, as the process continues, the second transition $t_{2,8}$ becomes disabled under M_1, shown in Figure 7c. Therefore, we move $t_{2,8}$ and its corresponding task 2 to the end of π_Δ and ϒ, respectively, resulting in a new transition sequence $\pi_\Delta{}' = t_{1,1}t_{1,4}t_{1,5}t_{1,2}t_{1,3}t_{2,7}t_{1,6}t_{2,8}$ and ϒ′ = <3, 2, 5, 4, 1, 5, 7, 2>. We then fire the next enabled transition $t_{1,4}$ in $\pi_\Delta{}'$ and obtain marking M_2 (i.e., $M_1[t_{1,4} > M_2$), as shown in Figure 7d. After sequentially firing all remaining transitions, we reach the final marking M_E, as shown in Figure 7e. Therefore, $\pi_\Delta{}'$ is a feasible transition sequence from M_0 to M_E. Based on $\pi_\Delta{}'$ and ϒ′, we can obtain a feasible individual Δ* = {ℵ*; ϒ*}, where ℵ* = <1, 4, 5, 2, 3, 7, 6, 8> and ϒ* = <3, 2, 5, 4, 1, 5, 7, 2>.*

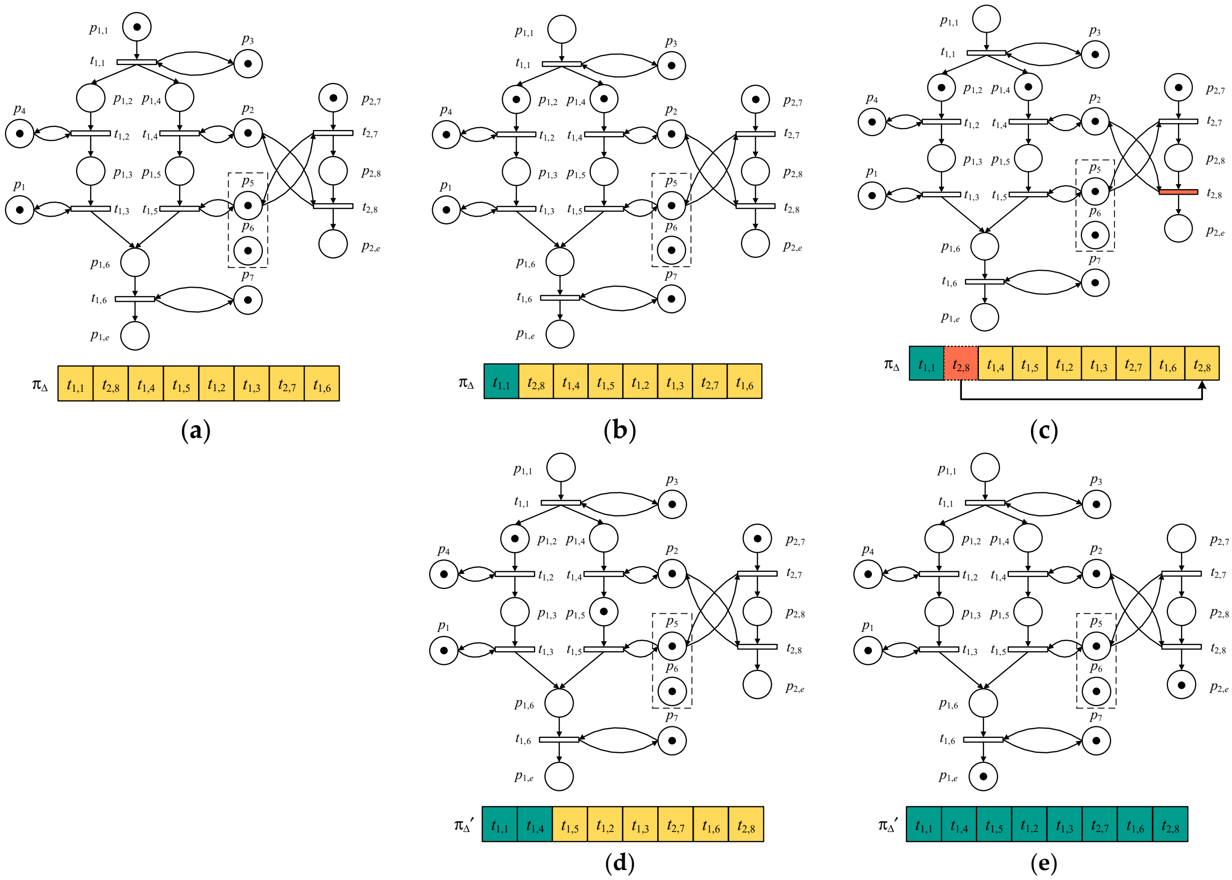


Figure 7. Amending process. (**a**) Initial marking M_0. (**b**) $t_{1,1}$ is fired, $M_0[t_{1,1} > M_1$. (**c**) $t_{2,8}$ is disabled under M_1. (**d**) $t_{1,4}$ is fired, and $M_1[t_{1,4} > M_2$. (**e**) All transitions are fired, and we reach final marking M_E.

3.3. Strengthened Dominance Relation Sort

Considering the multiple objectives in SEC, the traditional Pareto optimality treats all nondominated solutions equally, making it challenging to evaluate each nondominated individual [53]. Additionally, striking a balance for nondominated solutions between convergence and diversity also poses a challenge [54]. To address these issues, a new measure called strengthened dominance relation [55] is introduced. We improved it by proposing a strengthened dominance relation sort (SDRS) and embedded it in our MOWPS. The fundamental principles of this approach are described below.

Assume a problem with K objective functions $\boldsymbol{f}(\boldsymbol{x})$, denoted as $f_c(\bullet)$ for $c = \mathbb{N}_K$. We define the solution space as $\Phi \subset \mathbf{R}^b$, which contains all solution vectors $\boldsymbol{x}$, with dimensionality b.

Convergence degree: We can obtain a metric for the convergence degree of a solution vector $\boldsymbol{x} \in \Phi$ using (24):

$$Con(\boldsymbol{x}) = \sum_{c=1}^{K} e^{(\phi-1)} f_c(\boldsymbol{x}) \tag{24}$$

where ϕ is a convergence degree factor. As ϕ increases, the convergent pressure is strengthened, while weakening diversity.

Niche size: Given a set of solutions $\boldsymbol{Q}$, we can calculate the angle between any pair of solutions $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\boldsymbol{Q}$ as $\theta_{\boldsymbol{xy}}$ = arccos($\boldsymbol{f}(x)$, $\boldsymbol{f}(y)$). Then, the niche size $\overline{\theta}$ is defined as the [Υ | $\boldsymbol{Q}$ |]-th smallest element of

$$\left\{ \min_{y \in Q | \{x\}} \theta_{\boldsymbol{xy}} | \boldsymbol{x} \in \boldsymbol{Q} \right\} \tag{25}$$

where Υ is the niche size factor, $\Upsilon \in (0, 1)$, and [•] is the rounding down operation. As Υ increases, the dominated area expands, and the convergent pressure intensifies.

To calculate Con(x) and $\overline{\theta}$, we need to first normalize each $\boldsymbol{x} \in \boldsymbol{Q}$ with respect to the ideal point and nadir point of Q. The ideal point is a vector containing the optimal value of each objective function, while the nadir point is a vector consisting of the worst value.

Strengthened dominance value: The strengthened dominance value $D(\boldsymbol{x}, \boldsymbol{y})$ represents how much solution $\boldsymbol{x}$ dominates $\boldsymbol{y}$, as calculated by (26). Suppose that it is a multi-objective minimization problem.

$$D(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} \max\{0, Con(\boldsymbol{y}) - Con(\boldsymbol{x})\}, & \theta_{xy} \leq \overline{\theta} \vee \boldsymbol{x} \text{ Pareto dominates } \boldsymbol{y} \\ \max\left\{0, Con(\boldsymbol{y}) - \frac{\theta_{xy}}{\overline{\theta}} Con(\boldsymbol{x})\right\}, & \theta_{xy} > \overline{\theta} \end{cases} \tag{26}$$

where $D(\boldsymbol{x}, \boldsymbol{y})$ is nonnegative (i.e., $D(\boldsymbol{x}, \boldsymbol{y})$ is necessarily zero if $\boldsymbol{x}$ is inferior to $\boldsymbol{y}$).

