mathematics

MDPI

Article

A–ESD: Auxiliary Edge-Server Deployment for Load Balancing in Mobile Edge Computing

Sen Niu [1], Xuewei Zhang [1], Simin Wang [1], Kaili Liao [1,*], Bofeng Zhang [1,2] and Guobing Zou [3]

[1] School of Computer and Information Engineering, Institute for Artificial Intelligence, Shanghai Polytechnic University, Shanghai 201209, China; niusen@sspu.edu.cn (S.N.); xwzhang@sspu.edu.cn (X.Z.); smwang@sspu.edu.cn (S.W.); bfzhang@sspu.edu.cn (B.Z.)
[2] School of Computer Science and Technology, Kashi University, Kashi 844000, China
[3] School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China; gbzou@shu.edu.cn
* Correspondence: klliao@sspu.edu.cn

Abstract

In recent years, the deployment of edge servers has attracted significant research interest, with a focus on maximizing their utilization under resource constraint to improve overall efficiency. However, most existing studies concentrate on initial deployment strategies, paying limited attention to approaches involving incremental expansion. As user demands continue to escalate, many edge systems are facing overload situations that hinder their ability to meet performance requirements. To tackle these challenges, this paper introduces an auxiliary edge-server deployment strategy designed to achieve load balancing across edge systems and alleviate local server overloads. The problem is herein referred to as the Auxiliary Edge Server Deployment (A–ESD) problem, and the aim is to determine the optimal deployment scheme for auxiliary edge servers. A–ESD is modeled as a multi-objective optimization problem subject to global constraints and is demonstrated to be NP-hard. An enhanced genetic algorithm called LBA–GA is proposed to efficiently solve the A–ESD problem. The algorithm is designed to maximize overall load balance while minimizing total system delay. Extensive experiments conducted on real-world datasets demonstrate that LBA–GA outperforms existing methods, delivering superior load balancing, reduced latency, and higher cost-effectiveness.

Keywords: load balancing; edge computing; edge server deployment; genetic algorithm

MSC: 68W50

check for updates

Academic Editors: Fasheng Zhou and Jing Qiu

Received: 19 August 2025
Revised: 16 September 2025
Accepted: 19 September 2025
Published: 25 September 2025

Citation: Niu, S.; Zhang, X.; Wang, S.; Liao, K.; Zhang, B.; Zou, G. A–ESD: Auxiliary Edge-Server Deployment for Load Balancing in Mobile Edge Computing. *Mathematics* **2025**, *13*, 3087. https://doi.org/10.3390/math13193087

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the rapid advancement of network communication technologies and the Internet of Things (IoT), the volume of data generated on the Internet is experiencing explosive growth. According to IDC projections, the global datasphere is expected to reach 175ZB by 2025, with this value representing more than a five-fold increase over the 33ZB recorded in 2018 [1]. Cloud computing, as an emerging computational paradigm, offers significant advantages in terms of efficiency, flexibility, and scalability. It effectively supports the processing and storage of massive datasets, thereby injecting new vitality into Internet development. However, with the widespread adoption of IoT devices and the growing demand for real-time processing, latency has become a critical bottleneck that limits the applicability of cloud computing in latency-sensitive scenarios.

To mitigate this challenge, edge computing has emerged as a promising complementary paradigm. By relocating computation and services from the centralized cloud to the network edge—closer to end users and data sources [2]— edge computing facilitates localized data processing and analysis. This proximity considerably reduces transmission latency and bandwidth consumption while improving data security and privacy [3]. As a result, edge computing is increasingly regarded as a vital extension of cloud computing, and their integrated application offers a more comprehensive solution for modern information-processing requirements.

In recent years, edge computing has made substantial progress and has been deployed in numerous domains [4]. However, several challenges remain. Edge servers are typically constrained by cost, energy consumption, and computational capacity. Considering the aim of maximizing their utilization under such limitations, the Edge-Server Deployment (ESD) problem has attracted significant research attention [5,6]. A primary objective in tackling the ESD problem is to optimize the placement of edge servers based on user distribution, thereby improving service efficiency and reducing user-perceived latency.

Numerous approaches have been proposed to address the ESD problem. Researchers have developed various deployment strategies, often modeling the task as an optimization problem under specific performance constraints [7–9], such as minimizing latency, maximizing coverage, reducing energy consumption, or lowering costs. Metaheuristic algorithms [10–12] and enhanced genetic algorithms [13–15] are commonly used to identify optimal or near-optimal deployment configurations.

Despite these efforts, existing methods exhibit several limitations. First, most ESD strategies focus primarily on initial server placement, overlooking the potential for server overload as user requests increase over time, a phenomenon that can severely impact the user experience. Second, many current approaches rely on offloading strategies to alleviate load, thus failing to fundamentally address the shortage of computational resources. Finally, the prevailing deployment methods tend to prioritize latency and cost optimization, often neglecting the crucial objective of achieving system-wide load balance under high-demand conditions.

To enhance user experience and mitigate resource shortages, this paper explores the deployment of auxiliary edge servers as a scalable expansion of existing edge infrastructure [16]. Compared to conventional edge nodes, auxiliary edge servers feature a smaller communication radius and reduced capacity, resulting in lower cost and greater deployment flexibility. This expansion strategy is formalized as the Auxiliary Edge Server Deployment (A–ESD) problem. In contrast to prior work, our approach explicitly incorporates load balancing into the problem formulation. To tackle the NP-hard A–ESD problem effectively, we propose a novel genetic algorithm (GA)-based method named LBA–GA that efficiently identifies high-quality approximate solutions. Our method conducts a comprehensive analysis of users and edge servers within the system, incorporating user-demand satisfaction, server load levels, and connectivity conditions. It subsequently determines the optimal placement for auxiliary edge servers to effectively alleviate load pressure.

The main contributions of this paper are summarized below

- We formalize the A–ESD problem as a constrained multi-objective optimization model with an emphasis on load balancing, and establish its NP-hardness.
- We propose LBA–GA, an enhanced genetic algorithm, to efficiently identify optimal deployment strategies for auxiliary edge servers.
- We conduct extensive experiments on widely used real-world datasets to evaluate the performance of LBA–GA. Results demonstrate that our approach outperforms existing methods in achieving better load balance, lower latency, and higher cost-effectiveness.

The remainder of this paper is organized as follows. Section 2 provides a motivational example for this study. Section 3 introduces the system model and problem formulation. Section 4 presents the proposed algorithm and proves the NP-hardness of the A–ESD problem. Section 5 discusses experimental results and analysis. Related work is reviewed in Section 6. Finally, Section 7 concludes the paper and outlines future research directions.

2. Motivating Example

Figure 1 presents an example of the Auxiliary Edge-Servers Deployment scenario. There is an edge system that consists of a set of edge servers $S = \{s_1, s_2, s_3\}$ and a set of users $U = \{u_1, u_2, \ldots, u_8\}$. Every user can access edge servers in the coverage area. Please note that the computing and storage resources on the edge server are limited. Each user has many request tasks, all of which require some amount of resources.

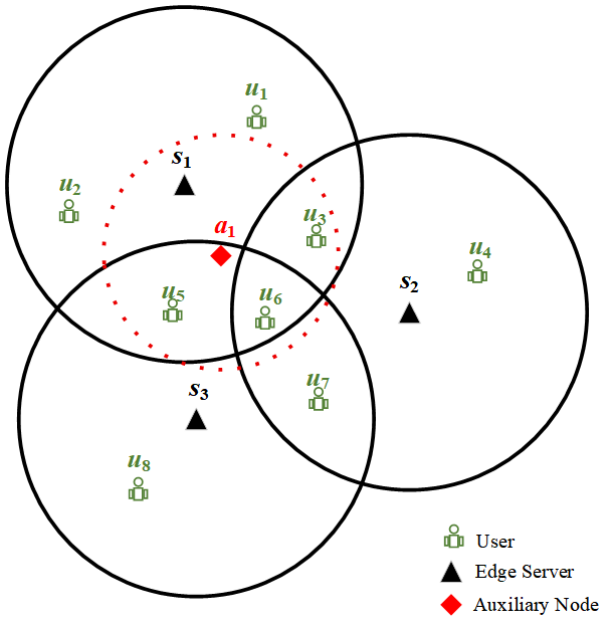


Figure 1. Example edge system.

