Article

An Approach for Deployment of Service-Oriented Simulation Run-Time Resources

Zekun Zhang, Yong Peng *, Miao Zhang, Quanjun Yin and Qun Li

Abstract: The requirements for low latency and high stability in large-scale geo-distributed training simulations have made cloud-edge collaborative simulation an emerging trend. However, there is currently limited research on how to deploy simulation run-time resources (SRR), including edge servers, simulation services, and simulation members. On one hand, the deployment schemes of these resources are coupled and have mutual impacts. It is difficult to ensure overall optimum by deploying these resources separately. On the other hand, the pursuit of low latency and high system stability is often challenging to achieve simultaneously because high stability implies low server load, while a small number of simulation services implies high response latency. We formulate this problem as a multi-objective optimization problem for the joint deployment of SRR, considering the complex combinatorial relationship between simulation services. Our objective is to minimize the system time cost and resource usage rate of edge servers under constraints such as server resource capacity and the relationship between edge servers and base stations. To address this problem, we propose a learnable genetic algorithm for SRR deployment (LGASRD) where the population can learn from elites and adaptively select evolution operators performing well. Extensive experiments with different settings based on real-world data sets demonstrate that LGASRD outperforms the baseline policies in terms of optimality, feasibility, and convergence rate, verifying the effectiveness and excellence of LGASRD when deploying SRR.

Keywords: cloud-edge collaboration; joint resource deployment; composite service; learnable genetic algorithm

1. Introduction

Training simulation is a safe, controllable, and cost-effective training method that can improve the abilities of trainees, becoming a critical method of talent cultivation in various industries. In large-scale geo-distributed training simulations, simulation members, i.e., users, can be categorized into three types [1]: live (L), virtual (V), and constructive (C) members, such as actual aircraft with various sensors, aircraft simulators controlled by simulated cockpits and computer generated aircraft, respectively. These members dispersed across different locations are integrated into a virtual space, where they can see and interact with each other.

However, systems capable of supporting such large-scale simulation activities are often excessively large and require sufficient computing resources and network resources to support their operation. Otherwise, users will suffer from tremendous latency, which can significantly impair user experience quality.

To address this issue, researchers have proposed deploying simulation systems in the cloud-edge collaborative computing environment, utilizing both the computing resources of cloud computing and the networking resources of edge computing [2]. Under the cloud-edge collaborative simulation paradigm, developers usually adopt the micro-service (MS)
architecture to develop simulation systems. By encapsulating common functional modules into containers to form services, MS architecture can achieve higher development efficiency, more efficient resource reuse, and better elasticity [3,4].

During the operation of an MS-based training simulation system deployed in cloud-edge collaborative computing architecture, simulation members send service requests to the system to realize the functions they need. They also send messages to other members through the middleware service to achieve the interaction between him/her and other members, and also promote the progress of the current simulation activity.

As shown in Figure 1, each service request is composed of multiple micro-services and can be formulated as a directed acyclic graph (DAG) [5,6], in which a node represents a task, defined as the execution of its corresponding MS instance. Each task also has its predecessor and successor tasks. After a task is completed, its successor tasks will either be completed on the same server or on other servers. The latter method will incur service communication time, a part of data transmission time. Furthermore, MS instances provide computation in a queue-based manner, which may lead to long service waiting times due to huge user demands and a small number of instances.

![Figure 1. System Architecture.](image)

Furthermore, in large-scale geo-distributed training simulations, message sending may lead to long-distance data transmission, generating user interaction time. For example, in a training simulation activity, if user A needs to send a message to user B, the message needs to be sent to the middleware service first, and then the middleware will forward it to user B.

To implement the training simulation process mentioned above, the primary challenge is how to deploy edge servers, micro-services, and members in the cloud-edge collaborative computing environment before the start of the simulation. On the one hand, edge servers can provide both computing and communication capabilities, making them an important infrastructure for cloud-edge collaborative computing architecture. In reality, there often exist situations where there are \( n \) available edge servers but \( m (n < m) \) candidate server locations, typically base stations (BSs) [7–11]. Therefore, we need to determine where these \( n \) edge servers should be deployed. We assume that the cloud center has existed, so we do not focus on the deployment of it. On the other hand, simulation members are the main participants of training simulations, and micro-services are indispensable components for enabling members to perform their required functions, so they are also
extremely important for training simulations. As software resources, we need to determine their deployment schemes on servers. Note that the referred members here are constructive members, because in large-scale geo-distributed simulations, live members and virtual members are often difficult to move due to their large numbers, so we assume that their locations are known and fixed. For convenience in the description, we define servers, micro-services, and constructive members as simulation run-time resources (SRR), and refer to the latter two software resources as simulation run-time software resources (SRSR).

Different deployment schemes of SRR have a great impact on the time cost of simulation systems, including data transmission time and service queuing time, as well as the resource occupancy rate of edge servers. Data transmission is caused by users’ service requests and interactions between users. Without the loss of generality, it is usually assumed that a task is always scheduled to be processed by the server with the corresponding MS instance that is closest to its input data [11,12]. Obviously, the same server will face different user demands under different deployment schemes of edge servers and simulation members. Deployment decisions of micro-services have a significant influence on task scheduling and service queuing time. In addition, scattered users will lead to large data transmission time due to their messages. On the other hand, although more micro-service instances can reduce time costs, they can increase resource usage of edge servers. This increases the risk of crashes and should be avoided as much as possible. Therefore [13], deployment schemes of SRR are closely coupled and should be considered together.

However, there is currently little research on this problem. Existing works mostly study one problem in service deployment or server deployment. As mentioned earlier, they are coupled and it is difficult to achieve the overall optimum by researching them separately. In addition, most works only consider the deployment of monolithic services, instead of micro-services, thus ignoring the data dependencies between micro-services [11,14]. However, this deployment scheme cannot fully capture the characteristics of systems based on MS architecture. Finally, the current related research focuses mainly on scenarios such as autonomous driving and the Internet of Things (IoT), rather than training simulation. However, there are some elements in training simulation that these fields do not possess, such as L, V, and C members and their interactions.

This work studies the joint deployment of SRR in cloud-edge collaborative computing environments, aiming to minimize system time cost, while minimizing the resource consumption rate of edge servers. We formulate the joint deployment problem and design a learnable multiple-objective evolutionary algorithm to solve it. Furthermore, we target the service-oriented training simulation field, a more specialized scenario. In this paper, the main contributions are as follows:

1. We propose a complete process for deploying edge servers and software resources together, taking a more complex service modeling approach into account. This is an improvement over existing work.
2. We formulate the joint deployment of edge servers and SRSR as a multi-objective optimization problem with two objectives: minimizing both the system’s time cost and the resource usage rate of edge servers. To solve this problem, we proposed a learnable genetic algorithm for simulation run-time resource deployment (LGASRD), which is capable of adaptively selecting genetic operators and evolving under the guidance of an elite population.
3. Extensive experiments are implemented in Python to evaluate the performance of the LGASRD based on the real-world EUA data set [15] and scientific workflows [16]. The experimental results show that LGASRD outperforms the other typical benchmark policies.

The remainder of this article is organized as follows. Section 2 provides an overview of related works. Section 3 elaborates on the motivations behind our study on the joint deployment of SRR. Section 4 introduces the system model and formulates a multi-objective optimization problem. LGASRD is proposed to solve the problem in Section 5 and its
performance analysis is conducted based on real-world data sets in Section 6. Finally, Section 7 concludes this article.