Example 4. *For a normalized solution vector of $\boldsymbol{x}$ = [0.5, 0.5], the dominated regions are depicted in green in Figure 8 under different values of ϕ and γ. The results in Figure 8a,b indicate that as the value of ϕ increases, the convergent pressure becomes stronger. Conversely, Figure 8b,c show that a smaller value of γ results in better diversity. It is worth noting that the upper right corner is always dominated due to the incorporation of Pareto dominance in Equation (26).*

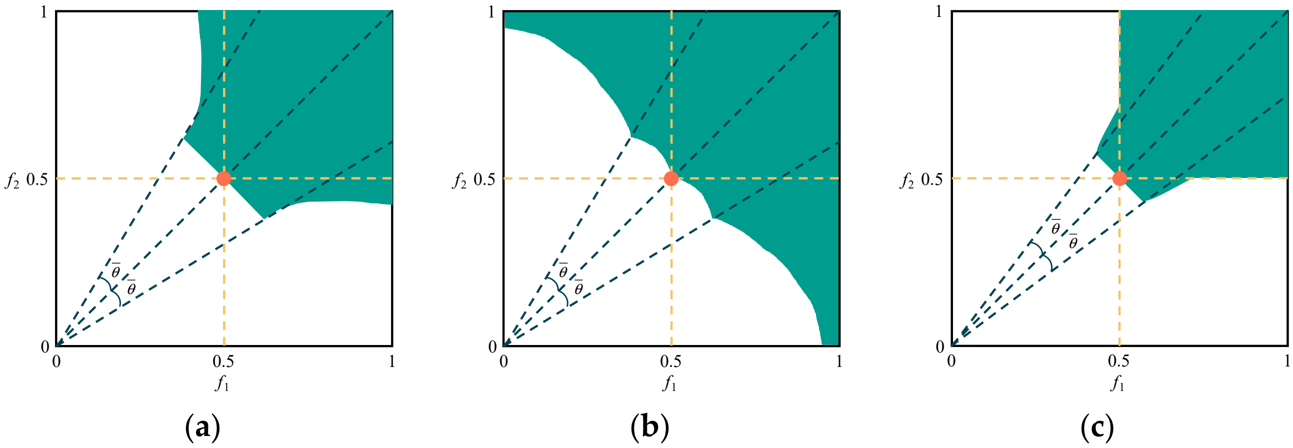


Figure 8. Proportion of the area dominated by [0.5, 0.5]. (**a**) ϕ = 1, Υ = 0.5; (**b**) ϕ = 1.8, Υ = 0.5; (**c**) ϕ = 1, Υ = 0.3.

SDRS value: Each solution $\boldsymbol{x}$ is assigned an SDRS value $\vartheta(\boldsymbol{x})$, which reflects its degree of domination over others in the entire solution set $\boldsymbol{Q}$. Thus, we have:

$$\vartheta(\boldsymbol{x}) = \frac{\sum_{y \in Q | \{x\}} D(\boldsymbol{x}, \boldsymbol{y})}{|\boldsymbol{Q}| - 1} \tag{27}$$

A higher value of $\vartheta(\boldsymbol{x})$ indicates better solution quality, which helps to generate diverse solutions near the Pareto frontier. To ensure a successful search process, we use an adaptive mechanism with the following formulas:

- For the parameter ϕ: $\phi = 1.8 - 1.3 \times K_{cur}/K_{max}$

- For the parameter Υ: $\Upsilon = 0.6 - 0.3 \times K_{cur}/K_{max}$

where K_{cur} is the number of iterations completed and K_{max} is the maximum number of iterations.

Next, we sort individuals based on their $\vartheta(x)$ value and select the top $[\xi \times N_p]$ individuals in each iteration to form the *elite set* Ψ_e. The parameter ξ represents the elite proportion.

3.4. Population Evolution

In a wolf pack, elite wolves roam to hunt for prey. Let us first explain the roaming process. Each elite wolf is represented by an individual denoted as $\Delta_e = \{\aleph_e; \Upsilon_e\} \in \Psi_e$. We randomly select N_c distinct individuals from Ψ_e, where N_c is determined as $N_c = [\varepsilon \times |\Psi_e|]$. For each selected individual $\Delta_s = \{\aleph_s; \Upsilon_s\}$, we generate two individuals using the following method:

Step 1: Initialize an empty individual $\Delta_{s1} = \{\aleph_{s1}; \Upsilon_{s1}\}$.

Step 2: For $i \in \mathbb{N}_v$, set $\aleph_{s1}[i] = \aleph_s[i]$ if $rand < P_{reserve}$, where $P_{reserve}$ is the given probability. Repeat this step for Υ_{s1}.

Step 3: Place all unassigned tasks in $\mathbf{W}$ into empty slots in $\aleph_{s1}$ following the order they appear in $\aleph_s$. Then, fill each empty slot in Υ_{s1} with the corresponding position element in Υ_s.

Step 4: Check each element in Υ_{s1} and replace any edge servers with random edge servers.

Step 5: Reverse the role of Δ_e and Δ_s and repeat steps 1−4 to create another individual Υ_{s2}.

For each elite individual Δ_e in Ψ_e, we generate $2N_c$ individuals. From these individuals, the optimal one is chosen to replace Δ_e, and the whole elite set Ψ_e is updated accordingly.

Once the elite wolves have finished roaming, all the wolves respond to the call of a wolf that has found prey. To prevent the population from converging on a single individual and to increase the diversity of non-dominant solutions, we designed the following *multiway attacking* mechanism.

For each individual $\Delta = \{\aleph; \Upsilon\}$ in Ψ_{all}, we select a non-dominated solution $\Delta_d = \{\aleph_d; \Upsilon_d\}$ from Ψ_{nd} as the target. We then randomly generate an interval $[i, j] \subseteq [1, v]$, and remove a consecutive block of tasks $<\aleph[i], \ldots, \aleph[j]>$ from $\aleph$ and reinsert them according to their order in $\aleph_d$. The elements in $<\Upsilon[i], \ldots, \Upsilon[j]>$ are divided into three groups by comparing them with elements in the same position in Υ_d. The elements in the smaller, bigger, and equal groups are then subjected to plus 1, minus 1, and no operations, respectively. The result is expressed as $\Delta' = \{\aleph'; \Upsilon'\}$. Figure 9 illustrates the above process.

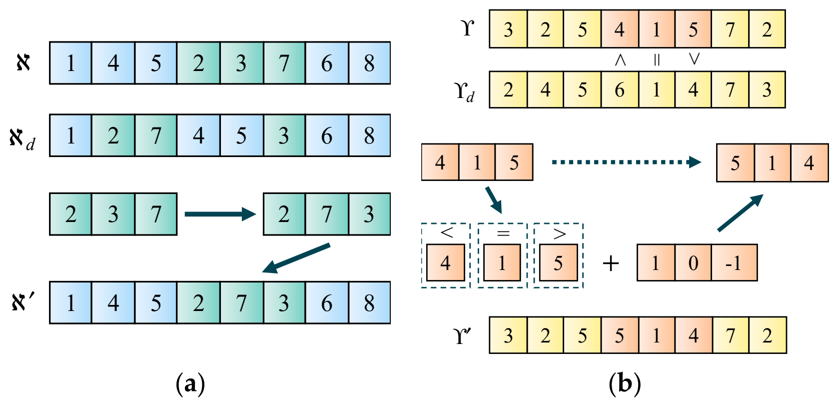


Figure 9. Multiway attacking process. (**a**) Processing of $\aleph$. (**b**) Processing of Υ.

To overcome the local optima drawback of greedy selection, a biased selection method is employed after obtaining the offspring individuals, denoted as Δ_s. Then, we have

$$\Delta_s = \begin{cases} \Delta', & \text{if } \vartheta(\boldsymbol{x}') > \vartheta(\boldsymbol{x}) \vee rand < P_{bias}(1 + \vartheta(\boldsymbol{x}') - \vartheta(\boldsymbol{x})) \\ \Delta, & otherwise \end{cases} \tag{28}$$

where P_{bias} is the biased selection probability; $\boldsymbol{x}$ and $\boldsymbol{x}'$ are solution vectors corresponding to individuals Δ and Δ', respectively, which gives the opportunity to retain some slightly inferior individuals.