Existing studies of the ESD problem assume that no edge server exists in the current region. Then, they choose the deployment locations of edge servers to ensure the lowest overall latency for all users. However, in practice, a certain number of edge servers have already been deployed in many regions. The assumption of starting completely from scratch may not reflect reality. Furthermore, most of the research on load balancing uses resource scheduling to satisfy large numbers of user requests. When an edge server is overloaded, it will send the user's request to other free edge servers for processing [17–19]. While these approaches can alleviate the overload on edge servers to some extent, they do not work when all edge servers (s_1, s_2, and s_3) are overloaded. There are then no devices in the system with sufficient computational resources. Under those conditions, it becomes necessary to add new computational resources.

However, the addition of computational resources increases costs, and the timing and circumstances of edge-server overloads are variable. Thus, the problem of how to add new computational resources and the extent to which additional resources should be added are important research questions.

Based on the edge system, an auxiliary edge server, denoted as a_1, is integrated to facilitate communication with users and offload requests, as shown in Figure 1. The auxiliary edge server a_1 can communicate with users u_3, u_5 and u_6. Consequently, these users are able to offload their requests to a_1, thus reducing the load on the original edge

servers that were processing these requests and adjusting the overall load distribution. It follows that the location and number of auxiliary edge servers affect the users they can communicate with, which in turn influences the overall load level. Therefore, examining the location and number of auxiliary edge servers is imperative if one is to determine the optimal placement and quantity for overall load balancing in the system.

3. System Model

3.1. Edge System

The main focus of this research is the auxiliary edge-server deployment problem, as presented within the context of pre-existing edge systems. The relevant symbols are shown in Table 1. An edge system consists of a set of edge servers and users.

Table 1. Symbol table.

Symbol	Definition
$S = \{s_1, s_2, \ldots, s_m\}$	Set of edge servers
(X_{max}, Y_{max})	Maximum values of the system on the X and Y axes
$U = \{u_1, u_2, \ldots, u_n\}$	User set
$Pos_{s_k} = (X_{s_k}, Y_{s_k})$	Position of edge server s_k
$L_S = \{L_{s_1}, L_{s_2}, \ldots, L_{s_m}\}$	Set of edge server load levels
$l_{s_k} = \{l_{s_k}^{max}, l_{s_k}^{now}\}$	Max and current loads of edge server s_k
R_{s_k}	Communication range of edge server s_k
(X_{u_k}, Y_{u_k})	Position of user u_k
$Q = \{Q_1, Q_2, \ldots, Q_n\}$	Set of user request demands
$A = \{a_1, a_2, \ldots, a_p\}$	Set of auxiliary edge servers
$Pos_{a_k} = (X_{a_k}, Y_{a_k})$	Position of auxiliary edge server a_k
$L_A = \{L_{a_1}, L_{a_2}, \ldots, L_{a_m}\}$	Set of auxiliary edge server load levels
$l_{a_k} = \{l_{a_k}^{max}, l_{a_k}^{now}\}$	Max and current loads of auxiliary edge server a_k
R_{a_k}	Communication range of auxiliary edge server a_k
p_{lim}	Max number of auxiliary edge servers
$d = \{d_1, d_2, \ldots, d_n\}$	Delays between users and their request destinations
$D = \{D_1, D_2, \ldots, D_n\}$	Distances between users and their request destinations
L_{aa}	Overload critical load level of edge servers

Definition 1 (Edge Users)**.** *The set of edge users is defined as* $U = \{u_1, u_2, \ldots, u_n\}$*. For an edge user, their geographical position is defined by the coordinates* $u_k = (X_{u_k}, Y_{u_k})$*.*

Definition 2 (User Request)**.** *The set of user request demands is defined as* $Q = \{Q_1, Q_2, \ldots, Q_n\}$*, corresponding to each edge user in the user set* $U = \{u_1, u_2, \ldots, u_n\}$*.*

In the initialization phase of the experiment, the values in set Q need to be assigned to the corresponding users according to their indices. Users are permitted to dispatch these requests to appropriate edge servers or auxiliary edge servers for processing based on the actual conditions of the edge system or, alternatively, process them on their own devices.

Definition 3 (Edge servers)**.** *The set of edge servers is defined as* $S = \{s_1, s_2, \ldots, s_m\}$ *. For the edge server* $s_k = \{Pos_{s_k}, R_{s_k}, l_{s_k}\}$*, its geographical position is defined by the coordinates* $Pos_{s_k} = (X_{s_k}, Y_{s_k})$*. Its communication range is defined as* R_{s_k} *, and the set of its maximum load capacity and current load is defined as* $l_{s_k} = \{l_{s_k}^{max}, l_{s_k}^{now}\}$*.*

For an edge server s_i, which has a communication range R_{s_i}, a maximum load capacity $l_{s_i}^{max}$, and a current load $l_{s_i}^{now}$, a user u_j that can communicate with s_i must satisfy the following conditions: the geospatial distance between u_j and s_i must be no greater than the

communication range, and the remaining load capacity of s_i must be sufficient to meet the current request demand of the user. That is,

$$\sqrt{\left(X_{s_i} - X_{u_j}\right)^2 + \left(Y_{s_i} - Y_{u_j}\right)^2} \leq R_{s_i} \tag{1}$$

$$l_{s_i}^{now} + Q_j \leq l_{s_i}^{max} \tag{2}$$

The main focus of the A–ESD Problem in this paper is to deploy auxiliary edge servers in existing edge systems to improve load balancing. Therefore, it is necessary to define the auxiliary edge servers themselves.

Definition 4 (auxiliary edge servers)**.** *The set of auxiliary edge servers to be deployed is defined as* $A = \{a_1, a_2, \ldots, a_p\}$*. For an auxiliary edge server* $a_k = \{Pos_{a_k}, R_{a_k}, l_{a_k}\}$*, geographical position is defined by the coordinates* $Pos_{a_k} = (X_{a_k}, Y_{a_k})$*. Its communication range is defined as* R_{a_k}*, and the set of its maximum load capacity and current load is defined as* $l_{a_k} = \{l_{a_k}^{max}, l_{a_k}^{now}\}$*.*

In an approach similar to that used for the edge servers, for an auxiliary edge server a_i, which has a communication range R_{a_i}, a maximum load capacity $l_{a_i}^{max}$, and a current load $l_{a_i}^{now}$, a user u_j capable of communicating with a_i must satisfy the following conditions: the geospatial distance between a_i and u_j must be no greater than the communication range, and the residual load capacity of a_i must be sufficient to meet the current request demand of the user. That is,

$$\sqrt{\left(X_{a_i} - X_{u_j}\right)^2 + \left(Y_{a_i} - Y_{u_j}\right)^2} \leq R_{a_i} \tag{3}$$

$$l_{a_i}^{now} + Q_j \leq l_{a_i}^{max} \tag{4}$$

The number of auxiliary edge servers is part of the optimization objective and is not predetermined. We initially posit that the number of auxiliary edge servers is p , which will be a variable to be determined during the process. To ensure the smooth operation of the algorithm, it is prudent to establish a maximum threshold for the number of deployable auxiliary edge servers, defined as p_{lim}. This limit should satisfy the following condition:

$$p \leq p_{\text{lim}} \tag{5}$$

During the algorithm's execution, any unused auxiliary edge servers will be designated as idle, and the set of auxiliary edge servers will be capped at p_{lim} elements. This ensures the stability and efficiency of the algorithm's execution.

When an auxiliary edge server a_k is deployed, its coordinates (X_{a_k}, Y_{a_k}) must not exceed the boundaries of the study area. The lower-left boundary point of the study area is defined as $(0, 0)$. That is,

$$\begin{cases} 0 \leq X_{a_k} \leq X_{max} \\ 0 \leq Y_{a_k} \leq Y_{max} \end{cases} \tag{6}$$

where (X_{max}, Y_{max}) represents the coordinates of the boundary point farthest from the initial point in the study area.