2. Related Works

Training simulation is one of the most effective and economical ways to enhance personnel’s professional ability, which has become a consensus. Faced with the service-oriented simulation and the use of cloud computing and edge computing for simulation, it is an inevitable trend in the development of computer simulation technology [17–20]. Therefore, the cloud-edge collaborative training simulation not only faces some challenges in the field of cloud computing and edge computing, such as server deployment [21–24] and service deployment [25–28], but also has some unique problems. Training simulation is different from other types of simulation such as supply chain, production, and manufacturing simulation. In the training simulation process, there will be a large amount of interaction between virtual and real entities. Therefore, unlike conventional research on edge computing technology, we need to consider the additional time cost, caused by interaction, of the edge computing systems used for training simulation. In addition, the deployment issue of computer-generated constructive simulation members, which can be regarded as a type of special simulation service, also needs to be resolved urgently. However, it is regrettable that we have not seen any relevant research yet.

Currently, research on how to deploy services to distributed edge computing systems is a hot topic. Chen et al. [29] defined the problem as a combinatorial contextual bandit learning problem. They utilized the Multi-armed Bandit theory to learn the optimal deployment scheme of services. Ouyang et al. [30] studied the problem of how to deploy services in the distributed edge computing environment in a mobility-aware and dynamic way, aiming to optimize the long-term averaged migration cost due to user mobility. To achieve this, they developed two efficient heuristic schemes based on the Markov approximation and best response update techniques. Gao et al. [28] designed an iterative algorithm to study how to achieve optimal service deployment and access network selection. Hu et al. [31] proposed an algorithm to adjust the task scheduling decisions and resource allocation by achieving a trade-off between the execution time of tasks and energy consumption.

However, those researchers only studied the services in a monolithic way, ignoring the composite property of services. Wang et al. [12,32] considered this factor, but only considered the linear combination relationship between services. However, in reality, the business logic relationships between services should be represented by DAG. In addition, though these works took different aspects into account, they assumed that all base stations could be deployed with edge servers. In fact, the number of available edge servers is limited and thus it is not reasonable that all base stations are enabled to execute computing tasks. Furthermore, different server deployment schemes will generate different service requests from users, resulting in different service deployment schemes. Therefore, the scheme of service deployment can be obtained unless the deployment of usable edge servers has finished. Considering this, these two elements should be deployed jointly.

In edge computing, the most common deployment method is to place edge servers at base stations [21,23]. Li et al. [21] identified a gap in the related works regarding energy efficiency in edge server deployment. To address this, they proposed a multi-objective optimization approach, incorporating an energy-aware edge server deployment algorithm. Their objective was to obtain a deployment solution that minimized energy consumption while maintaining access delay at an acceptable level. Mohan et al. [22] allowed edge servers to request processing of the extra workload from the cloud based on the assumption that the capacity constraints on each edge server were relaxed. Wang et al. [23] focused on the deployment of edge servers in a metropolitan area, assuming that all edge servers were homogeneous. They proposed a mixed-integer programming approach that aimed to minimize the edge server access latency as well as balance the workload among edge servers to achieve this goal. Lähderanta et al. [24] investigated the deployment of edge servers as a constrained location-allocation problem. The study incorporated the constraints of
both upper and lower server capacities, aiming to minimize distance and achieve workload balance among the edge servers. However, the mentioned works just considered where to place the edge servers with different optimization goals and constraints, without taking the service placement under the current structure of the edge server placement scheme into account. They assume that each edge server hosts only one type of service, ignoring the demand of users for different services. In addition, they overlooked the diverse service requests of users with the assumption that each edge server was dedicated to hosting a single type of service.

3. Motivation

Figure 2 provides a more vivid illustration of the importance of considering the joint deployment of SRR, although there is currently little research on this topic. The figure includes training bases (TBs), edge servers (ESs), BSs, and a scheduling center located in the cloud. Software resources on ESs include MS instances and constructive members, depicted by colored circles and aircraft, respectively. To facilitate distinction, a jagged symbol is used to represent the middleware service instead of using a circle.

![Figure 2](image.png)

**Figure 2.** Deployment schemes for different objects: (a) Edge server deployment; (b) SRSR deployment; (c) Joint SRR deployment.

Taking Figure 2a as an example, only the deployment of servers is considered without considering data dependencies between services. In this case, three ESs are deployed on BS 1, 2, and 4, with purple, green, and middleware services deployed on them, respectively. A live user submits a service request for path planning. Since all ESs do not have the corresponding instance, the task can only be completed in the cloud, where all types of services are deployed. This will remarkably increase network latency. Additionally, a live user sends a message “Hello” to a constructive white aircraft, resulting in two data transmissions. However, if the white aircraft and middleware are deployed together, the second data transmission can be avoided.

The situation shown in Figure 2b optimizes the deployment of services and simulation members. The yellow service request submitted by a live user is dispatched to ES 1, avoiding huge communication costs generated by execution in the cloud. Additionally, some constructive members are deployed together with middleware, reducing user interaction costs to a certain extent. However, BS 3 and BS 5 are far away from TB A and B. Intuitively, it would be a better choice to deploy the two servers at BS 1 and 2.

Figure 2c illustrates the joint deployment of SRR, including ESs, micro-services, and constructive members, considering data dependencies between micro-services. A live user submits a request for path planning, including two tasks, $t_1$ and $t_2$. Task $t_1$ is scheduled to server 1, and then the output is sent to server 2 for processing the second task $t_2$. It should be noted that the yellow MS instances are deployed on both server 1 and server 2. The reason $t_1$ is scheduled to server 1 rather than server 2 is that the instance on server 2 has other two tasks to process and therefore has a long waiting time.
Another significant difference between Figure 2a–c is that Figure 2c considers micro-services, using a more complex service representation approach, DAG, whereas Figure 2a,b consider monolithic services. This is mainly caused by the granularity difference between the services that the simulation system can support and the services required by users. The performance of the micro-services architecture depicted in Figure 2c is more flexible, so it is currently the mainstream architecture for developing large systems, such as simulation systems, in both the present and future [33]. Therefore, it is important, reasonable, and practical to study simulation resource deployment based on MS architecture.

4. System Model and Problem Formulation

In this section, we first give mathematical definitions of services, tasks, servers, users, and the representation method for joint deployment decision-making. Then, we calculate two objective functions: time cost and resource occupancy rate. Finally, we formulate the joint deployment optimization model. The main notations used in this article are listed in Table 1.

Table 1. Summary of key notations.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS_i</td>
<td>the i-th type of MS</td>
</tr>
<tr>
<td>r_{MS_i}</td>
<td>the resource requirement of MS_i</td>
</tr>
<tr>
<td>S_i</td>
<td>the i-th type of CS combined by micro-services</td>
</tr>
<tr>
<td>S_{ij}</td>
<td>the j-th type of service request from the i-th user</td>
</tr>
<tr>
<td>t_{ijl}</td>
<td>the l-th task in S_{ij}</td>
</tr>
<tr>
<td>R_i</td>
<td>the i-th edge server</td>
</tr>
<tr>
<td>R_{Ci}</td>
<td>the resource capacity of R_i</td>
</tr>
<tr>
<td>L_{R_i}</td>
<td>the location coordinate of R_i</td>
</tr>
<tr>
<td>U_i</td>
<td>the i-th user</td>
</tr>
<tr>
<td>r_{Ci}</td>
<td>the resource requirement of the i-th constructive member</td>
</tr>
<tr>
<td>DS</td>
<td>the deployment scheme, i.e., the solution of the problem to be solved</td>
</tr>
<tr>
<td>R(M)</td>
<td>the edge server deploying the middleware</td>
</tr>
<tr>
<td>R(U_i(t_j))</td>
<td>the closest edge server to user U_i that deploying instances can execute t_j</td>
</tr>
<tr>
<td>R(t_j)</td>
<td>the edge server can process t_j with the smallest average data transmission time from the server executing the predecessor tasks of t_j</td>
</tr>
<tr>
<td>R(C_i)</td>
<td>the edge server deploying constructive member C_i</td>
</tr>
<tr>
<td>R(U_i)</td>
<td>the edge server closest to user U_i</td>
</tr>
<tr>
<td>t^C</td>
<td>the communication time caused by the service request S_i</td>
</tr>
<tr>
<td>t^M</td>
<td>the total queuing time of all tasks in the service request S_i</td>
</tr>
<tr>
<td>t^S</td>
<td>the data transmission time caused by users’ service requests</td>
</tr>
<tr>
<td>t^M</td>
<td>the interaction time cost among users</td>
</tr>
<tr>
<td>B_i</td>
<td>the resource usage rate of R_i</td>
</tr>
</tbody>
</table>