3.5. Avoiding Local Convergence and Population Replacement

We propose a Lamarckian-learning-based multi-neighborhood search (LMS) to prevent MOWPS from becoming stuck in local optima and introduce an adaptive restart strategy to enhance population diversity.

The LMS acts on each elite individual $\Delta_e = \{\aleph_e; \Upsilon_e\} \in \Psi_e$ and employs four neighborhood structures as follows:

- Ne_1: *Swap*. Randomly select two elements in $\aleph_e$ and swap them;
- Ne_2: *Insert*. Randomly remove an element in $\aleph_e$ and reinsert it into a different position;
- Ne_3: *Inverse*. Randomly select two positions in $\aleph_e$ and invert the elements between them;
- Ne_4: *compound*. Randomly choose two neighborhoods from Ne_1, Ne_2, and Ne_3 and execute them in turn.

The Lamarckian learning method [56] is improved to select the neighborhood for LMS. Initially, a utility value φ_i of 1/4 is assigned to each neighborhood Ne_i. Then, we use the roulette wheel method to choose neighborhood based on its utility value. After selected Ne_i is performed on Δ_e, φ_i is updated as follows.

$$\varphi_i = \varphi_i + \max\left\{\frac{\vartheta(\boldsymbol{x}_a) - \vartheta(\boldsymbol{x}_b)}{\vartheta(\boldsymbol{x}_b)}, 0\right\} \tag{29}$$

where the solution vectors before and after a neighborhood operation are $\boldsymbol{x}_b$ and $\boldsymbol{x}_a$, respectively. If a neighborhood produces a better result, it is more likely to be chosen. All individuals obtained through LMS are called Ψ_{lms}.

To address the issue of population homogeneity, an adaptive restart strategy is implemented. In each iteration, N_{re} individuals are restarted, and we have

$$N_{re} = floor(1 + |\Psi_e| \times K_{cur}/K_{max}) \tag{30}$$

Recall K_{cur} (current number of iterations) and K_{max} (maximum number of iterations). All restarted individuals are referred to as Ψ_{re}.

During each MOWPS iteration, the worst $|\Psi_{lms} \cup \Psi_{re}|$ individuals in population Ψ_{all} are replaced by generated $\Psi_{lms} \cup \Psi_{re}$.

3.6. Overall MOWPS Algorithm

By incorporating the aforementioned designs, the complete process of the proposed MOWPS can be summarized as follows:

Algorithm MOWPS

Input: Parameters N_p, ξ, ε, $P_{reserve}$, P_{bias};
Output: the nondominated solutions set Ψ_{nd};
Generate initial population Ψ_{all} contains N_p individuals, set K_{cur} = 0;
While $K_{cur} < K_{max}$
  Evaluate individuals in Ψ_{all} using SDRS;
  Obtain the elite set Ψ_e;
  For each individual $\Delta_e \in \Psi_e$
    Perform roaming process on Δ_e;
    Update Δ_e;
  end
  Perform multiway attacking on each individual in Ψ_{all};
  Perform LMS on each individual $\Delta_e \in \Psi_e$ and generate Ψ_{lms};
  Perform adaptive restart strategy and generate Ψ_{re};
  Replace the worst individuals in Ψ_{all} with $\Psi_{lms} \cup \Psi_{re}$;
  Obtain the nondominated solutions in Ψ_{all}, and subsequently update Ψ_{nd};
  $K_{cur} = K_{cur} + 1$;
end
Output the nondominated solutions set Ψ_{nd};

Complexity Analysis *:* Recall that N_p, ξ, and ε represent the number of individuals in Ψ_{all}, the elite proportion, and the selection proportion during the roaming process, respectively. The loop of MOWPS repeats K_{max} times. In each loop, the SDRS with complexity $O(|\Psi_{all}|^2)$ is first adopted to evaluate individuals in Ψ_{all}. Then the roaming process generates $2\varepsilon|\Psi_e|$ individuals for each elite individual in Ψ_e, and the total complexity is $O(2\varepsilon|\Psi_e|^2)$. After that, multiway attacking with complexity $O(|\Psi_{all}|)$ is performed. Finally, LMS is applied to individuals in Ψ_e, with a complexity of $O(|\Psi_e|)$. Thus, the complexity of MOWPS is $O(K_{max} \times (|\Psi_{all}|^2 + 2\varepsilon|\Psi_e|^2 + |\Psi_{all}| + |\Psi_e|)) = O(K_{max} \times (N_p{}^2 + 2\varepsilon\xi^2 N_p{}^2 + N_p + \xi N_p))$.

4. Results

A series of experiments were conducted in this section to evaluate the performance of the proposed MOWPS algorithm.

4.1. Experimental Setup

We created a SEC network simulation environment to carry out experiments. Six Walker Delta constellations were developed, each with different elements in Table 2, including Orbcomm [57], Globalstar [58], and Starlink [59]. The simulation started on [1 June 2022 00:00:00.000 UTCG], and the required orbit elements were obtained using the software STK [60]. We conducted extensive experiments using three groups of testing instances: *small*, *medium*, and *large*. Table 3 shows the constellations (κ) and the number of applications (v) for each instance type. In total, there are $3 \times 2 \times 3 = 18$ combinations and we generated 10 testing instances for each combination.

Table 2. Satellite constellations used in simulation.

Constellation (κ)	Altitude (km)	Inclination (deg)	Planes	Satellites (n)
A [57]	825	45	4	32
B [58]	1414	52	6	48
C	825	45	8	160
D	1110	53.8	12	300
E	1175	60	18	864
F [59]	1110	53.8	32	1600

Table 3. Parameter size for each instance type.

Instance Type	κ	v	$\kappa \times v$
Small	A, B	1, 3, 5	$2 \times 3 = 6$
Medium	C, D	2, 5, 8	$2 \times 3 = 6$
Large	E, F	5, 10, 15	$2 \times 3 = 6$

In each texting instance, the edge server's properties such as computing capacity f_k, processing energy consumption coefficient g_w, standby power g_s, and unit rental cost h_e^k for edge server $u_k \in \mathbf{U}$ are generated randomly from the ranges [5, 10], $[1 \times 10^{-28}, 2 \times 10^{-28}]$, [0.1, 0.2], and [1, 2], respectively. The assumed values for transmission power g_c, unit data transmission cost h_d, and unit network rental cost h_o are 30, 0.02, and 0.1, respectively. The evaluation of the algorithm was performed using five real-world applications [61], and their task structures are shown in Figure 10. A testing instance is composed of a random selection of these applications. For any task m_p, the amount of data $d_{i,p}$ and workload e_p are randomly generated from [5, 10] and [1, 2], respectively. Table 4 provides a summary of the main parameter settings used in the simulation.

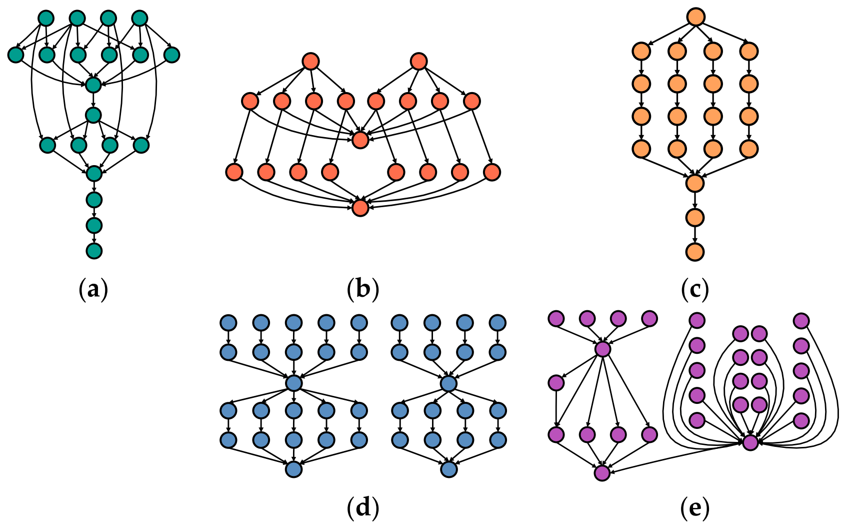


Figure 10. The structure of computation-intensive applications. (**a**) Montage; (**b**) CyberShake; (**c**) Epigenomics; (**d**) LIGO; (**e**) SIPHT.