3.2. Load Balancing- and Delay-Aware Model

This paper establishes optimizing load balancing and delay as the objectives of the model.

To effectively address these metrics, it is necessary to quantify their specific performance based on the user-offloading process, thereby deriving concrete values that can measure load balancing and delay.

User requests to the edge system should, whenever feasible, be processed by edge servers. The edge server must fall within the communication range of the user and possess ample idle resources to support request offloading. To achieve better load balancing, it is imperative for users to assess the load levels of edge servers to ascertain whether they are operating beyond capacity, while ensuring that offloading actions contribute positively to the overall load distribution.

User requests are typically routed to the edge server exhibiting the lowest current load level that is within the communication range, thereby enhancing overall load distribution and reducing the incidence of overloading on edge servers. Therefore, it is imperative to precisely delineate the load thresholds of edge servers. Additionally, assessing server-overload status constitutes a critical aspect; hence, establishing unambiguous criteria for overload conditions is essential.

Definition 5 (Load level)**.** *The load levels of edge server s_k and auxiliary edge server a_k are defined as L_{s_k} and L_{a_k}, respectively, representing the percentage of the current load relative to the total load. The critical load level for determining whether an edge server or auxiliary edge server is overloaded is defined as L_{aa}.*

The load levels of the edge server L_{s_k} and the auxiliary edge server L_{a_k} can be expressed as follows:

$$L_{s_k} = \frac{l_{s_k}^{max} - l_{s_k}^{now}}{l_{s_k}^{max}} \times 100\% \tag{7}$$

$$L_{a_k} = \frac{l_{a_k}^{max} - l_{a_k}^{now}}{l_{a_k}^{max}} \times 100\% \tag{8}$$

In this paper, load levels are quantified as percentages to mitigate potential biases in the assessment of actual load levels arising from the heterogeneity of edge servers and auxiliary edge servers.

To reduce the number of auxiliary edge servers deployed, when the resources of the edge system are sufficient, users should refrain from offloading requests to auxiliary edge servers. Only when all edge servers within the user's communication range are overloaded will users consider offloading requests to auxiliary edge servers. Specifically, when the load level of an edge server reaches L_{aa}, that server is deemed overloaded and ceases to accept new requests from users. If, after attempting to find auxiliary edge servers for offloading, a user still cannot find any available devices and if the overloaded edge server still possesses the capacity to process the request, the request will be returned to that server for processing. This is because, in the absence of available edge servers, users must perform local computations, which significantly degrades the user experience. As this could potentially impair overall load distribution, such a situation should be avoided as much as possible.

During the offloading process, this paper utilizes the system's overall standard deviation *Std* and the average distance $\bar{D}$ between users and the devices to which they offload requests to measure system performance metrics, specifically the overall load-balancing level and the overall delay.

Currently, the standard deviation of edge-server loads is frequently employed as an optimization criterion [6,20]. A lower value indicates a more balanced load distribution within the edge system. Therefore, the standard deviation can be expressed as follows:

$$Std = \sqrt{\frac{\sum_{i=1}^{m} (L_{s_i} - \bar{L_s})^2}{m}} \tag{9}$$

where $\bar{L_s}$ represents the average load ratio of all edge servers.

Employing the distance D as a metric for delay is a common approach in the field of edge-server deployment [13,21]. The following will demonstrate the rationality of using the average distance $\bar{D}$ between users and the devices to which they offload requests to measure delay.

The delay between user u_i and the device to which it offloads tasks can be calculated as follows:

$$d_i = \frac{Q_i}{c_i} \tag{10}$$

where c_i denotes the data-transmission rate between user u_i and the edge server s_j or auxiliary edge server a_j to which it offloads requests. For a fixed user u_i, Q_i is a predetermined value and remains constant, so d_i is related only to c_i and they are negatively correlated. The signal-transmission rate is influenced by numerous interference conditions, making it challenging to express this value with a single formula. These interference factors can be collectively considered, and under an ideal assumption, the calculation formula can be expressed as follows:

$$c_i = B \log_2\left(1 + \frac{g_i p_i}{N}\right) \tag{11}$$

where B denotes the signal bandwidth, p_i denotes the transmission power of user u_i, and N denotes the sum of all interferences and noise. In the research scenario of this paper, N can be considered a fixed value. Therefore, for a fixed user u_i, all these variables remain constant, and it can be concluded that c_i is related only to g_i and that they are positively correlated. g_i denotes the channel gain between user u_i and the device to which it offloads tasks. According to the 3GPP technical report, the calculation formula can be expressed as follows:

$$g_i = \frac{g_0}{(D_i)^2} \tag{12}$$

where g_0 denotes the reference channel gain at a distance of 1 m. For a fixed user u_i, g_0 remains constant throughout the experiment. Therefore, h_i is solely dependent on the distance D_i between user u_i and the device to which it offloads requests, and they are negatively correlated. Combining the derivations from Equations (10) and (11), it can be concluded that the delay d_i is solely a function of the distance D_i and that they are positively correlated. Therefore, this distance can be used as a standard for measuring the delay of a single user.

The overall system delay can be obtained by averaging the delays of all users. The average distance $\bar{D}$ between users and the devices to which they offload requests can be expressed as follows:

$$\bar{D} = \frac{\sum_{i=1}^{n} D_i}{n} \tag{13}$$

3.3. A–ESD Problem

Definition 6 (Auxiliary Edge-Server Deployment Problem)**.** *Based on the derivations from the preceding sections, the A–ESD problem to be studied in this paper can be defined as a quadruple* $\langle U, Q, S, A \rangle$ *where*

$U = \{u_1, u_2, \ldots, u_n\}$ *is the set of users.*
$Q = \{Q_1, Q_2, \ldots, Q_n\}$ *is the set of user request demands*
$S = \{s_1, s_2, \ldots, s_m\}$ *is the set of edge servers.*
$A = \{a_1, a_2, \ldots, a_p\}$ *is the set of auxiliary edge servers.*

4. Approach

4.1. Optimization Model

Integrating the insights gleaned in the preceding sections, the optimization model for the A–ESD problem in this paper can be formulated as follows:

$$min\sqrt{\frac{\sum_{i=1}^{m}(L_{s_i}-\bar{L_s})^2}{m}} \tag{14}$$

$$min\frac{\sum_{i=1}^{n}D_i}{n} \tag{15}$$

s.t.

$$p \le p_{\lim} \tag{16}$$

$$\begin{cases} 0 \le X_{a_i} \le X_{max} \\ 0 \le Y_{a_i} \le Y_{max} \end{cases} a_i \in A \tag{17}$$

$$\begin{cases} \sqrt{\left(X_{s_i}-X_{u_j}\right)^2+\left(Y_{s_i}-Y_{u_j}\right)^2} \le R_{s_i} u_j \in U_{s_i} s_i \in S \\ \sqrt{\left(X_{a_i}-X_{u_j}\right)^2+\left(Y_{a_i}-Y_{u_j}\right)^2} \le R_{a_i} u_j \in U_{a_i} a_i \in A \end{cases} \tag{18}$$

$$\begin{cases} \sum_{u_j \in U_{s_i}} Q_j \le l_{s_i}^{max} s_i \in S \\ \sum_{u_j \in U_{a_i}} Q_j \le l_{a_i}^{max} a_i \in A \end{cases} \tag{19}$$

where U_{s_i} represents the set of users associated with the edge server s_i and U_{a_i} represents the set of users associated with the auxiliary edge server a_i.

Function (14) is the load-balancing optimization objective, which aims to minimize the overall standard deviation of the edge-server loads. Function (15) is the delay optimization objective, which aims to minimize the average distance between users and the devices to which they offload tasks. Function (16) ensures that the number of deployed auxiliary edge servers does not surpass the predefined maximum. Function (17) ensures that the coordinates of the deployed auxiliary edge servers fall within the predefined boundaries. Function (18) ensures that the distance between users and their offloading devices is within the communication range. Function (19) ensures that users are prohibited from offloading tasks to devices that have insufficient remaining capacity.