4.1. Definitions

- **Service**: In a system implemented with MS architecture, the granularity of service requests submitted by users is usually greater than the granularity of individual micro-services (also called simulation services) that the system can provide. In such cases, it is necessary to combine simulation services into a composite service (CS) to meet user needs. Typically, users’ service requests are modeled as DAG. Assuming that the MS library consists of n classes of micro-services, expressed by set MS = [MS_1, ..., MS_n]. To simplify the problem, assume that the resource requirement of MS_i is expressed as r_{MS_i} = [cpu_{MS_i}, gpu_{MS_i}, ram_{MS_i}]^T and the input size and output size of MS_i are m_{in}^i and m_{out}^i, respectively. All micro-services can be combined to generate m composite services, represented by S = [S_1, ..., S_m], to meet users’ demands.
- **Task**: The execution of a fine-grained simulation service is referred to as a task. In the $j$-th type of service request $S_j$, submitted by a user $U_i$, the $l$-th task $t_{ijl}$ has its predecessor and successor tasks, represented by sets $\text{pred}(t_{ijl})$ and $\text{succ}(t_{ijl})$, respectively. The initial task and final task in the request are $t_{ij1}$ and $t_{ijf}$, whose $\text{pred}(t_{ij1}) = \text{succ}(t_{ijf}) = U_i$. To facilitate the description of tasks and their corresponding micro-services, we define $\text{tms}_i$ to represent the MS whose instances can process task $t_{ij}$.

- **Server**: Assuming there are $k$ available edge servers $R = [R_1, \ldots, R_k]$ for training simulation. The resource capacity of edge server $R_i$ is $R_i = [\text{cpu}_{R_i}, \text{gpu}_{R_i}, \text{ram}_{R_i}]^T$. Generally, each edge server is deployed on a BS. Assuming $q (q \geq 1)$ BSs are available, each BS can deploy one edge server at most. Once the mapping relationship between edge servers and BSs is determined, each server’s location can be described by a two-dimensional coordinate as $L_{R_i} = (x_{R_i}, y_{R_i})$. Thus, the distance between $k$ edge servers can be calculated as a matrix $A = [a_{ij}]_{k \times k}$.

- **User**: Simulation users include live, virtual and $h$ constructive simulation members, with a total of $u (u > h)$, so the user set can be represented as $U = [U_1, \ldots, U_{u-h}, U_{u-h+1}, \ldots, U_u]$. Without the loss of generality, live and virtual users only make small movements around their original positions. We ignore their movements and treat their positions as fixed. For a constructive user, its location is the same as the location of the server deploying it. Thus, the location of user $U_i$ can be expressed as $L_{U_i} = (x_{U_i}, y_{U_i})$. As a kind of software resource, the computing resource demand of constructive member $C_l$ is defined as $t_{C_l} = [\text{cpu}_{C_l}, \text{gpu}_{C_l}, \text{ram}_{C_l}]^T$.

- **Deployment scheme**: Resources to be deployed include edge servers, simulation services, and constructive members. Servers need to be deployed on BSs and need to deploy software resources. Therefore, a deployment scheme $DS$ can be represented by a matrix, with the $i$-th column representing the deployment scheme for server $R_i$. The matrix is divided into three parts from top to bottom: $DS_R$, $DS_S$, and $DS_C$.

The first part, $DS_R = [DS_R^1, \ldots, DS_R^k]_{1 \times k}$, occupies one row of the matrix, indicating the mapping relationship between edge servers and BSs. The value of the $i$-th element in this row, $DS_R^i$, represents that the $i$-th edge server is determined to be deployed on BS $DS_R^i$. Note that a BS can deploy one edge server at most, so $1 \leq DS_R^i, DS_R^j \leq q$ and $DS_R^i \neq DS_R^j$, for $\forall i, j = 1, \ldots, k$ and $i \neq j$.

The second part, $DS_{MS} = [DS_{MS}^1, \ldots, DS_{MS}^N]_{1 \times k}$, is a $n$-row sub-matrix of matrix $DS$, where $DS_{MS}^i = [DS_{MS}^{i1}, \ldots, DS_{MS}^{in}]^T$. $DS_{MS}^i$ indicates the number of instances of the $j$-th type of micro-services deployed on server $R_i$. The number of instances is non-negative integer, so $DS_{MS}^i \in N$ for $\forall i = 1, \ldots, k$ and $\forall j = 1, \ldots, n$, where $N$ is the set of natural numbers. Note that as a special simulation service, one and only one middleware is necessary for training simulation activities. To reduce the interaction cost, we assume that it is deployed on edge servers rather than the cloud. To distinguish it from other simulation services, we use $DS_M = [DS_M^1, \ldots, DS_M^k]_{1 \times k}$, the first row of matrix $DS_{MS}$ as the deployment decision for the cloud. Due to limitations mentioned above, $DS_M^i$ are binary variables for $\forall i = 1, \ldots, k$, and $\sum_{i=1}^k DS_M^i = 1$.

The third part, $DS_C = [DS_C^1, \ldots, DS_C^h]_{h \times k}$, is also a sub-matrix, with $h$ rows, of $DS$, where $DS_C^i = [DS_C^{i1}, \ldots, DS_C^{ih}]^T$. The binary variable $DS_C^i$ indicates that if the $j$-th constructive user is deployed on edge server $R_i$. We assume that all constructive users are deployed on the edge instead of the cloud, thus $\sum_{i=1}^k DS_C^i = 1$ for $\forall j = 1, \ldots, h$.

Therefore, the deployment scheme decision $DS$ can be defined as

\[
DS = \begin{bmatrix}
    DS_R \\
    DS_S \\
    DS_C \\
  \end{bmatrix}_{(1+n+h) \times k}
\]

\[
= \begin{bmatrix}
    DS_R \\
    DS_M \\
    DS_S/M \\
    DS_C \\
  \end{bmatrix}_{(1+n+h) \times k}
\]
In addition to the previous definitions, we also define a few symbols for convenience.

- $R(M)$: the edge server deploying the middleware;
- $R_{U_i}(t_i)$: the closest edge server to user $U_i$ that deploying instances can execute $t_i$;
- $R(t_i)$: the edge server can process $t_i$, with the shortest summation of distances between the current server and other servers executing the predecessor tasks of $t_i$, i.e., the server executing task $t_i$;
- $R(C_i)$: the edge server deploying constructive member $C_i$;
- $R(U_i)$: the edge server closest to user $U_i$;

### 4.2. Time Cost Model

#### 4.2.1. Service Response Model

In terms of service response, we consider two factors: the network communication time between users and servers and between tasks; the queuing time of tasks at their corresponding instances calculated by $M/M/c$ queuing model, where $c \geq 1$.