Table 4. Simulation parameters.

Parameters	Value
Number of edge servers	n
The computing capacity f_k	[5, 10] Gcycles/s
The processing energy consumption coefficient g_w	$[1 \times 10^{-28}, 2 \times 10^{-28}]$
The standby power g_s	[0.1, 0.2] W/s
The unit rental cost h_e^k	[1, 2] \$/s
The transmission power g_c	30 W
The unit data transmission cost h_d	0.02 \$/MByte
The unit network rental cost h_o	0.1 \$/s
The communication capability of ISL r	1 Gbps
The amount of data volume $d_{i,p}$	[5, 10] MByte
The workload e_p of a task m_p	[1, 2] Kcycles/Byte

We conducted a comparative analysis between our proposed algorithm and three existing algorithms, namely IMOPSOQ [27], MCHO [15], and NSGA-II [41] (using Algorithm AM to meet the precedence constraints). The performance evaluation of these algorithms was based on three indicators:

1. *Number of non-dominated solutions* (*NS*) indicates the average quantity of nondominated solutions in each experiment.
2. *Hypervolume* (*HV*) [62] represents the hypercube's size enclosed by individuals and a reference point in the target space. The reference point $\boldsymbol{r}$ is normalized to a unit vector, and the enclosed volume of solution $\boldsymbol{x}$ is calculated as follows:

$$V(\boldsymbol{x}) = \{\boldsymbol{y} \in \boldsymbol{O} | \boldsymbol{x} \prec \boldsymbol{y} \wedge \boldsymbol{y} \prec \boldsymbol{r}\} \tag{31}$$

where $\boldsymbol{O}$ is the objective space; then, the hypervolume of Ψ_{nd} can be obtained by (32).

$$HV(\Psi_{nd}) = \bigcup_{\boldsymbol{x} \in \Psi_{nd}} \mathrm{V}(\boldsymbol{x}) \tag{32}$$

3. *Dominance rate* (*DR*) in this paper is defined as the ratio of non-dominated solutions obtained by an algorithm that dominate other algorithms' solutions. Therefore, we can express it as:

$$DR(\Psi_{nd}) = \frac{\sum\limits_{\boldsymbol{x} \in \Psi_{nd}} D(\boldsymbol{x})}{|\Psi_{nd}|} \tag{33}$$

where $D(\boldsymbol{x})$ is the ratio at which solution $\boldsymbol{x}$ dominates the solutions of other algorithms Ψ_{nd}^{other}, obtained by:

$$D(\boldsymbol{x}) = \frac{\sum \left\{ \boldsymbol{x} \prec \boldsymbol{y} \middle| \boldsymbol{y} \in \Psi_{nd}^{other} / \Psi_{nd} \right\}}{\left| \Psi_{nd}^{other} / \Psi_{nd} \right|} \tag{34}$$

To eliminate the impact of randomness, each algorithm was independently executed 10 times for each testing instance, and the average of three metrics (*aNS*, *aHV*, and *aDR*) was used to evaluate each algorithm. The maximum number of iterations was set to K_{max} = 100 × v. All simulations were conducted using MATLAB 2021a and run on a computer with an Intel Core i9-9900K CPU @3.60 GHz and 64 GB of RAM.

4.2. Constraint Amending Verification

To confirm the efficacy of Algorithm AM in addressing precedence constraints, we used eight combinations, A × 1, B × 2, C × 4, D × 4, D × 8, E × 15, E × 30, and F × 30. We randomly generated 2000 solutions for each combination and recorded the success rate and running time after applying Algorithm AM.

Table 5 summarizes the statistical results. In general, increasing the number of satellites for a certain number of applications results in fewer solutions violating the precedence constraint due to the availability of more edge servers to perform tasks. However, most of the randomly generated solutions in all combinations do not satisfy the task dependencies, highlighting the importance of constraint amending. Algorithm AM achieved a 100% success rate in obtaining feasible solutions, and the running time increased slightly with problem size. Even for the largest problem size of F × 30 (approximately 1600 satellites and 900 tasks), Algorithm AM was able to obtain a feasible solution within 0.12 s. These results demonstrate the effectiveness and polynomial complexity of the Petri-net-based amending method.

Table 5. Amending results of algorithm AM.

Scale $\kappa \times v$	Infeasible Solution Amount	Success Rate	Total Running Time	Single Solution Amending Time
A × 1	1315	100%	1.0049	7.64×10^{-4}
B × 2	1611	100%	2.2276	1.38×10^{-3}
C × 4	1723	100%	5.9061	3.42×10^{-3}
D × 4	1016	100%	6.6486	6.54×10^{-3}
D × 8	1826	100%	16.7135	9.15×10^{-3}
E × 15	1334	100%	51.6734	0.0387
E × 30	1899	100%	142.8397	0.0752
F × 30	1417	100%	169.0484	0.1193

4.3. Parameter Calibration

In this section, we calibrated all five parameters (N_p, ξ, ε, $P_{reserve}$, P_{bias}) of the MOWPS using the Design of the Experiment (DOE) [63]. We started by setting candidate factor levels for the parameters, as shown in Table 6.

Table 6. Candidate factor levels for each parameter.

Factor Level	N_p	ξ	ε	$P_{reserve}$	P_{bias}
1	20	0.1	0.1	0.2	0
2	30	0.2	0.2	0.3	0.1
3	40	0.3	0.3	0.4	0.2
4	50	0.4	0.4	0.5	0.3

We utilized the $L_{16}(4^5)$ orthogonal array in our DOE based on the number and factor levels of each parameter. We conducted the experiment thrice, as we had three instance types (small, medium, and large) with combinations A × 3, C × 5, and E × 10. Table 7 shows the $L_{16}(4^5)$ with 16 combinations of parameter factor levels. Each combination underwent ten independent runs of the MOWPS, and the *aHV* among these runs was considered as the response variable.

Table 7. Orthogonal array $L_{16}(4^5)$ and response variable.

Trial	Factor Level					Response Value (*aHV*)		
	N_p	ξ	E	$P_{reserve}$	P_{bias}	Small	Medium	Large
1	1	1	1	1	1	0.3572	0.1233	0.0227
2	1	2	2	2	2	0.5711	0.3283	0.1754
3	1	3	3	3	3	0.6881	0.4484	0.3008
4	1	4	4	4	4	0.6910	0.6475	0.4984
5	2	1	1	2	2	0.4918	0.0473	0.0492
6	2	2	2	1	1	0.5381	0.4592	0.3512
7	2	3	3	4	4	0.6290	0.5325	0.3719
8	2	4	4	3	3	0.7611	0.7062	0.6448
9	3	1	2	3	4	0.5324	0.3368	0.2184
10	3	2	1	4	3	0.6736	0.5558	0.4160
11	3	3	4	1	2	0.7681	0.5738	0.5567
12	3	4	3	2	1	0.7494	0.7155	0.7325
13	4	1	2	4	3	0.5183	0.3724	0.2932
14	4	2	1	3	4	0.7111	0.6951	0.5376
15	4	3	4	2	1	0.7191	0.5462	0.4146
16	4	4	3	1	2	0.7949	0.8282	0.7636

In Table 8, the statistical analysis for the response variables is presented. The optimal parameter values for each instance type are highlighted in bold black. For medium and large-type instances, the influence priority of parameters is the same. The most significant parameter is ξ, followed by ε, N_p, $P_{reserve}$, and P_{bias}. For small-type instances, ξ also has the highest rank, followed by ε, N_p, P_{bias}, and $P_{reserve}$. Based on these findings, we determined the parameter values of MOWPS as follows: $\{N_p, \xi, \varepsilon, P_{reserve}, P_{bias}\}$ = {50, 0.4, 0.4, 0.4, 0.2}, {50, 0.4, 0.3, 0.4, 0.3}, and {50, 0.4, 0.3, 0.4, 0.2} for small, medium, and large-type instances, respectively.