4.2. A–ESD Problem Hardness

The Auxiliary Edge Server Deployment (A–ESD) problem is NP-hard. Proof. We establish this by a reduction from the classical capacitated k-median problem, which is well known to be NP-hard [13]. The capacitated k-median problem is defined on a set of clients $U = u_1, \ldots, u_n$, a set of candidate facilities $F = f_1, \ldots, f_m$, capacity limits C_f for each facility $f \in F$, and a distance metric $d(\cdot, \cdot)$. The goal is to select at most k facilities such that all clients are assigned to an open facility without violating capacity constraints while minimizing the total assignment cost $\sum_{u \in U} d(u, f(u))$. To reduce this problem to A–ESD, we let the candidate auxiliary edge servers A correspond directly to the facilities F, while the existing edge servers S are treated as fixed facilities with given residual capacities. The user set U remains unchanged, with identical demand quantities Q_j, and the communication ranges are set to be sufficiently large that feasibility is governed solely by the capacity constraints. Under this construction, any feasible solution of the capacitated k-median problem with cost at most B yields an auxiliary deployment in A–ESD with no more than $p = k$ nodes that satisfies constraints (16)–(19) and achieves cost at most B; conversely, any feasible A–ESD solution can be mapped back to a capacitated k-median solution with equivalent feasibility and objective value. Since this reduction is clearly computable in

polynomial time, the NP-hardness of the capacitated *k*-median problem directly implies that A–ESD is also NP-hard.

4.3. LBA–GA Method

4.3.1. Overview of LBA–GA Algorithm

The LBA–GA algorithm is an enhanced approach based on the genetic algorithm. Genetic algorithms generate a random population for the problem and then encode this population, resulting in encoded segments similar to chromosomes in biology. By manipulating these encoded segments and simulating the evolutionary processes guided by a fitness function, the population is continuously optimized to obtain an approximate optimal solution.

Traditional genetic algorithms exhibit limitations when applied to the A–ESD problem, such as inappropriate configurations of the fitness function and insufficient responsiveness to specific problem conditions. Hence, there is a compelling need to refine the genetic algorithm to better address the A–ESD problem.

The pseudocode for the enhanced genetic algorithm, LBA–GA, is shown in Algorithm 1. Firstly, the initial parameters are set; these include population size N_{pop}, number of iterations Nit, crossover probability Pcr, and mutation probability Pmu (1), Then, a random population of the specified size $Npop$ is generated as the initial population (2). Next, a fitness calculation is performed to obtain the necessary data and fitness values (3). Before the maximum number of iterations is reached, each loop cycle involves selection (7), crossover (8–10, 16–18), and mutation (11–15) operations. At the end of each iteration, a fitness calculation (20) and selection-coefficient calculation (21) are performed. Based on the fitness and selection coefficients, the probability of each individual in the population entering the next generation is calculated (22). The next generation's population is then formed based on these probabilities and proceeds to the next iteration. Once the maximum number of iterations has been reached, the individual with the highest fitness in the current population is returned, and the approximate solution is obtained by decoding this individual (23–25).

4.3.2. Generation Update of LBA–GA

During the initialization of the population, each individual is regarded as a set of positions for auxiliary edge servers, constructed through the concatenation of the positions of each node. This encompasses all pertinent details regarding the deployment positions of the auxiliary edge servers, which are subsequently encoded into a binary chromosome sequence.

In the fitness-calculation step, to obtain the required variables, auxiliary edge servers should be deployed in the edge system according to the chromosome code. A single offloading process is then performed, and data for various system variables are recorded. Necessary data are extracted for subsequent calculations, and the data from the offloading process are used to supplement and save the attributes of the auxiliary edge servers. In this paper, the load-balancing level serves as the fitness metric and is represented as the reciprocal of the standard deviation of the load levels of all edge servers. The formula can be expressed as follows:

$$f_{lb}(H_i) = \frac{\sqrt{m \sum_{j=1}^{m} \left(L_{s_j} - \bar{L_s} \right)^2}}{\sum_{j=1}^{m} \left(L_{s_j} - \bar{L_s} \right)^2} \tag{20}$$

Algorithm 1: Load balancing-based auxiliary edge-server deployment genetic algorithm (LBA–GA)

Input: An A–ESD problem $\langle U, Q, S, A \rangle$
Output: The approximate solution $Pos = \{Pos_{a_1}, Pos_{a_2}, \ldots, Pos_{a_p}\}$

Set parameters $N_{\text{pop}}, N_{\text{it}}, P_{\text{cr}}, P_{\text{mu}}$;
Initialize population $H = \{H_1, H_2, \ldots, H_{N_{\text{pop}}}\}$ randomly ;
Compute fitness:
$f_{\text{lb}}(H_i) = \frac{\sqrt{m \sum_{j=1}^{m} (L_{s_j} - \bar{L}_s)^2}}{\sum_{j=1}^{m} (L_{s_j} - \bar{L}_s)^2}$;
for $i = 1$ **to** N_{it} **do**
    Initialize H' as next-generation set ;
    for $j = 1$ **to** N_{pop} **do**
        Select individuals H_j and H_k randomly ;
        if $\text{rand}(0,1) \leq P_{\text{cr}}$ **then**
            Perform crossover: $(H_j, H_k) \rightarrow H_{\text{cr}}$;
            Compute mutation coefficient C_{mu} ;
            if $\text{rand}(0,1) \leq P_{\text{mu}} \cdot C_{\text{mu}}$ **then**
                Perform mutation: $H_{\text{cr}} \rightarrow H'_j$;
            else
                $H'_j \leftarrow H_{\text{cr}}$;
        else
            $H'_j \leftarrow H_j$;
    Compute fitness of new population using $f_{\text{lb}}(H'_i)$;
    Compute selection coefficient C_{sel} ;
    Compute selection probability:
    $P_{\text{sel},i} = \frac{f_{\text{lb}}(H'_i) \cdot C_{\text{sel},i}}{\sum_{j=1}^{N_{\text{pop}}} f_{\text{lb}}(H'_j) \cdot C_{\text{sel},j}}$;
    Generate next generation H based on $P_{\text{sel},i}$;
Select best solution $H_{\text{best}} \leftarrow \arg\max_{H_i \in H} f_{\text{lb}}(H_i)$;
Decode and return solution:
$Pos = \{Pos_{a_1}, Pos_{a_2}, \ldots, Pos_{a_p}\}$ from H_{best} ;

To enhance the convergence velocity of the algorithm, in the crossover process, we forgo the conventional random-crossover approach employed in genetic algorithms; instead, the chromosome code is segmented based on the chromosome code corresponding to each auxiliary edge server. The individuals H_j and H_k are decomposed into $H_j = \{H_{j,1}, H_{j,2}, \ldots, H_{j,p}\}$ and $H_k = \{H_{k,1}, H_{k,2}, \ldots, H_{k,p}\}$, as depicted in Figure 2. For each corresponding pair of segments $H_{j,l}$ and $H_{k,l}$, which correspond to auxiliary edge servers $a_{j,l}$ and $a_{k,l}$, the segment exhibiting a lower total delay for the user sets $U_{a_{j,l}}$ and $U_{a_{k,l}}$ is selected as the chromosome code segment post-crossover. That is,

$$H_{j,l}^{cr} = \begin{cases} H_{j,l}, & \sum D_{U_{a_{j,l}}} < \sum D_{U_{a_{k,l}}} \\ H_{k,l}, & \sum D_{U_{a_{j,l}}} \geq \sum D_{U_{a_{k,l}}} \end{cases} \tag{21}$$

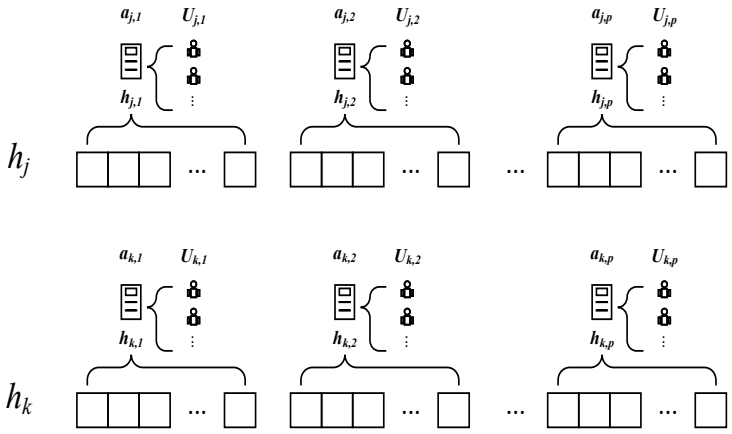


Figure 2. Chromosome coding segmentation.