- Service communication model:
  Without the loss of generality, we assume that a $j$-th type of service request $S_{ij}$ from user $U_i$ is always first scheduled to $R_{U_i}(t_{ij})$ to process the initial task $t_{ij}$, for $\forall i = 1, \cdots, u$ and $\forall j = 1, \cdots, m$. The data size from $U_i$ to $R_{U_i}(t_{ij})$ is $ms_{in}^{\text{Sij}}$. Subsequently, each task $t_{ijk}$ after $t_{ij}$ will be scheduled to $R(t_{ijk})$ in topological order, and the data dependency scale between this task and its predecessor task $t_{ijp}$ is $ms_{tr}^{\text{Sij}}$. When the service request is finished, the result, i.e., the output of $t_{ijp}$ is sent to user $U_i$.
  Therefore, the communication time caused by the service request $S_{ij}$ is

$$
t_{ij}^C = \left\lceil \frac{a_{\text{in}}}{a} \right\rceil ms_{\text{in}}^{\text{Sij}} + \sum_{k=2}^{N_{S_{ij}}} \sum_{t_{ijp} \in \text{pred}(t_{ijk})} \left[ \frac{a_{R(t_{ijp}),R(t_{ijk})} ms_{tr}^{\text{Sij}}}{a} \right] \frac{ms_{out}^{\text{Sij}}}{BW}$$

where $a_{\text{in}} = a_{R(U_i),R(t_{ij})}$ and $a_{\text{out}} = a_{R(t_{ij}),R(U_i)}$. $N_{S_{ij}}$ represents the number of small tasks in this request and $a$ and $BW$ are the network distance of one hop and the network bandwidth, respectively. In the network environment uniformly configured for training simulations, they are all constants.

- Service queuing model: If a task is scheduled to a busy instance, it will suffer a long queuing time. To simplify the problem without the loss of generality, we assume that the calculation time of all MS instances is consistent, which is equal to $\mu$. Thus, the waiting time of task $t_k$ is calculated by $M/M/c$ queuing model as

$$
t_{ik}^W = \frac{\rho_k}{\lambda_k(c_k - 1)(c_k - \rho_k)^2}
$$

where $\lambda_k$ represents the arrival rate of tasks of the same type as $t_k$ and $\rho_k = \lambda_k/\mu \leq 1$. $c_k \geq 1$ indicates the number of instances that can process $t_k$ deployed on the current server. Note that if a task is scheduled to the cloud rather than an edge server, we ignore its queuing time but emphasize the data transmission time. The parameter $\rho_k$ that can be calculated as

$$
\rho_k = \left( \sum_{n=0}^{c_k-1} \frac{\rho_k^n}{n!} + \frac{\rho_k^{c_k}}{c_k!} \left( \frac{c_k}{c_k - \rho_k} \right) \right)^{-1} \tag{4}
$$

Therefore, the total queuing time of all tasks in the $j$-th type service request from user $U_i$, $S_{ij}$, can be represented as

$$
\sum_{k=1}^{N_{S_{ij}}} t_{ik}^W
$$

\tag{5}
Based on the proposed service communication model and service queuing model, the total data transmission time caused by users’ service requests can be formulated as

\[ t^S = \sum_{i=1}^{u} \sum_{j=1}^{m} d_{ij} (t^C_{ij} + t^W_{ij}) \]  

where \( d_{ij} \) represents the demand for service request \( S_j \) of \( U_i \).

### 4.2.2. User Interaction Model

In training simulation, a user can subscribe to messages of any other user on demand. If user \( U_i \) subscribes to the messages of \( U_j \), indicating that the results of service requests submitted by \( U_j \) will be sent to \( R(U_i) \) from \( R(U_j) \), we define \( \text{Ita}_{ij} = 1 \). Thus, the time cost of the subscription of \( U_i \) to \( U_j \)’s messages due to the service request \( S_l \) from \( U_j \) is

\[ t^M_{ijl} = \left\lceil \frac{a_{R(U_i), R(U_j)} - ms_{\text{msl}}_{\text{msg}}}{BW} \right\rceil \text{ms} \]  

Therefore, the time cost of the subscription of \( U_i \) to \( U_j \) can be calculated as

\[ t^M_{ij} = \sum_{l=1}^{m} t^M_{ijl} d_{jl} \]  

Furthermore, considering the subscription relationship between users, the interaction time cost is formulated as

\[ t^M = \sum_{i=1}^{u} \sum_{j \neq i} (t^M_{ij} \text{Ita}_{ij} + t^M_{ji} \text{Ita}_{ji}) \]  

In summary, the first objective function, total time cost due to data transmission between tasks among each service request from all users and interactions between them, can be represented as

\[ f_1(DS) = t^S + t^M \]  

### 4.3. Resource Occupancy Rate Model

The software resources that may be deployed on a server include simulation services and constructive members, whose resource demands are represented by \( r_{MS} \) and \( r_{C} \), respectively. Once a deployment scheme is determined, the occupied resources of server \( R_i \) can be modeled as a 3D column vector:

\[ B_i = \sum_{j=1}^{u} D^MS_{ji} r_{MSj} + \sum_{j=1}^{h} D^C_{ji} r_{Cj} \]  

Assuming that server \( R_i \)'s resource consumption rate is \( \text{max}(B_i / R_i) \). The second objective function, the resource usage rate of all edge servers, can be formulated as

\[ f_2(DS) = \max_{i=1, \ldots, \lambda} \{ \text{max}(B_i / R_i) \} \]  

### 4.4. Problem Formulation

The object of deploying SRR jointly is to minimize the overall time cost of the training simulation system while minimizing the resource consumption rate of edge servers. At the same time, constraints of resource capacities of edge servers, deploying rules, and variable
properties should be satisfied. The joint deployment problem \( P \), which has been proven to be NP-Hard [13], is formally defined as follows:

\[
\begin{align*}
\min & \quad f(DS) = (f_1(DS), f_2(DS)) \\
\text{s.t.} & \quad \max(B_i/R_i) < 1, \quad \forall R_i \in R \\
& \quad (DS_i^R - DS_j^R)^2 > 0, \quad \forall i, j = 1, \ldots, k \quad \text{and} \quad i \neq j \\
& \quad \sum_{i=1}^{k} DS_i^M = 1 \\
& \quad \sum_{j=1}^{k} DS_{ij}^C = 1, \quad \forall C_i \in C \\
& \quad DS_i^R \in \{1, \ldots, q\}, \quad \forall i = 1, \ldots, k \\
& \quad DS_i^M \in \{0, 1\}, \quad \forall i = 1, \ldots, k \\
& \quad DS_{ij}^C \in N, \quad \forall i = 1, \ldots, n \quad \text{and} \quad j = 1, \ldots, k \\
& \quad DS_{ij}^C \in \{0, 1\}, \quad \forall i = 1, \ldots, h \quad \text{and} \quad j = 1, \ldots, k
\end{align*}
\]

The related constraint formula is as follows: Equation (14) means that the software resources deployed on an edge server cannot exceed the server’s capacity limit. Equation (15) indicates that one BS can only deploy one edge server at most, that is, the BSs where two edge servers are deployed are different. Equations (16) and (17), respectively, specify that the middleware service and a constructive member must and can only be deployed on one edge server. Equations (18)–(21) specify the range of values for various decision variables, which must all be integers.

Obviously, \( P \) is a nonlinear integer optimization problem that requires finding the Pareto optimal solution of the training simulation system’s time cost and edge servers’ resource occupancy while not violating constraints, e.g., Equations (15)–(21).

5. Simulation Run-Time Resources Deployment Algorithm

In this section, we elaborate on our Learnable Genetic Algorithm for Simulation Run-time Resources Deployment based on NSGA-II, called LGASRD, for \( P \). Firstly, we propose the framework of LGASRD and analyze its differences from NSGA-II. Then, based on the encoding method represented as Equation (1), we introduce some novel crossover and mutation operators and let LGASRD adaptively choose the operator with better performance. The details are presented as follows.

5.1. The LGASRD Algorithm

LGASRD is implemented based on the well-known Non-dominated Sorting Genetic Algorithm II (NSGA-II). As an evolutionary algorithm, NSGA-II requires multiple iterations before termination, has a poor ability to escape from local optimal solutions, and cannot guarantee a feasible solution when faced with constrained optimization problems. Furthermore, other improved methods have not completely solved this problem. For example, while accelerating the convergence speed of the algorithm, there is a high possibility of sacrificing the diversity of the population, which can result in the inability to find good solutions. We introduce the elite knowledge and operator performance knowledge to speed up the convergence rate without too much loss of diversity of the population. The constraint handling mechanism is also designed to guarantee the feasibility of obtained solutions.