Table 8. Statistical analyses and suggested parameter values.

Factor Level		N_p	Ξ	ε	$P_{reserve}$	P_{bias}
Small	1	0.5769	0.4749	0.5584	0.6146	0.5910
	2	0.6050	0.6235	0.5400	0.6329	0.6565
	3	0.6809	0.7011	0.7154	**0.6732**	**0.6603**
	4	**0.6859**	**0.7491**	**0.7348**	0.6280	0.6409
	Delta	0.1090	0.2742	0.1949	0.0586	0.0693
	Rank	3	1	2	5	4
	SPV	50	0.4	0.4	0.4	0.2
Medium	1	0.3869	0.2199	0.3554	0.4961	0.4611
	2	0.4363	0.5096	0.3742	0.4093	0.4444
	3	0.5455	0.5252	**0.6312**	**0.5466**	0.5207
	4	**0.6105**	**0.7244**	0.6184	0.5270	**0.5530**
	Delta	0.2236	0.5044	0.2758	0.1373	0.1086
	Rank	3	1	2	4	5
	SPV	50	0.4	0.3	0.4	0.3
Large	1	0.2493	0.1459	0.2564	0.4236	0.3803
	2	0.3543	0.3701	0.2596	0.3429	0.3862
	3	0.4809	0.4110	**0.5422**	**0.4254**	**0.4137**
	4	**0.5023**	**0.6598**	0.5286	0.3949	0.4066
	Delta	0.2529	0.5140	0.2858	0.0825	0.0334
	Rank	3	1	2	4	5
	SPV	50	0.4	0.3	0.4	0.2

4.4. Comparison with Existing Algorithms

After conducting the calibration, we compared MOWPS with three existing algorithms, using the instances from Section 4.1 for the comparison. In Figure 11, the trade-offs produced by various algorithms for different instance sizes are depicted in 2D and 3D plots, with each point on the plot representing a potential task assignment. The results indicate that MOWPS generates superior trade-off fronts compared to other algorithms, with a more evenly distributed set of solutions. This is attributed to the proposed Petri-net-based constraint amending method, which does not constrain the solution space, and the SDRS, which effectively evaluates solutions. Therefore, the task assignments produced by MOWPS offer a better trade-off between energy, cost, and makespan.

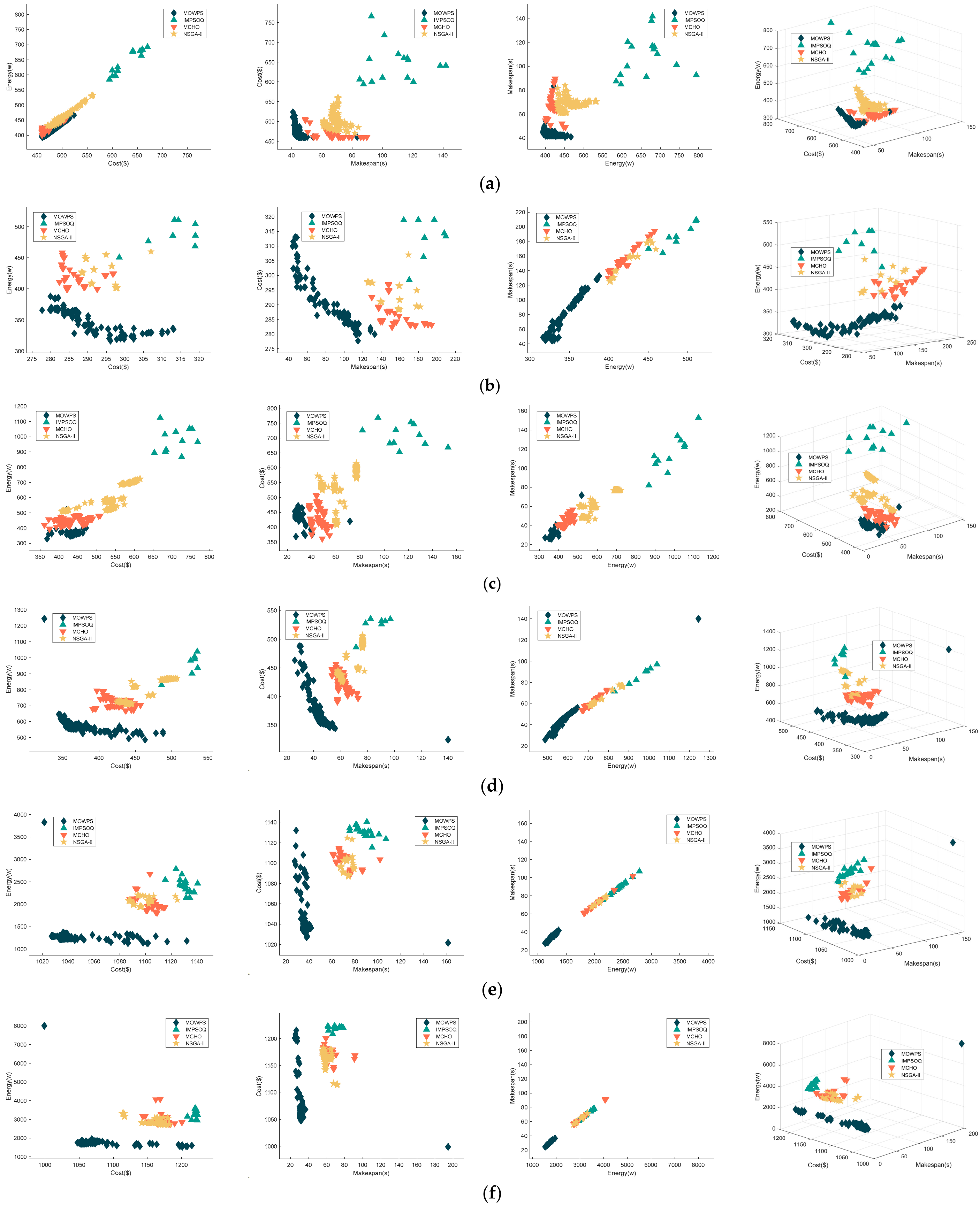


Figure 11. Trade-offs for all four algorithms on different instances. (**a**) A × 1 (**b**) B × 1 (**c**) C × 2 (**d**) D × 2 (**e**) E × 5 (**f**) F × 5.

Table 9 displays the statistics of the comparison results for all algorithms. The performance of each algorithm varies significantly, with the optimal value for each instance highlighted in bold black. MOWPS outperforms all other algorithms in terms of *aNS*, *aHV*, and *aDR* in almost all cases, except for a few instances where NSGA-II has the highest *aNS*.

Table 9. Comparison results of different algorithms.