This ensures that the resulting offspring possess the superior-performing segment in terms of total delay.

When the crossover process is enhanced with a focus on delay optimization, the delay performance of the offspring chromosome code is optimized and thereby rapidly converges to the optimal solution. By contrast, load balancing, which must be assessed from a global perspective, is challenging to incorporate within local adjustments. To prevent the algorithm from being excessively swayed by delay considerations, which could potentially compromise the optimization of load balancing, an independent traditional crossover probability $P_{cr}^{'}$ is employed in the crossover operation. The chromosome code of the offspring individual $H^{'}$ after crossover is defined as follows:

$$H^{'} = \begin{cases} H_{cr}, & 0 \leq rand(0,1) \leq P_{cr} \\ H_{cr}^{'}, & P_{cr} < rand \leq P_{cr}^{'} \end{cases} \tag{22}$$

where H_{cr} denotes the offspring's individual chromosome code obtained using the improved crossover method proposed in LBA–GA, and $H_{cr}^{'}$ denotes the offspring's individual chromosome code obtained using the traditional crossover method. When a random number meets the criteria for the traditional crossover probability yet fails to meet those for the improved crossover probability, solely the traditional crossover method is applied. This approach accelerates convergence while avoiding the risk of missing the optimal solution.

To broaden the search space and prevent premature convergence, the mutation process is further refined. After the chromosome code based on auxiliary edge servers has been segmented as $H_j^{\mathrm{cr}} = H_{j,1}^{\mathrm{cr}}, H_{j,2}^{\mathrm{cr}}, \ldots, H_{j,p}^{\mathrm{cr}}$, a distinct mutation operation is applied to each segment $H_{j,k}^{\mathrm{cr}}$ within the group. This entails randomly inverting a bit within the binary representation to derive $H_{j,k}^{'}$. The resulting mutated segments are subsequently recombined to constitute $H^{'}$ as a candidate for the subsequent generation.

However, this approach markedly increases the number of bits that can potentially be mutated and the probability of mutation, thereby potentially resulting in an excessively high mutation rate and thus impeding algorithmic convergence. To address this, a mutation-adjustment coefficient C_{mu} is introduced. This coefficient is derived from the aggregate standard deviation of each solution set, as follows:

$$C_{mu} = C_{mu,0}\sqrt{\frac{\sum_{i=1}^{m}(L_{s_i} - \bar{L}_s)^2}{m}} \tag{23}$$

where $C_{mu,0}$ denotes the baseline mutation coefficient, a value established a priori. Adjusting the mutation probability ensures that the likelihood of mutation for higher-performing solutions is proportionally diminished. This approach guarantees that the algorithm, in its pursuit of the optimal solution, sustains stability in convergence throughout the execution process.

To account for the effects of latency and the count of auxiliary edge servers within the algorithm's deployment, a set of coefficients C_{sel} is defined in the selection step of the genetic algorithm. These coefficients are derived from the average latency across all users within the system and the tally of auxiliary edge servers. The calculation formula is as follows:

$$C_{sel} = C_{sel,0}\frac{n}{p\sum_{i=1}^{n} D_i} \tag{24}$$

where $C_{sel,0}$ denotes the selection-balancing coefficient, which serves to correlate the number of auxiliary edge servers and the average user latency into the same space. It is calculated as follows:

$$C_{sel,0} = p_{max}D_{max} \tag{25}$$

where p_{max} denotes the maximum number of auxiliary edge servers among all individuals in the population and D_{max} denotes the maximum latency among all individuals in the population.

C_{sel} serves as the novel weighting factor in the computation with the fitness set and is subsequently normalized to determine the selection probability $P_{sel,i}$ for each individual:

$$P_{sel,i} = \frac{f_{lb}\left(H_i'\right)C_{sel,i}}{\sum_{j=1}^{N_{pop}} f_{lb}\left(H_j'\right)C_{sel,j}} \tag{26}$$

Through the modulation of selection probabilities, the influence of delay and the number of auxiliary edge servers deployed are manifested in the subsequent generation of the population.

Upon the conclusion of all iterative processes, the chromosome of the fittest individual in the final population is decoded to yield the auxiliary edge server positions, which constitute the determined solution.

5. Experiments

5.1. Experimental Setup and Dataset

In order to verify the performance of LBA–GA, we evaluate its effectiveness and efficiency experimentally. All experiments are conducted on a computer equipped with an AMD Ryzen 7 6800H CPU, 16 GB RAM, and an NVIDIA GeForce RTX 3060 Laptop GPU. Our approach, as well as the comparative methods, were implemented in Python 3.9.

The experiments are carried out on a public benchmarking dataset, the EUA dataset. It contains 125 edge servers in the Melbourne CBD area in Australia, as shown in Figure 3. The dataset records the latitude and longitude of edge servers and users. It has been used extensively in existing studies. In alignment with the research scenarios for the A–ESD problem, user task data are synthesized following the Zipf distribution, and the communication radius and maximum load capacity of edge servers are defined in the experiment.

To explore the effectiveness of the method in different situations, this paper employs two distinct experimental configurations based on an existing dataset, depicting scenarios

of absolute and relative scarcity of edge system resources. and adjusts the number of auxiliary edge servers deployable to reflect real-world constraints.

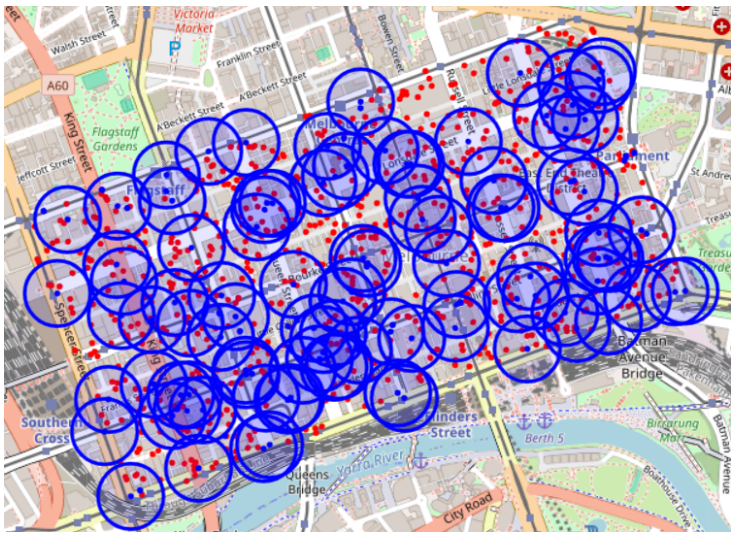


Figure 3. EUA dataset visualization.

5.2. Comparative Methods and Evaluation Metrics

To evaluate the performance of LBA–GA, we compare it with four competing approaches.

- *No-Aux:* It does not use additional auxiliary edge servers.
- *Random-B:* It randomly deploys a number of auxiliary edge servers and selects the servers with optimal performance.
- *Trad-GA:* It uses traditional genetic algorithms to solve the optimization problem.
- *HE-GA* [13]: It employs a genetic algorithm to find the optimal solution for the deployment of edge servers by calculating the impact on communication.

In the experiments, we employ five evaluation metrics to compare and analyze the results.