5.1.1. Constraint Handling Mechanism

As shown in Equation (1), this encoding method makes it easy to ensure that some constraints are satisfied by reasonable crossover and mutation operators, which mainly restrict the properties of variables and the relationships between variables, as formulated in Equations (15)–(21). However, it is difficult to satisfy the server resource occupancy
constraints, e.g., Equation (14), by intuition. In some cases where server resources are limited, we must adopt strategies to ensure that the solution obtained is feasible.

To address this issue and preserve the diversity of the population as much as possible, we have divided the algorithm into two stages. In the first stage, we transfer the two objectives into one by weighting them using $\omega_1$ and $\omega_2$ and apply a penalty function to penalize solutions that violate the resource capacity constraint. To enhance the effectiveness of this mechanism, the penalty coefficient $M$ increases exponentially with the iteration of the algorithm. Therefore, the objective function in the first stage is

$$f(DS) = f_1(DS)\omega_1 + f_2(DS)\omega_2 - \max\{0, f_2(DS) - 1\}M$$

(22)

In the second stage, we consider both $f_1(DS)$ and $f_2(DS)$. To enhance the algorithm’s exploratory ability, we apply a parameter $\epsilon$ to relax the resource occupancy rate constraint and make $\epsilon$ rapidly approach 0 with each iteration of the algorithm. This ensures the feasibility of the solution obtained.

To ensure the availability of feasible solutions, our basic idea is to accept some unfeasible solutions in the early stages of the algorithm to increase the diversity of individuals in the population, while focusing more on the feasibility of solutions as the algorithm iterates.

5.1.2. Elite Knowledge

Elitist refers to the individual in a population that has the highest fitness value, e.g., the optimal objective function, from the start of evolution until now. It possesses the best genetic structure and excellent traits. To prevent the destruction or loss of elitists in the current population during subsequent iterative optimization processes, the elite retention strategy should be adopted, which involves adding an elitist from each generation to the Elite Population (EP). Rudolph has theoretically proven that the standard genetic algorithm with elite retention strategy is globally convergent [34]. Therefore, we use the individuals in EP, called elite knowledge, to guide the subsequent optimization. There are two aspects to consider when using elite knowledge: updating EP and migrating elitists.

The size of EP is $E$, and the top $E$ individuals from the first generation population can be selected to form the first generation EP. Starting from the second-generation population, if there is an individual A in the current population that is superior to one of the individuals in EP, we replace this elitist with A. In the multi-objective stage, we consider that individual A is superior to individual B if one of the following three conditions is met:

- **Condition 1:**
  $$f_1(A) \leq f_1(B)$$
  $$f_2(A) \leq f_2(B)$$
  $$A \notin EP$$

- **Condition 2:**
  $$f_1(A) < f_1(B)$$
  $$f_2(A) \geq f_2(B)$$
  $$\frac{f_1(B) - f_1(A)}{f_1(B)} > \frac{f_2(A) - f_2(B)}{f_2(B)}$$
  $$randp < 1 - \frac{f_2(A) - f_2(B)}{f_2(B)} \frac{f_1(B) - f_1(A)}{f_1(B) - f_1(A)}$$

(24)
• Condition 3:

\[
\begin{align*}
  f_1(A) & \geq f_1(B) \\
  f_2(A) & < f_2(B) \\
  \frac{f_1(A) - f_1(B)}{f_1(B)} & < \frac{f_2(B) - f_2(A)}{f_2(B)} \\
  \text{randp} & < 1 - \frac{f_1(A) - f_1(B)}{f_1(B)} \frac{f_2(B)}{f_2(B) - f_2(A)}
\end{align*}
\]  

(25)

where \( \text{randp} \) is a randomly generated probability. Equation (23) represents the case that A dominates B and no individual the same as A is in EP. Equations (24) and (25) mean that although A does not perform well in terms of \( f_1 \) or \( f_2 \), it performs exceptionally well in terms of another objective function. Take Equation (24) as an example, it indicates that although A is worse (larger) than B in terms of \( f_2 \), it is better (smaller) than B in terms of \( f_1 \) and the degree to which A is smaller than B for \( f_1 \) is greater than the degree to which B is smaller than A for \( f_2 \).

In addition to trying to replace the inferior individuals in EP with outstanding ones in the current population to ensure the excellence of EP, we hope to use EP to guide the subsequent optimization process, e.g., elite migration. Every ten rounds of iteration in the second stage, we randomly select an elitist located in the top Pareto rank of EP and replace one individual in the lowest Pareto rank in the current population with it. This approach can bring the population closer to EP to some extent and accelerate the algorithm’s convergence speed without too much loss of diversity within the population.

5.1.3. Operator Performance Knowledge (OPK)

In the genetic algorithm, three operations are necessary: parent selection, crossover, and mutation. For a particular problem, using different operators to perform these three operations can result in different outcomes. In standard NSGA-II [35], the operators of performing these three operations are fixed, and the algorithm can only perform them in one way. This makes the algorithm lack adaptive ability and learning ability. To improve this, we introduce OPK into NSGA-II, allowing the algorithm to record the performance of operators for each operation. The higher the historical performance of an operator, the greater the probability it will be selected during the corresponding operation in a certain iteration.

We design two crossover operators and three mutation operations (detailed later); thus, we can obtain six different genetic methods. During each iteration, the algorithm records which crossover and mutation operator each individual used. If a new individual A dominates the corresponding old one B, that is, \( f_1(\text{new}) \leq f_1(\text{old}) \) and \( f_2(\text{new}) \leq f_2(\text{old}) \), or A performs exceptionally better than B in terms of one objective function, represented as Equation (24) or Equation (25), the performance of the used crossover and mutation operators is increased by one. Take mutation operation as an example, if the current mutation operation is successful through the \( j \)-th mutation operator, the performance of the \( j \)-th mutation operator \( \text{PK}(j) = \text{PK}(j) + 1 \). When selecting an operator to change an individual in the next iteration, the probability of the \( j \)-th operator will be selected as

\[
P(j) = \frac{\text{PK}(j)}{\sum_{i=1}^{3} \text{PK}(i)}
\]

(26)

where \( P(j) \) represents the probability of the \( j \)-th operator is selected. As shown in Table 2, in the first iteration of our algorithm, the performance of each mutation operator is initialized to 1. The probability of each operator being selected is calculated as 33.3% according to Equation (26). If the second operator is selected and the mutated individual is better than the original one during the first mutation operation, the historic performance of it is increased to 2, and the probability of each operator being selected is calculated as 25%, 50%, and 25%. In the second mutation, if the first operator is chosen
but the mutated individual does not become better, the performance and probability of selection of all operators remain unchanged.

Table 2. Update and application performance knowledge of mutation operators.

<table>
<thead>
<tr>
<th>PK Operator</th>
<th>Operator 1</th>
<th>Operator 2</th>
<th>Operator 3</th>
<th>Probability of Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>33.3%</td>
</tr>
<tr>
<td>mutation1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>25%</td>
</tr>
<tr>
<td>mutation2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>25%</td>
</tr>
</tbody>
</table>

5.1.4. The LGASRD Framework

Based on the well-known NSGA-II and the aforementioned operations, we obtain the proposed LGASRD in Algorithm 1. Firstly, we randomly generate the initial population and select \( E \) individuals performing well to create EP based on their fitness values (lines 3–5). After that, LGASRD starts the first phase of iteration. At the beginning of the iteration, executes three genetic operations, creating the new population. Note that LGASRD uses the roulette wheel strategy to perform the parent selection operation (line 7). Next, the fitness values of the new population are calculated, and then OPK and EP are updated (lines 8–10). At the final step of this phase, some parameters are updated (line 11). After the first stage, two objective functions of the current population are calculated and a new EP is created (lines 14–15). Compared to the first stage, there are several differences between the second stage of the algorithm:

- LGASRD uses tournament strategy for parent selection in the second stage (line 17);
- As the number of objective functions increases, some evaluation criteria become more complex, such as how to update OPK and EP (line 19);
- Elite migration strategy is introduced (lines 25–27);
- LGASRD stops iterating after multiple failed attempts to update EP (lines 28–30).