Instance Type	Scale $\kappa \times v$	IMOPSOQ			MCHO			NSGA-II (Using AM)			MOWPS		
		aNS	*aHV*	*aDR*	*aNS*	*aHV*	*aDR*	*aNS*	*aHV*	*aDR*	*aNS*	*aHV*	*aDR*
small	A × 1	3	0.0074	0	7	0.7080	0.0534	46.2	0.4538	0.0406	**584.6**	**0.9895**	**0.9105**
	B × 1	1.6	2.82×10^{-4}	0	19.4	0.1612	0.2651	26	0.0609	0.0270	**49.4**	**0.7865**	**0.6964**
	A × 3	3.2	0.0046	0	6.6	0.5695	0.2249	36.6	0.3531	0.0445	**170**	**0.9986**	**0.9908**
	B × 3	2.2	0	0	17.2	0.3215	0.2216	54.4	0.1430	0.0097	**266.6**	**0.9155**	**0.9640**
	A × 5	4.2	0.0193	0	13.2	0.4061	0.3558	49.6	0.2489	0.0852	**91.2**	**0.9999**	**1**
	B × 5	2.4	1.80×10^{-4}	0	11.6	0.1930	0.3005	138.2	0.0860	0.0556	**194.4**	**0.9891**	**0.9994**
medium	C × 2	2.2	0.0012	0	16.8	0.7287	0.5179	**103**	0.2823	0.0427	77.4	**0.9519**	**0.7718**
	D × 2	1.4	5.66×10^{-4}	0	18.4	0.3424	0.1039	64.8	0.1584	0.0064	**774**	**0.8065**	**0.9943**
	C × 5	1.8	0	0	13	0.3037	0.2933	208	0.1829	0.0116	**242**	**0.9357**	**0.9953**
	D × 5	1.8	2.40×10^{-5}	0	7.8	0.3318	0.1662	111.60	0.2633	0.0109	**434.8**	**0.9193**	**0.9950**
	C × 8	2.4	0.0012	0	6	0.4119	0.4786	**168.2**	0.2175	0.0466	83	**0.9496**	**0.9751**
	D × 8	1.4	3.28×10^{-4}	0	8.8	0.2710	0.2494	161.4	0.1674	0.0116	**241.8**	**0.8994**	**0.9878**
large	E × 5	4.4	0.0291	0	9	0.1826	0.0117	84.6	0.1696	0.0247	**481.4**	**0.8657**	**0.9929**
	F × 5	2.8	0.0117	0	8.2	0.1828	0.0585	215.2	0.1977	0.0426	**412.2**	**0.7256**	**0.9783**
	E × 10	3.4	0.0494	0	6.8	0.1756	0.0803	99.4	0.1849	0.0370	**281**	**0.7306**	**0.9919**
	F × 10	1.2	0.0032	0	9	0.1961	0.1275	158.6	0.1684	0.0297	**234.6**	**0.5744**	**0.8337**
	E × 15	2.4	0.0148	0	4.8	0.1374	0.1972	**271.6**	0.1384	0.1774	93	**0.4188**	**0.5080**
	F × 15	2.2	0.0198	0	9.6	0.4417	0.3541	**120.6**	0.3103	0.0937	11.2	**0.4675**	**0.6564**

Figure 12 illustrates the performance of each algorithm on different instances. In Figure 12a, MOWPS and NSGA-II produce a significantly higher number of non-dominated solutions than MCHO and IMPOSQ for all instances, as MCHO and IMPOSQ's constraint processing method restricts the solution space to produce a specific task sequence. In contrast, our Petri-net-based method efficiently amends any randomly generated task sequence to satisfy the constraints, as demonstrated in Section 4.2. Figure 12b indicates that MOWPS outperforms other algorithms in terms of *aHV*, considering the balance between convergence and diversity of the non-dominated solutions, while others only consider diversity as a submetric (e.g., IMPOSQ and NSGA-II), which enables MOWPS to obtain more diverse and high-quality solutions. Figure 12c demonstrates that MOWPS consistently generates solutions with high *aDR* values, validating the effectiveness of the algorithmic search mechanism. These numerical experiments confirm the exceptional performance of MOWPS for offloading tasks with dependencies in SEC.

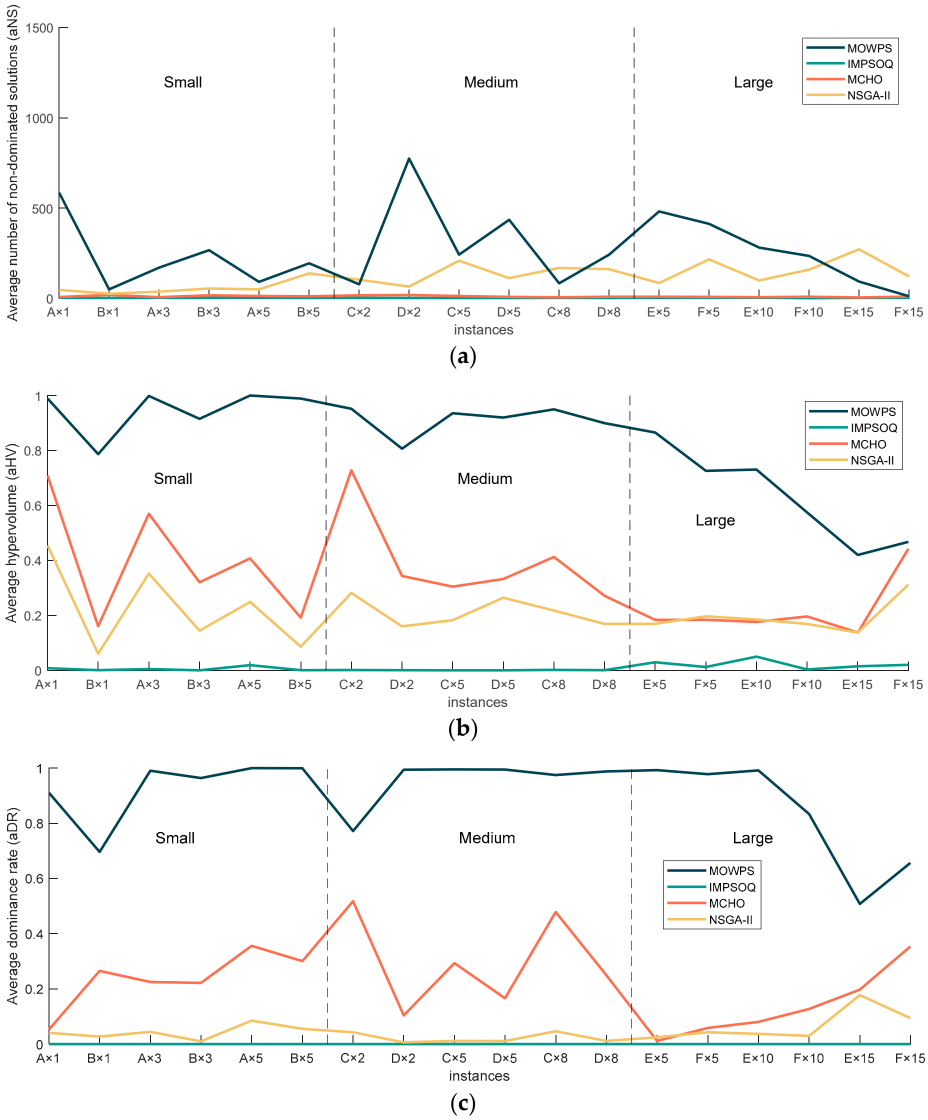


Figure 12. Variation trend of *aNS*, *aHV*, and *aDR* for each algorithm in different instances. (**a**) Average number of nondominated solutions (*aNS*); (**b**) average hypervolume (*aHV*); (**c**) average dominance rate (*aDR*).

5. Discussion

Based on the simulation and analysis results, we can provide a concise analysis and discussion, as follows:

(1) As the number of tasks assigned to edge servers increases, the likelihood of violating precedence constraints also increases, resulting in unpredictable wait times due to the existence of tasks with dependencies. However, our proposed Petri-net-based constraint amending method can efficiently obtain feasible solutions even for large-scale scenarios with 1600 satellites and approximately 900 tasks within a short time frame of 0.12 s. This highlights the effectiveness and efficiency of our proposed method.

(2) Compared to IMPSOQ and MCHO, our algorithm is more effective in generating solutions closer to the Pareto front for all 18 instances, as indicated by the dominance rate indicator. This is because our proposed constraint amending method does not restrict the solution space and can efficiently repair precedence constraints in any randomly generated solution. In contrast, IMPSOQ and MCHO use a queue-based method and the UpwardRank method, respectively, to calculate feasible task sequences under the

current task parameters, which not only increases computational cost but also affects the quality of the assignment solutions.

(3) Our proposed algorithm is superior to others in achieving a balance between the convergence and diversity of non-dominated solutions. This is demonstrated by the hypervolume indicator, which evaluates the strength of non-dominated solutions. We achieved this balance by implementing our SDRS method, which evaluates both the convergence and diversity of non-dominated solutions, resulting in better solutions overall. In contrast, other algorithms such as NSGA-II and IMPSOQ rely on crowding distance measures and grid-based methods, respectively, to filter solutions after obtaining a non-dominated solution set. This approach can eliminate some dominated solutions that may have good diversity. Additionally, MCHO optimizes multiple populations for different objectives, but the population used for diversity evaluation only employs normalized linear aggregation, which can compromise the effectiveness of balancing diversity.