- *Satisfiability (Sat):* It represents a binary indicator used to ascertain whether the task requirements of all users are satisfied.

$$Sat = \begin{cases} 1, & \sum_{j=1}^{n} Q_j^{'} = 0 \\ 0, & \sum_{j=1}^{n} Q_j^{'} \neq 0 \end{cases} \tag{27}$$

  where $Q_j^{'}$ represents the unprocessed task demand of user u_j.
- *Load-Balancing Optimization Rate (Ra_{Std}):* It is measured by the standard deviation reduction ratio to indicate the effectiveness of the method in terms of load balancing.

$$Ra_{Std} = \frac{Std_{NA} - Std_M}{Std_{NA}} \times 100\% \tag{28}$$

  where Std_{NA} represents the overall standard deviation of the No-Aux method and Std_M represents the overall standard deviation of the current method.
- *Delay-Optimization Rate ($Ra_{\bar{D}}$):* It is measured by the reduction ratio of the mean distance to indicate the effectiveness of the method in terms of delay.

$$Ra_{\bar{D}} = \frac{\bar{D}_{NA} - \bar{D}_M}{\bar{D}_{NA}} \times 100\% \tag{29}$$

where $\bar{D}_{NA}$ represents the overall average distance of the No-Aux method and $\bar{D}_M$ represents the overall average distance of the current method.

- *Load-Balancing Optimization Rate per Node (Ra^{per}_{Std}):* It is measured by the reduction ratio of the standard deviation per node to indicate the cost-effectiveness of deploying auxiliary edge servers for load-balancing optimization.

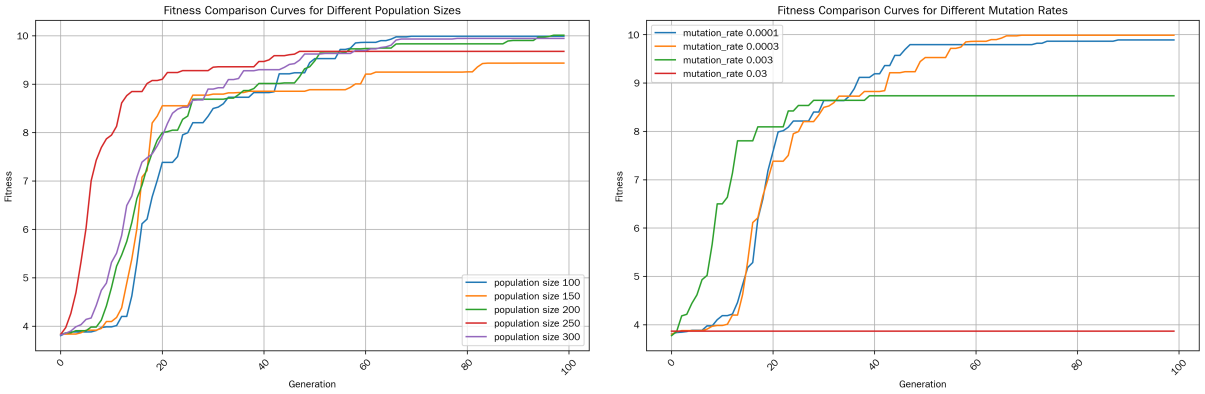
$$Ra^{per}_{Std} = \frac{Ra_{Std}}{p} \tag{30}$$

- *Delay optimization rate per node ($Ra^{per}_{\bar{D}}$):* It is measured by the reduction ratio of the mean distance per node to indicate the cost-effectiveness of deploying auxiliary edge servers for delay optimization.

$$Ra^{per}_{\bar{D}} = \frac{Ra_{\bar{D}}}{p} \tag{31}$$

5.3. Experiment Results and Analyses

In the experiment, the parameters are configured to achieve the optimal performance for the competing methods. The user task demand set Q was simulated based on Zipf's law. The initial parameters were set as follows: $N_{pop} = 100$, $N_{it} = 100$, $P_{cr} = 0.7$, $P^{'}_{cr} = 0.8$, $P_{mu} = 0.0003$ and $p_{lim} = 40$. The selection criteria for the initial population size and mutation rates are as shown in Figure 4a,b. We conducted 50 rounds of simulations on Dataset 1 and Dataset 2, respectively. Then, the average value was taken as the final experimental result.



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

Figure 4. (**a**) Fitness-comparison curves for different population sizes. (**b**) Mutation rates.

5.4. Performance Impact of Parameters

Table 2 summarizes the experimental results on EUA Dataset 1 and Dataset 2 with Laa = 0.8. The No-Aux method fails to satisfy all user demands. Random-B and HE–GA utilize auxiliary servers but achieve only limited improvement due to the lack of a global optimization mechanism. Trad–GA performs better in allocating tasks but shows weaker cost-effectiveness in per-node metrics. In contrast, LBA–GA consistently achieves the best results across all five metrics. Specifically, compared to Trad–GA, LBA–GA improves Ra_{Std} by 2.19% and 3.22% and $Ra_{\bar{D}}$ by 4.08% and 12.05% on the two datasets, respectively. Furthermore, it achieves higher Ra^{per}_{Std} and $Ra_{\bar{D}}$ demonstrating superior cost-effectiveness. These results confirm that LBA–GA outperforms baselines both in overall performance and in efficiency when deployment of auxiliary servers is limited.

Table 2. Experimental data sheet for EUA Dataset 1 and EUA Dataset 2 for the case where $L_{aa} = 0.8$.

Methods	EUA Dataset 1					EUA Dataset 2				
	Sat	Ra_{Std}	$Ra_{\bar{D}}$	Ra_{Std}^{per}	$Ra_{\bar{D}}^{per}$	Sat	Ra_{Std}	$Ra_{\bar{D}}$	Ra_{Std}^{per}	$Ra_{\bar{D}}^{per}$
No-Aux	0	0	0	0	0	0	0	0	0	0
Random-B	0	29.58%	6.68%	1.56%	0.35%	0	28.59%	8.89%	1.51%	0.47%
HE-GA	0	24.64%	6.05%	1.45%	0.36%	0	33.46%	7.21%	2.39%	0.52%
Trad-GA	**1**	54.68%	8.82%	2.05%	0.33%	**1**	**62.48%**	9.46%	3.29%	0.49%
LBA-GA	**1**	**55.88%**	**9.18%**	**2.31%**	**0.38%**	**1**	**62.48%**	**15.06%**	**3.68%**	**0.89%**

Figure 5 illustrates the load distribution of servers in EUA Dataset 1 with $L_{aa} = 0.7$. Under the No-Aux setting, the system shows poor load balance, with several servers overloaded while others remain underutilized. Introducing auxiliary servers alleviates the imbalance to varying degrees. Random-B and HE–GA provide only limited improvement, while Trad–GA significantly reduces the number of overloaded servers. LBA–GA delivers the best results, achieving both lower peak loads and more uniform distribution across servers. This visualization confirms the superior capability of LBA–GA in balancing loads under complex system conditions.

The iteration processes of the LBA–GA and Trad–GA methods are depicted in Figure 6. It compares the convergence curves of Trad–GA and LBA–GA. Trad–GA converges slowly and is prone to local optima in the early iterations. In contrast, LBA–GA, enhanced by improved crossover, mutation, and selection strategies, achieves faster convergence and higher solution quality within fewer iterations. It also maintains stable performance without premature convergence, highlighting its effectiveness in efficiently solving the A–ESD problem.