5.2. Crossover Operators

We design two crossover operators, which can be understood as vertical and horizontal combined with the encoding method shown as Equation (1).

The first operation is to exchange the three parts given in Equation (1) from the two parent individuals that will crossover with each other. It should be noted that the two parts related to micro-services and constructive members cannot be completely exchanged, or this crossover operation will be invalid. Instead, we will randomly select a range and exchange the parts that are within this range.

The second approach involves randomly selecting a server range and swapping the deployment schemes of servers within the range in the two parent chromosomes. However, this crossover operation may result in the newly obtained individuals violating certain constraints, such as Equations (14)–(17). Therefore, some repair mechanisms are introduced to ensure the feasibility of the resulting solutions.

These two crossover operators enable the exchange of chromosome segments between the two parent bodies in different ways, allowing LGASRD to adaptively select the crossover approach with better historic performance compared to using only one operator. As a result, information exchange between two parent individuals is more efficient.

5.3. Mutation Operators

We design two crossover operators, which can be understood as random and heuristic. The first approach is random. It randomly selects two servers within the mutated chromosome, swapping the BSs deploying them, and randomly changing the locations of the middleware and constructive members. As for micro-services, it randomly selects a server and decides whether to increase or decrease the number of micro-services deployed.
on it based on its resource usage rate, i.e., the second optimization objective. The purpose of designing this operator is to enhance the randomness of our algorithm, thereby increasing its ability to escape from local optimal solutions.

Algorithm 1 Learnable GA-based Simulation Resources Deployment (LGASRD)

Require: Set of micro-services $MS$; Resource demand of micro-services $r_{MS}$; Set of composite services $S$; Set of edge servers $R$; Resource capacity of edge servers $RC$; Location of base stations $LBS$; Set of users $U$; Resource demand of constructive members $RC$; Location of live and virtual members $L_{LV}$; User demand for composite services $d$; Counts of interactions among users $Ita$

Ensure: Deployment schemes $DS$ and $EP$

1: $g \leftarrow 1$
2: $FailCount \leftarrow 0$
3: Randomly generate $P$ individuals $DS_1, \ldots, DS_P$ to form the population $DS$
4: Calculate $f(DS)$ according to Equation (22)
5: Select the top $E$ individuals in $DS$ to form the first generation EP
6: while in the single-objective stage do
7: Perform parent selection, crossover, and mutation operations to generate the next generation $DS$ according to OPK
8: Calculate $f(DS)$
9: Update OPK based on $f(DS)$
10: Update EP based on $DS$ and $f(DS)$
11: Update parameters including $M, \epsilon$ and $Pm$
12: $g \leftarrow g + 1$
13: end while
14: Calculate $f_1(DS)$ and $f_2(DS)$ according to Equations (10)–(12)
15: Select the top $E$ individuals in $DS$ based on $f_1(DS), f_2(DS)$ and their Pareto rank and congestion to form EP in the second stage
16: while in the multi-objective stage do
17: Perform parent selection, crossover, and mutation operations to generate the next generation $DS$ according to OPK
18: Calculate $f_1(DS), f_2(DS)$ according to Equations (10)–(12)
19: Update EP and OPK according to Equations (23)–(25)
20: if fail to update EP then
21: $FailCount \leftarrow FailCount + 1$
22: else
23: $FailCount \leftarrow 0$
24: end if
25: if $g\%Replace\_Freq == 0$ then
26: Replace one of the worst individuals in $DS$ with one of the best individuals in EP
27: end if
28: if $FailCount == G \times Stop\_Ratio$ then
29: break
30: end if
31: Update parameters including $M, \epsilon$ and $Pm$
32: $g \leftarrow g + 1$
33: end while
34: return $DS, EP$

The second variation operation is heuristic. It defines the weighted distance between live and virtual users and a BS, using the size of the communication data generated by these users as the weight to add up the physical distance between these users and the BS. Subsequently, it can be determined which BS currently deploying servers is the farthest from these users, and then a closer, idle BS will be selected to replace it. Additionally, with resource availability, the middleware and as many constructive members as possible will be deployed on the BS with the smallest weighted distance. Evidently, this heuristic
operator primarily aims to reduce communication time caused by user interactions. We have also devised operations for other objectives, including service queuing time, service communication time, and resource occupancy rate, which are similar to the current operator and hence will not be elaborated on further. These operators can make use of some heuristic information to enable our algorithm to quickly find some not-bad solutions. In other words, while guaranteeing the lower bound of the obtained solution, they greatly speed up the convergence rate of LGASRD.

The third approach utilizes the information of EP, allowing the current variant individual to either become more similar to or less similar to one elitist. If the current individual’s crowding distance is not equal to 0, it learns from the elitist and becomes more similar to it. For example, if the elitist deploys the middleware on server A, the variant individual will deploy it on B, which is the closest server to A among all servers. If the current individual’s crowding distance is equal to 0, we want it to be unique. In other words, we will let it be different from other individuals in the current population. From an algorithmic perspective, the entire population tends towards EP, so we can make the current variant individual distant from them by reducing its similarity to the imitated elitist. If the variant individual overlaps with its adjacent individuals (solutions), this mutation operator increases the diversity of the population. Otherwise, when the current individual is approaching the imitated elitist, it may obtain better objective function values.

6. Experiment Evaluation

In this section, we report evaluation results using real-world data sets to verify the superiority of the proposed LGASRD. We first introduce the benchmark policies for evaluation. Then, we describe the data set and experimental settings. Finally, the detailed experiment results of LGASRD with respect to the compared methods are presented and discussed.

6.1. Benchmark Policies

Referring to the experimental evaluation in [8,10,13,35], we provide the following five representative baselines for comparative studies.

- K-Means Clustering + NSGA-II (KMNSGA): This strategy first deploys edge servers on \( k \) BSs using the K-Means clustering algorithm. On this basis, it uses NSGA-II to determine how other software resources should be deployed.
- NSGA-II: This approach is to directly use NSGA-II to solve the deployment problem for SRR we are facing.
- K-Means Clustering + Top-k Service Placement (TServiceD): This algorithm first deploys \( k \) edge servers using the K-Means clustering algorithm. Then, it greedily deploys micro-services by first deploying the MS with the highest user demand, then the service with the second highest user demand, and so on. Finally, it randomly deploys the simulation middleware and constructive members.
- Top-V Server Deployment + Top-V Service Deployment (TSD\(^2\)): This method first selects the \( k \) most heavily used BSs to deploy edge servers. Then, it greedily deploys micro-services by prioritizing the MS with the highest user demand, followed by the second highest, and so on. Finally, like TServiceD, it also randomly deploys the simulation middleware and constructive members.
- Random: This policy randomly determines the deployment scheme while ensuring its feasibility.

6.2. Data Sets

In our evaluation, we utilized the EUA data set [15], which was derived from the Wireless Communications Licensing data set published by the Australian Communications and Media Authority (ACMA). This data set contains detailed information about the geographic locations of all cellular base stations in Australia, including their respective latitudes and longitudes. We randomly selected \( q \) base stations for use in our experiment,
which we referred to as base stations \( M \). To model user distribution, we drew upon the distribution generated by the EUA data set, which comprised \((u - h)\) users.