6. Conclusions

This paper models the task offloading with data-dependent constraints in an SEC networks as a multi-objective optimization problem. We address the challenges of dependency constraints by proposing a Petri-net-based constraint amending method. Our theoretical and experimental analyses illustrate its effectiveness and polynomial complexity. For the multiple optimization objectives, a strengthened dominance relation sort is established to balances the convergence and diversity of nondominated solutions. Based on these, we propose the MOWPS algorithm. MOWPS incorporates adaptive mechanisms to reduce computational overhead and uses Lamarckian-learning-based multi-neighborhood search to avoid local optima. Extensive experiments demonstrate that MOWPS outperforms existing algorithms in terms of energy, cost, and makespan tradeoffs when solving task offloading with data-dependent constraints in an SEC networks. In the future, we plan to expand our algorithms to address problems such as satellite failures and uncertain computation times.

Author Contributions: R.Z. and Y.F. conceived the conceptualization and algorithm. R.Z. completed the implementation of the algorithm and the writing of the paper and supported the writing review and editing. Y.F. completed some preliminary simulations and performed the preliminary research and summary. Y.Y. and X.L. provided theoretical guidance and suggestions for revision of the paper. Y.Y. provided funding support and necessary assistance for thesis writing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Science and Technology Innovation 2030-Key Project of "New Generation Artificial Intelligence" under Grant 2020AAA0108203 and the National Natural Science Foundation of P.R. China under Grants 62003258 and 62103062.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Boero, L.; Bruschi, R.; Davoli, F.; Marchese, M.; Patrone, F. Satellite networking integration in the 5G ecosystem: Research trends and open challenges. *IEEE Netw.* **2018**, *32*, 9–15. [CrossRef]
2. Zhang, H.; Liu, R.; Kaushik, A.; Gao, X. Satellite Edge Computing with Collaborative Computation Offloading: An Intelligent Deep Deterministic Policy Gradient Approach. *IEEE Internet Things J.* **2023**, *10*, 9092–9107. [CrossRef]
3. Chai, F.; Zhang, Q.; Yao, H.; Xin, X.; Gao, R.; Guizani, M. Joint Multi-task Offloading and Resource Allocation for Mobile Edge Computing Systems in Satellite IoT. *IEEE Trans. Veh. Technol.* **2023**, *72*, 7783–7795. [CrossRef]
4. Xie, R.; Tang, Q.; Wang, Q.; Liu, X.; Yu, F.R.; Huang, T. Satellite-terrestrial integrated edge computing networks: Architecture, challenges, and open issues. *IEEE Netw.* **2020**, *34*, 224–231. [CrossRef]
5. Liu, J.; Du, X.; Cui, J.; Pan, M.; Wei, D. Task-oriented intelligent networking architecture for the space–air–ground–aqua integrated network. *IEEE Internet Things J.* **2020**, *7*, 5345–5358. [CrossRef]
6. Qiu, C.; Yao, H.; Jiang, C.; Guo, S.; Xu, F. Cloud computing assisted blockchain-enabled Internet of Things. *IEEE Trans. Cloud Comput.* **2019**, *10*, 247–257. [CrossRef]
7. Tang, Q.; Fei, Z.; Li, B.; Han, Z. Computation offloading in LEO satellite networks with hybrid cloud and edge computing. *IEEE Internet Things J.* **2021**, *8*, 9164–9176. [CrossRef]

8. Islam, A.; Debnath, A.; Ghose, M.; Chakraborty, S. A survey on task offloading in multi-access edge computing. *J. Syst. Archit.* **2021**, *118*, 102225. [CrossRef]
9. Mao, B.; Tang, F.; Kawamoto, Y.; Kato, N. Optimizing computation offloading in satellite-UAV-served 6G IoT: A deep learning approach. *IEEE Netw.* **2021**, *35*, 102–108. [CrossRef]
10. Zhang, Z.; Zhang, W.; Tseng, F.H. Satellite mobile edge computing: Improving QoS of high-speed satellite-terrestrial networks using edge computing techniques. *IEEE Netw.* **2019**, *33*, 70–76. [CrossRef]
11. Hu, Y.; Gong, W.; Zhou, F. A Lyapunov-Optimized Dynamic Task Offloading Strategy for Satellite Edge Computing. *Appl. Sci.* **2023**, *13*, 4281. [CrossRef]
12. Qin, J.; Guo, X.; Ma, X.; Li, X.; Yang, J. Application and Performance Evaluation of Resource Pool Architecture in Satellite Edge Computing. *Aerospace* **2022**, *9*, 451. [CrossRef]
13. Yu, S.; Gong, X.; Shi, Q.; Wang, X.; Chen, X. EC-SAGINs: Edge-computing-enhanced space–air–ground-integrated networks for internet of vehicles. *IEEE Internet Things J.* **2021**, *9*, 5742–5754. [CrossRef]
14. Zhang, Y.; Chen, C.; Liu, L.; Lan, D.; Jiang, H.; Wan, S. Aerial edge computing on orbit: A task offloading and allocation scheme. *IEEE Trans. Netw. Sci. Eng.* **2022**, *10*, 275–285. [CrossRef]
15. Li, H.; Wang, D.; Zhou, M.; Fan, Y.; Xia, Y. Multi-Swarm Co-Evolution Based Hybrid Intelligent Optimization for Bi-Objective Multi-Workflow Scheduling in the Cloud. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 2183–2197. [CrossRef]
16. Ma, Y.; Wang, L.; Zomaya, A.Y.; Chen, D.; Ranjan, R. Task-tree based large-scale mosaicking for massive remote sensed imageries with dynamic dag scheduling. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *25*, 2126–2137. [CrossRef]
17. Zhou, H.; Ma, A.; Niu, Y.; Ma, Z. Small-Object Detection for UAV-Based Images Using a Distance Metric Method. *Drones* **2022**, *6*, 308. [CrossRef]
18. Li, J.; Lu, H.; Xue, K.; Zhang, Y. Temporal netgrid model-based dynamic routing in large-scale small satellite networks. *IEEE Trans. Veh. Technol.* **2019**, *68*, 6009–6021. [CrossRef]
19. Verma, A.; Kaushal, S. A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. *Parallel Comput.* **2017**, *62*, 1–19. [CrossRef]
20. Song, Z.; Hao, Y.; Liu, Y.; Sun, X. Energy-efficient multiaccess edge computing for terrestrial-satellite Internet of Things. *IEEE Internet Things J.* **2021**, *8*, 14202–14218. [CrossRef]
21. Kaur, K.; Mangat, V.; Kumar, K. A review on Virtualized Infrastructure Managers with management and orchestration features in NFV architecture. *Comput. Netw.* **2022**, *217*, 109281. [CrossRef]
22. Tirmizi, S.B.R.; Chen, Y.; Lakshminarayana, S.; Feng, W.; Khuwaja, A.A. Hybrid satellite–terrestrial networks toward 6G: Key technologies and open issues. *Sensors* **2022**, *22*, 8544. [CrossRef] [PubMed]
23. Jiang, W. Software defined satellite networks: A survey. *Digit. Commun. Netw.* **2023**, in press. [CrossRef]
24. Wang, C.; Ren, Z.; Cheng, W.; Zheng, S.; Zhang, H. Time-expanded graph-based dispersed computing policy for LEO space satellite computing. In Proceedings of the 2021 IEEE Wireless Communications and Networking Conference (WCNC), Nanjing, China, 29 March–1 April 2021; pp. 1–6.
25. Kim, T.; Kwak, J.; Choi, J.P. Satellite edge computing architecture and network slice scheduling for IoT support. *IEEE Internet Things J.* **2021**, *9*, 14938–14951. [CrossRef]
26. Liu, B.; Xu, X.; Qi, L.; Ni, Q.; Dou, W. Task scheduling with precedence and placement constraints for resource utilization improvement in multi-user MEC environment. *J. Syst. Archit.* **2021**, *114*, 101970. [CrossRef]
27. Ma, S.; Song, S.; Yang, L.; Zhao, J.; Yang, F.; Zhai, L. Dependent tasks offloading based on particle swarm optimization algorithm in multi-access edge computing. *Appl. Soft Comput.* **2021**, *112*, 107790. [CrossRef]
28. Al-Habob, A.A.; Dobre, O.A.; Armada, A.G.; Muhaidat, S. Task scheduling for mobile edge computing using genetic algorithm and conflict graphs. *IEEE Trans. Veh. Technol.* **2020**, *69*, 8805–8819. [CrossRef]
29. Luan, Q.; Cui, H.; Zhang, L.; Lv, Z. A Hierarchical Hybrid Subtask Scheduling Algorithm in UAV-Assisted MEC Emergency Network. *IEEE Internet Things J.* **2021**, *9*, 12737–12753. [CrossRef]
30. An, X.; Fan, R.; Hu, H.; Zhang, N.; Atapattu, S.; Tsiftsis, T.A. Joint task offloading and resource allocation for IoT edge computing with sequential task dependency. *IEEE Internet Things J.* **2022**, *9*, 16546–16561. [CrossRef]
31. Song, F.; Xing, H.; Wang, X.; Luo, S.; Dai, P.; Li, K. Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach. *Future Gener. Comput. Syst.* **2022**, *128*, 333–348. [CrossRef]
32. Wang, Z.J.; Zhan, Z.H.; Yu, W.J.; Lin, Y.; Zhang, J.; Gu, T.L.; Zhang, J. Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling. *IEEE Trans. Cybern.* **2019**, *50*, 2715–2729. [CrossRef] [PubMed]
33. Qin, Z.; Yao, H.; Mai, T.; Wu, D.; Zhang, N.; Guo, S. Multi-Agent Reinforcement Learning Aided Computation Offloading in Aerial Computing for the Internet-of-Things. *IEEE Trans. Serv. Comput.* **2022**, *16*, 1976–1986. [CrossRef]
34. Abd Elaziz, M.; Xiong, S.; Jayasena, K.P.N.; Li, L. Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution. *Knowl.-Based Syst.* **2019**, *169*, 39–52. [CrossRef]
35. Hu, Y.; Gong, W. An On-Orbit Task-Offloading Strategy Based on Satellite Edge Computing. *Sensors* **2023**, *23*, 4271. [CrossRef]
36. Liu, J.; Ren, J.; Zhang, Y.; Peng, X.; Zhang, Y.; Yang, Y. Efficient dependent task offloading for multiple applications in MEC-cloud system. *IEEE Trans. Mob. Comput.* **2021**, *20*, 105–118. [CrossRef]