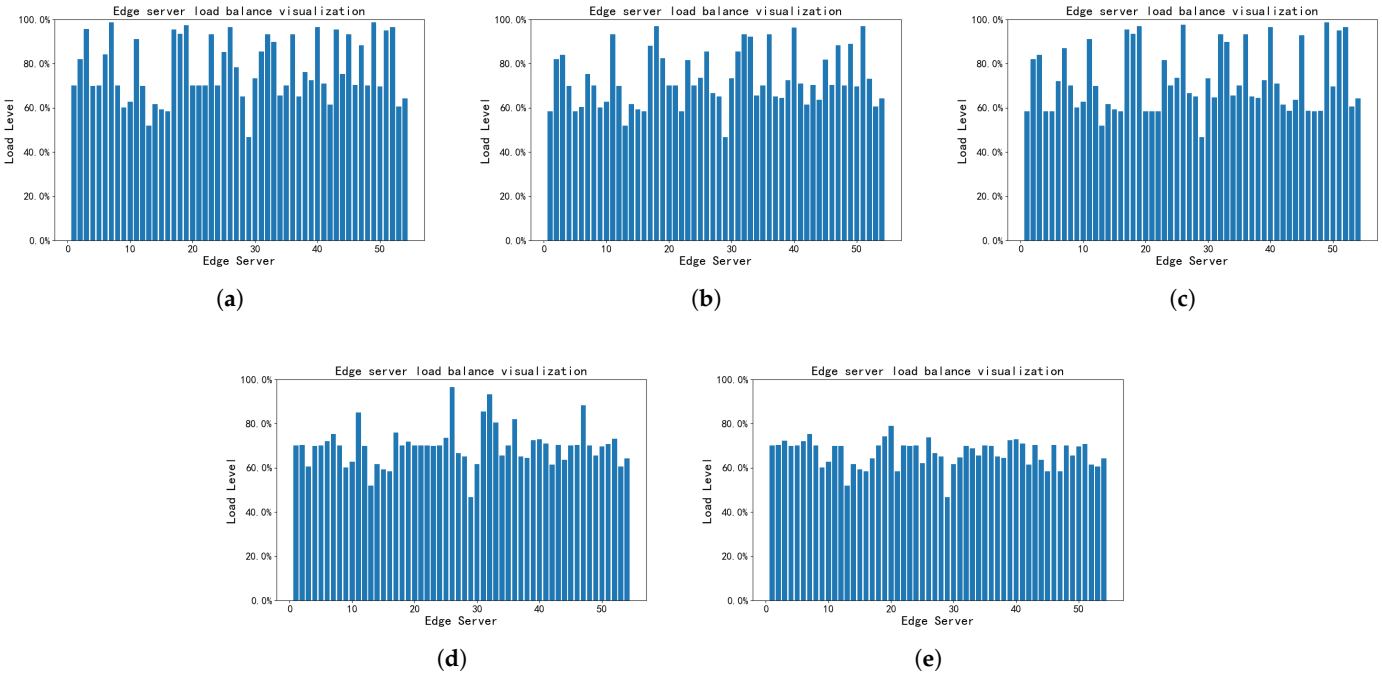


Figure 5. Visualization of edge-server load balancing by different methods. (**a**) No-Aux; (**b**) Random-B; (**c**) HE–GA; (**d**) Trad–GA; (**e**) LBA–GA.

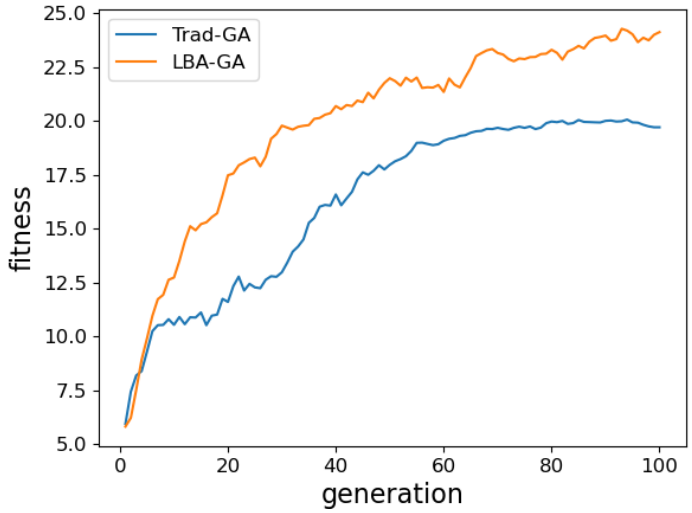


Figure 6. Iteration process of the genetic algorithm.

5.5. Performance Impacts of Parameters

In addition, to assess the method's effectiveness and performance trends across varying conditions, several experiments were also conducted for various L_{aa} values in the same edge environment, and the results are depicted in Figures 7 and 8.

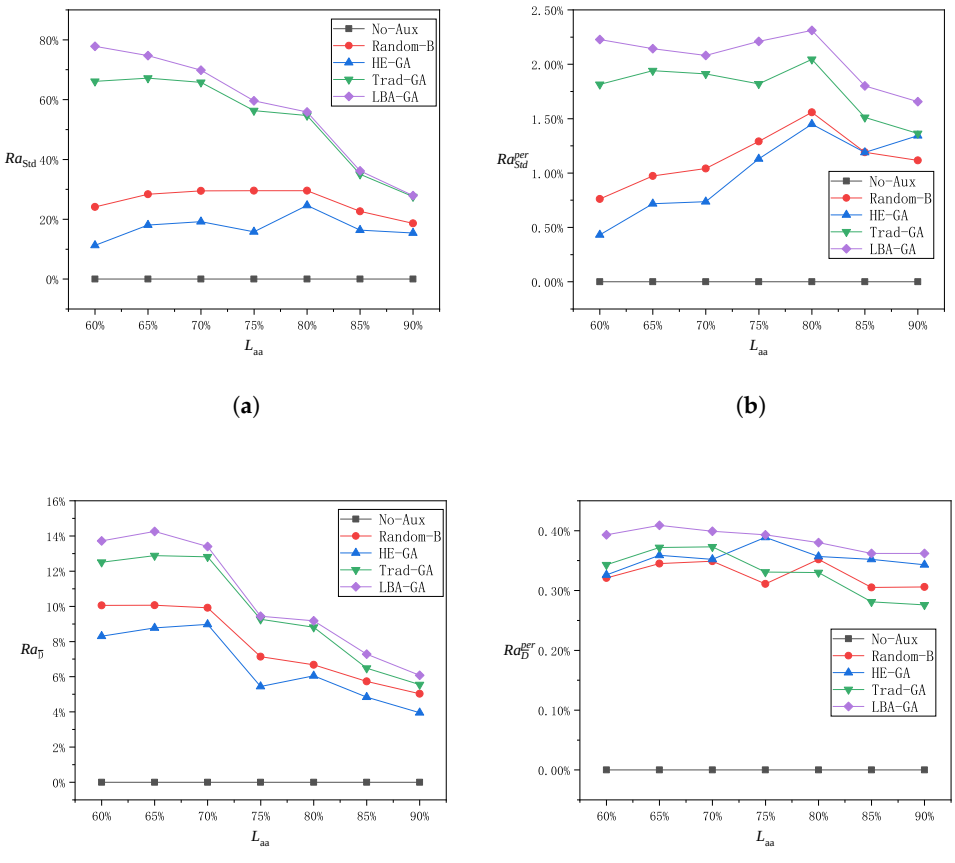


Figure 7. Performance of each method on Dataset 1. (**a**) Load-balancing optimization; (**b**) load-balancing optimization per node; (**c**) delay optimization; (**d**) delay optimization per node.

Figures 7a and 8a display the overall optimization level of each method in terms of load-balancing optimization. For Dataset 1, as L_{aa} increases, the need for auxiliary edge

servers also increases and the algorithm's operational flexibility decreases, leading to a gradual decline in the overall load-balancing optimization effect. For Dataset 2, since the overall system resources are more sufficient, a small number of auxiliary edge servers can be used to search for the optimal solution, and it is not until L_{aa} reaches a certain threshold that the disadvantage of reduced operational space becomes apparent. For the Random and HE–GA methods, the absence of global load-balancing considerations during their operation may prevent the algorithms from finding the optimal solution, resulting in inferior outcomes and irregular variations in the standard deviation of the algorithms as L_{aa} changes. The LBA–GA method generally shows greater improvement in load-balancing optimization than Trad–GA across different L_{aa} values, while Trad–GA has more significant advantages over the other methods.

The per-node optimization level of each method in terms of load-balancing optimization is shown in Figures 7b and 8b. Although the effect of the overall load-balancing optimization decreases as the value of L_{aa} increases, the per-node optimization level in terms of standard deviation for each method tends to remain stable, as this also concurrently reduces the number of deployed auxiliary edge servers. The LBA–GA method exhibits a more pronounced advantage over the other methods; therefore, it is concluded that the LBA–GA method is superior for both overall load-balancing optimization and the cost-effectiveness of load-balancing optimization.