To simulate the business logic relationships between multiple tasks in a service request, we used five workflows, provided by [16], that have been widely used in related research: Montage, Cybershake, Epigenomics, LIGO, SIPHT. Each workflow has a certain number of tasks, and each task can only be executed by containers that encapsulate the corresponding MS. The number of different types of micro-services included in each workflow is shown in Table 3.

<table>
<thead>
<tr>
<th>Workflow (Service Request)</th>
<th>Number of Micro-Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Montage (WfM)</td>
<td>9</td>
</tr>
<tr>
<td>Cybershake (WfC)</td>
<td>5</td>
</tr>
<tr>
<td>Epigenomics (WfE)</td>
<td>8</td>
</tr>
<tr>
<td>LIGO (WfL)</td>
<td>4</td>
</tr>
<tr>
<td>SIPHT (WfS)</td>
<td>13</td>
</tr>
</tbody>
</table>

6.3. Experimental Settings

All the experiments are implemented in Python 3.10.9 on a desktop computer equipped with a 2.30 GHz Intel Core i7-11800H CPU and 16 GB RAM (The source code is available by accessing https://github.com/ZkofZhang/An-Approach-for-Deployment-of-Service-Oriented-Simulation-Run-time-Resources (accessed on 1 September 2023)). The parameter settings are discussed as follows.

We conducted experiments with various scales, involving scale parameters such as the number of tasks in a workflow (\( N_t \)), the number of micro-services (\( N_{ms} \)), the number of available edge servers (\( N_{es} \)), and the number of users (\( N_u \)), i.e., simulation members, and the number of available BSs that was fixed to 50. Users can be divided into live, virtual, and constructive users. In our experiments, we do not distinguish between live and virtual users and use normal users in the EUA data set to represent them. Moreover, for simplicity, we assume that the number of these two types of users is the same as that of constructive users. To facilitate description, we summarized the scale parameter settings for each set of experiments, as shown in Table 4. Note that the middleware is a type of MS that exists in all settings, so it does not appear in the table.

<table>
<thead>
<tr>
<th>Setting</th>
<th>( N_t )</th>
<th>( N_{ms} )</th>
<th>( N_{es} )</th>
<th>( N_u )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>50</td>
<td>WfM</td>
<td>10</td>
<td>10 + 10</td>
</tr>
<tr>
<td>B</td>
<td>50</td>
<td>WfM</td>
<td>15</td>
<td>30 + 30</td>
</tr>
<tr>
<td>C</td>
<td>50</td>
<td>WfM</td>
<td>20</td>
<td>50 + 50</td>
</tr>
<tr>
<td>D</td>
<td>50</td>
<td>All except WfC</td>
<td>10</td>
<td>10 + 10</td>
</tr>
<tr>
<td>E</td>
<td>50</td>
<td>All except WfC</td>
<td>15</td>
<td>30 + 30</td>
</tr>
<tr>
<td>F</td>
<td>50</td>
<td>All except WfC</td>
<td>20</td>
<td>50 + 50</td>
</tr>
<tr>
<td>G</td>
<td>50</td>
<td>All</td>
<td>10</td>
<td>10 + 10</td>
</tr>
<tr>
<td>H</td>
<td>50</td>
<td>All</td>
<td>15</td>
<td>30 + 30</td>
</tr>
<tr>
<td>I</td>
<td>50</td>
<td>All</td>
<td>20</td>
<td>50 + 50</td>
</tr>
<tr>
<td>J</td>
<td>100</td>
<td>WfM</td>
<td>10</td>
<td>10 + 10</td>
</tr>
<tr>
<td>K</td>
<td>100</td>
<td>WfM</td>
<td>15</td>
<td>30 + 30</td>
</tr>
<tr>
<td>L</td>
<td>100</td>
<td>WfM</td>
<td>20</td>
<td>50 + 50</td>
</tr>
</tbody>
</table>

In our experiments, the resources required by each MS are directly read from the corresponding XML file. In the service-oriented training simulation, all members rely on requesting services to achieve their desired functions, and the execution process of services is not performed locally for constructive members. Therefore, they do not require a large
amount of CPU resources for calculation. On the contrary, they need a certain amount of memory resources to store a large number of data. Taking this factor into account, we assume that each constructive member only needs one unit CPU resource and does not need any GPU resources. The amount of memory resources they require, which is randomly generated, is between 0.5 and 1.5 times the average memory required by all micro-services. Based on the above settings, we can obtain the total amount of resources consumed in the simulation system and set the resource capacity of each edge server to be between 0.05 and 0.75 times this total amount, reflecting the heterogeneity among servers. At the same time, we set the demand for various service requests by each member to be a random integer between 0 and 20. During our experiments, we also adopted an extreme scenario where all members needed to interact with each other by sending messages.

In LGASRD, we set the population size $P$ to 50, and the maximum number of iterations $G$ to 250. Every 5 generations, we select an elitist to replace one individual in the lowest Pareto rank of the current population. If EP fails to be updated for consecutive 0.15$G$ generations, LGASRD will terminate.

6.4. Experiment Results

We conducted extensive experiments to demonstrate the advantages of LGASRD over comparison methods. Firstly, we presented the results of six algorithms under settings A to L to illustrate that LGASRD has better performance than others over multiple problem scales. Then, we reduced the resources provided by edge servers to verify the effectiveness of the constraint-handling mechanism in LGASRD. Under one experimental setup, each algorithm was run 20 times, and we took the average of these 20 runs.

6.4.1. Optimality and Scalability

Figures 3–5 present the two objective function values of solutions in the highest Pareto rank generated by the mentioned six algorithms in the upper half when the number of micro-services is 10, 35, and 40, respectively. They also illustrate the iteration process of three evolutionary algorithms, including KMNSGA, NSGA-II, and LGASRD, in the lower half. Due to the fact that some parameters related to resources and user demand were randomly generated for each run, TServiceD and TSD2 obtained different results even though they have specific deployment rules. In these figures, we present the results of LGASRD in two forms: the normal population and the elite population, represented by red and green stars, respectively. Note that since the solutions obtained by LGASRD are too dense, they appear as a single point in the figures. Therefore, we enlarged the points representing the results calculated by LGASRD and placed them in a black box in the blank area.

As can be observed from Figures 3–5 and Table 5, LGASRD exhibits the best comprehensive performance among the six algorithms tested. In experiments conducted with settings A, C, and D, both two objective functions achieved minimal values by LGASRD. This fully demonstrates the effectiveness of the strategies we used, such as elite knowledge and OPK, in enhancing the performance of NSGA-II. Under other experimental settings, LGASRD always obtained solutions that were not dominated by those of other algorithms, with a minimum value for a specific objective function.
Figure 3. Algorithm performance comparison and iteration process of three evolutionary algorithms under experimental settings A, B and C.

Figure 4. Algorithm performance comparison and iteration process of three evolutionary algorithms under experimental settings D, E and F.
Figure 5. Algorithm performance comparison and iteration process of three evolutionary algorithms under experimental settings G, H and I.

Table 5. Performance of mentioned algorithms on two objective functions across nine experimental settings.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Best on O1</th>
<th>Better than LGASRD on O1</th>
<th>Best on O2</th>
<th>Better than LGASRD on O2</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMNSGA</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>LGASRD</td>
<td>7</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TServiceD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TSD^2</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Under these experimental settings, the performance of two other evolutionary algorithms, including KMNSGA and NSGA-II, and TServiceD, was also good and they sometimes obtained smaller values on another objective function than LGASRD. The reason for this may be that when the scale of experiments increased, LGASRD reached the termination condition before fully converging, failing to obtain better solutions. It can also be observed that Random, TServiceD, and TSD^2 often produce unfeasible solutions. This may be due to their failure to consider the resources required by the simulation middleware and constructive members when using heuristic deployment rules for micro-services, thus directing a large amount of resources towards the deployment of micro-services.