37. Zhang, R.; Feng, Y.; Yang, Y.; Li, X. A Deadlock-free Hybrid Estimation of Distribution Algorithm for Cooperative Multi-UAV Task Assignment with Temporally Coupled Constraints. *EEE Trans. Aerosp. Electron. Syst.* **2023**, *59*, 3329–3344. [CrossRef]
38. Feng, Y.; Xing, K.; Liu, H.; Wu, Y. Two-stage design method of robust deadlock control for automated manufacturing systems with a type of unreliable resources. *Inf. Sci.* **2019**, *484*, 286–301. [CrossRef]
39. Murata, T. Petri nets: Properties, analysis and applications. *Proc. IEEE* **1989**, *77*, 541–580. [CrossRef]
40. Ishibuchi, H.; Tsukamoto, N.; Nojima, Y. Evolutionary many-objective optimization: A short review. In Proceedings of the 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), Hong Kong, China, 1–6 June 2008; pp. 2419–2426.
41. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T.A.M.T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
42. Cheng, R.; Jin, Y.; Olhofer, M.; Sendhoff, B. A reference vector guided evolutionary algorithm for many-objective optimization. *IEEE Trans. Evol. Comput.* **2016**, *20*, 773–791. [CrossRef]
43. Aravanis, A.I.; MR, B.S.; Arapoglou, P.D.; Danoy, G.; Cottis, P.G.; Ottersten, B. Power allocation in multibeam satellite systems: A two-stage multi-objective optimization. *IEEE Trans. Wirel. Commun.* **2015**, *14*, 3171–3182. [CrossRef]
44. Dai, C.Q.; Xu, J.; Wu, J.; Chen, Q. Multi-objective intelligent handover in satellite-terrestrial integrated networks. In Proceedings of the 2022 IEEE International Conference on Communications Workshops (ICC Workshops), Seoul, Republic of Korea, 16–20 May 2022; pp. 367–372.
45. Gao, W.; Qu, L.; Wang, L. Multi-Objective Optimization of Joint Resource Allocation Problem in Multi-Beam Satellite. In Proceedings of the 2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 17–19 June 2022; Volume 10, pp. 2331–2338.
46. Qi, X.; Zhang, B.; Qiu, Z.; Zheng, L. Using Inter-Mesh Links to Reduce End-to-End Delay in Walker Delta Constellations. *IEEE Commun. Lett.* **2021**, *25*, 3070–3074. [CrossRef]
47. Available online: https://github.com/ZhangRuiPeng94/Aerospace-2023-Task_Offloading (accessed on 1 August 2023.).
48. Jia, M.; Zhu, S.; Wang, L.; Guo, Q.; Wang, H.; Liu, Z. Routing algorithm with virtual topology toward to huge numbers of LEO mobile satellite network based on SDN. *Mob. Netw. Appl.* **2018**, *23*, 285–300. [CrossRef]
49. Mathew, A.; Andrikopoulos, V.; Blaauw, F.J. Exploring the cost and performance benefits of AWS step functions using a data processing pipeline. In Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing, Leicester, UK, 6–9 December 2021; pp. 1–10.
50. Chen, S.; Deng, Y.; Attie, P.; Sun, W. Optimal deadlock detection in distributed systems based on locally constructed wait-for graphs. In Proceedings of the 16th International Conference on Distributed Computing Systems, Hong Kong, China, 27–30 May 1996; pp. 613–619.
51. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 2022.
52. Coffman, E.G.; Elphick, M.; Shoshani, A. System deadlocks. *ACM Comput. Surv.* **1971**, *3*, 67–78. [CrossRef]
53. Zhou, X.; Zhang, G.; Sun, J.; Zhou, J.; Wei, T.; Hu, S. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. *Future Gener. Comput. Syst.* **2019**, *93*, 278–289. [CrossRef]
54. Bao, C.; Xu, L.; Goodman, E.D. A new dominance-relation metric balancing convergence and diversity in multi-and many-objective optimization. *Expert Syst. Appl.* **2019**, *134*, 14–27. [CrossRef]
55. Shen, J.; Wang, P.; Wang, X. A controlled strengthened dominance relation for evolutionary many-objective optimization. *IEEE Trans. Cybern.* **2020**, *52*, 3645–3657. [CrossRef]
56. Ong, Y.S.; Keane, A.J. Meta-Lamarckian learning in memetic algorithms. *IEEE Trans. Evol. Comput.* **2004**, *8*, 99–110. [CrossRef]
57. Orabi, M.; Khalife, J.; Kassas, Z.M. Opportunistic navigation with Doppler measurements from Iridium Next and Orbcomm LEO satellites. In Proceedings of the 2021 IEEE Aerospace Conference, Big Sky, MT, USA, 6–13 March 2021; pp. 1–9.
58. Dietrich, F.J.; Metzen, P.; Monte, P. The Globalstar cellular satellite system. *IEEE Trans. Antennas Propag.* **1998**, *6*, 935–942. [CrossRef]
59. McDowell, J.C. The low earth orbit satellite population and impacts of the SpaceX Starlink constellation. *Astrophys. J. Lett.* **2020**, *892*, L36. [CrossRef]
60. McCamish, S.; Romano, M. Simulation of relative multiple spacecraft dynamics and control with MATLAB-SIMULINK and Satellite Tool Kit. In Proceedings of the AIAA Modeling and Simulation Technologies Conference and Exhibit, Hilton Head, SC, USA, 20–23 August 2007; p. 6805.
61. Masdari, M.; ValiKardan, S.; Shahi, Z.; Azar, S.I. Towards workflow scheduling in cloud computing: A comprehensive analysis. *J. Netw. Comput. Appl.* **2016**, *66*, 64–82. [CrossRef]
62. Zitzler, E.; Deb, K.; Thiele, L. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evol. Comput.* **2000**, *8*, 173–195. [CrossRef] [PubMed]
63. Antony, J. *Design of Experiments for Engineers and Scientists*; Elsevier: Amsterdam, The Netherlands, 2014.