Under three experimental settings where the number of micro-services is consistent, i.e., experiments presented in Figures 3–5, as the number of edge servers and simulation members increases, the time cost within the system also increases. This is normal, as an increase in the number of elements within the simulation system will inevitably result in an increase in the scale and frequency of requests for services and data transmission.

In order to explore this in more detail, we summarize how the time costs of Pareto optimal solutions obtained by three evolutionary algorithms change as the scale of our experiment varies, as shown in Table 6. The experimental scale mentioned here mainly refers to the product of the number of users and edge servers, as well as the number of micro-services. In Table 6, they are presented in the left half and the right half, respectively. To provide a visual representation, we have only provided ratios of time costs under
different scales without providing specific numerical values. We are only interested in the impact of the experiment scale on time cost and not comparing the performance of the mentioned algorithms, so the results for each algorithm are independent. Due to the limited data, it is difficult to quantitatively describe the growth trend of time cost with experimental scale increases. However, it is evident that the increase in time cost caused by the increase in the number of micro-services is significant, and the underlying reasons still need to be explored. On the contrary, the impact of the number of edge servers and users on time cost is relatively small, even though when the quantity of edge servers and simulation users increases, the increment of time cost exceeds the increase in experimental scale.

**Table 6.** Changes in Time Cost with the Increase in the Number of Edge Servers, Users, and Micro-services.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Number of Edge Servers and Users</th>
<th>LGASRD</th>
<th>Number of Micro-Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>4.5</td>
<td>8.17</td>
<td>5.76</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>37.14</td>
<td>22.62</td>
</tr>
<tr>
<td>D</td>
<td>4.5</td>
<td>7.87</td>
<td>6.77</td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>32.74</td>
<td>18.98</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>4.5</td>
<td>5.17</td>
<td>3.09</td>
</tr>
<tr>
<td>H</td>
<td>10</td>
<td>12.74</td>
<td>9.13</td>
</tr>
</tbody>
</table>

From the lower part of Figures 3–5, it can be observed that, unlike KMNSGA and NSGA-II, LGASRD exhibits significant fluctuations in its iteration curve. After reaching a local minimum, LGASRD often continues to reduce the objective function value through guided by EP and OPK. This indicates that LGASRD possesses a stronger ability to escape local optimal solutions compared to the other two evolutionary algorithms, which is one of the main reasons for its superior performance. Furthermore, compared to KMNSGA and NSGA-II, LGASRD also exhibits a faster convergence speed. When EP fails to update consecutively for multiple generations, the algorithm terminates even if the normal population is still becoming better.

We also conducted experiments with settings J, K, and L, and the results are shown in Figure 6. It can be observed that LGASRD still exhibits excellent optimization performance, with its solutions dominating those generated by other comparison algorithms, except for slightly higher time cost compared to the solutions produced by TServiceD and TSD\(^2\) in Figure 6b. Furthermore, if we compare the numerical values of time cost in Figures 3 and 6, we will find that if other settings are the same, the granularity of service requests does not have a significant impact on the magnitude of the training simulation system’s time cost. The granularity of a service request refers to the number of micro-services involved in it, i.e., the number of tasks in a workflow application.
6.4.2. Effectiveness of Constraint Handling Policy

The SRR deployment problem is an optimization problem with constraints. When solving this problem, certain rules can be set to ensure that the obtained solution does not violate the constraints. However, this approach is ineffective for constraints like the one shown in Equation (14). Therefore, additional constraint-handling methods, which have already been discussed in Section 5.1.1, need to be introduced into our algorithm.

To validate the effectiveness of the constraint-handling approach used in LGASRD, we conducted experiments using some of the previous experimental settings. In contrast to previous experiments, we reduced the resource capacity of edge servers to one-tenth of their original capacity to simulate a resource-constrained scenario. Previous experiments’ results have shown better performance for three evolutionary algorithms, while Random, TServiceD, and TSD² performed poorly and generated infeasible solutions under certain settings. Therefore, we will not discuss them further in this section.

As shown in Figure 7, in most cases, KMNSGA and NSGA-II can only generate a fraction of feasible solutions, or even produce entirely infeasible solutions, as illustrated in Figure 7a. In contrast, LGASRD is capable of producing feasible solutions in all scenarios, with the minimal time cost among the three algorithms. This clearly demonstrates the effectiveness and significance of the constraint-handling mechanism we employed.

In Figure 8, we present a comprehensive depiction of the time costs of solutions obtained from three algorithms in resource-constrained scenarios. The training simulation system’s time cost is the summation of service communication time, service queuing time, and user interaction time. We can observe that the time spent on queuing at MS instances is relatively small compared to the data transmission time incurred due to the system’s response to user-initiated service requests. Moreover, these two types of time costs do not exhibit significant increments with the increase in the granularity of service requests, the number of servers, and simulation users. In contrast, the time resulting from user interaction tends to increase significantly with the increase in the number of edge servers and users. This is evident. If micro-services are deployed properly, users may experience low service latency. However, as the number of simulation members increases, it inevitably leads to an increase in the frequency of interactions between users, resulting in a significant growth in user interaction time.
7. Conclusions

This paper investigates the deployment problem of SRR in the cloud-edge collaborative computing architecture, effectively improving the user experience during training simulation activities. Firstly, we formulate this problem as a multi-objective optimization problem, aiming to simultaneously minimize the time cost of the training simulation system and the resource occupancy rate of edge servers under various constraints. It is worth noting that we adopt a combinatorial service modeling approach, which is unique and valuable. Subsequently, we propose LGASRD to solve this problem. The algorithm is a two-stage approach that incorporates elite knowledge and operator performance knowledge, enabling it to adaptively learn and find better solutions at a faster pace. Extensive experiments based on real-world and widely used data sets demonstrate the superiority of LGASRD compared to the benchmark algorithms across multiple evaluation metrics. In future work, we plan to validate our algorithm in training simulation prototype systems, and explore more complex and coupled research by integrating resource deployment with task scheduling.
Author Contributions: Conceptualization, Z.Z. and Y.P.; methodology, Z.Z. and M.Z.; validation, Z.Z.; investigation, Z.Z.; data curation, Z.Z.; writing—original draft preparation, Z.Z.; writing—review and editing, Z.Z. and M.Z.; supervision, Q.Y. and Q.L.; funding acquisition, Q.Y. and Y.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Natural Science Foundation of China [grant number 62103425] and Natural Science Foundation of Hunan Province, China [grant number 2022J40559].

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We thank Xiuling Zhang and Hailiang Chen at the National University of Defense Technology for attending discussions and giving us important advice.

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

Abbreviations
The following abbreviations are used in this manuscript:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>Live</td>
</tr>
<tr>
<td>V</td>
<td>Virtual</td>
</tr>
<tr>
<td>C</td>
<td>Constructive</td>
</tr>
<tr>
<td>LGASRD</td>
<td>Learnable Genetic Algorithm for Simulation Run-time Resource Deployment</td>
</tr>
<tr>
<td>SRR</td>
<td>Simulation Run-time Resource</td>
</tr>
<tr>
<td>SRSR</td>
<td>Simulation Run-time Software Resource</td>
</tr>
<tr>
<td>MS</td>
<td>Micro-Service</td>
</tr>
<tr>
<td>BS</td>
<td>Base Station</td>
</tr>
<tr>
<td>EP</td>
<td>Elite Population</td>
</tr>
<tr>
<td>OPK</td>
<td>Operator Performance Knowledge</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>Non-dominated Sorting Genetic Algorithm II</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed Acyclic Graph</td>
</tr>
<tr>
<td>TB</td>
<td>Training Base</td>
</tr>
<tr>
<td>ES</td>
<td>Edge Server</td>
</tr>
</tbody>
</table>

References


*Disclaimer/Publisher’s Note:* The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.