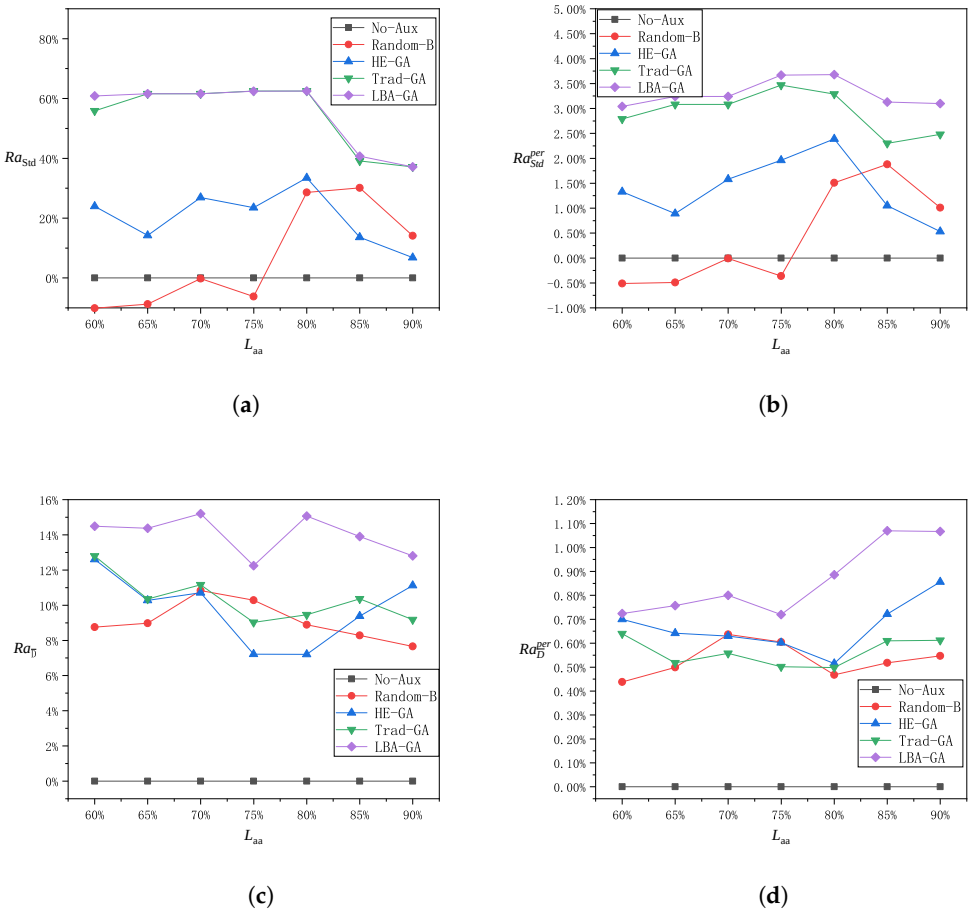


Figure 8. Performance of each method on Dataset 2. (**a**) Load-balancing optimization; (**b**) load-balancing optimization per node; (**c**) delay optimization; (**d**) delay optimization per node.

The overall optimization level of each method in terms of delay is demonstrated in Figures 7c and 8c. For Dataset 1, as L_{aa} increases, the need for auxiliary edge servers increases and the operational space for the algorithm is reduced, resulting in a gradual

reduction in the overall optimization of the delay. In contrast, in Dataset 2, the overall delay is less sensitive to changes in L_{aa}. The LBA–GA method generally outperforms the other methods in terms of average improvement across different L_{aa} values and holds a more significant advantage.

The per-node optimization level of each method in terms of delay, as presented in Figures 7d and 8d, indicates that for Dataset 1, although the overall standard deviation effect diminishes as L_{aa} increases, the per-node optimization level in terms of standard deviation for each method tends to remain stable, as this approach also concurrently reduces the number of auxiliary edge servers deployed. For Dataset 2, although the overall delay changes little, the per-node improvement becomes greater as the number of auxiliary edge servers decreases at higher L_{aa} values. When the methods are compared, the performance difference is not significant, and LBA–GA is concluded to be the best-performing method across all scenarios.

Overall, the LBA–GA method demonstrates significant advantages in load-balancing optimization, both in terms of overall improvement and cost-effectiveness, compared to other methods. Additionally, it sustains superior performance in terms of latency while ensuring load-balancing optimization. Therefore, it can be concluded that the LBA–GA method is effective and superior in addressing the A–ESD problem proposed in this paper.

6. Related Work

During the deployment of edge servers, a multitude of factors must be taken into account. Among these, geographical location is of paramount importance due to the high sensitivity of edge servers to distance. The performance of the edge system can exhibit substantial variation across different environments, and the objectives of research on edge-server deployment can differ significantly as well. Therefore, identifying effective deployment strategies for edge servers and the selection of appropriate specifications to achieve specific goals are crucial research goals [22,23].

To optimize edge-server deployment, a variety of multi-objective and heuristic strategies have been proposed in recent studies. Ye et al. [13] proposed a scheme for edge-server deployment that combines multi-objective optimization methods and genetic algorithms. By iteratively determining the number of servers through probabilistic modeling and then optimizing their locations, they achieved load balancing while also minimizing delay and energy consumption. Lovén et al. [16] employed a method for edge-server deployment that considers a large number of constraints and parameters to expand edge servers in a system that already contains deployed edge servers, achieving load balancing and demonstrating stability [24]. Cao et al. [25] introduced a clustering method where the clustering radius varies with density, along with an improved multi-objective optimization technique, and presented a six-objective optimization framework for large-scale vehicular networks. Given the complexity of the edge-server deployment problem, it is generally modeled as a multi-objective optimization problem [21], which can be effectively addressed using evolutionary or heuristic algorithms. Chang et al. [26] analyzed vehicle trajectories for urban vehicular services and proposed a heuristic multi-objective optimization method for roadside edge-server deployment. Zhang et al. [27] developed an integrated approach using clustering algorithms and nonlinear programming to jointly optimize the deployment of both edge servers and services. Ling et al. [28] employed a graph convolutional network-based traffic prediction model to inform edge-server placement, thereby reducing overall deployment cost and the frequency of server overload. To enhance robustness, Cui et al. [29] formulated a k-edge server-placement problem considering possible server failures and solved it using an integer programming-based optimization approach. Chen et al. [30] proposed a preference-aware placement method that accounts for user query

behavior across different regions, demonstrating improved performance on large-scale datasets. Jasim et al. [9] applied edge-server deployment to the healthcare domain, developing an optimal placement algorithm that enhances service efficiency, cost-effectiveness, and response latency. Liu et al. [31] addressed the security challenges in Web 3.0 vehicular networks by proposing a non-cooperative game-theoretic placement scheme integrated with anomaly-detection mechanisms to enhance deployment security.

Currently, significant progress has been made in the research on edge-server deployment, yet certain limitations persist. Existing approaches rarely consider the incremental deployment of new servers into an already existing system under overload conditions. In contrast, our work explicitly addresses this gap by formulating the Auxiliary Edge Server Deployment (A–ESD) problem, which emphasizes load balancing and incremental scalability.

7. Conclusions and Future Work

This paper studies a server-deployment scheme for the auxiliary edge-server deployment problem (A–ESD Problem) and includes load-balancing optimization in the optimal objective. By deploying lightweight auxiliary edge servers, it addresses the severe resource-shortage problem in edge systems. First, the paper formalizes the A–ESD Problem, models it as a constrained optimization problem, and proves its NP-hardness. To effectively solve the A–ESD Problem, this paper presents an improved method based on genetic algorithms, called LBA–GA, which searches for the optimal deployment positions of auxiliary edge servers to achieve the best possible load balance. Extensive experiments were conducted on real-world datasets to evaluate the performance of LBA–GA. The experimental results show that the LBA–GA method demonstrates significant advantages over the compared methods across multiple metrics.

In future work, we plan to introduce the time-varying characteristics of user locations and request demands. We will conduct research on server expansion in a time-varying edge environment to explore strategies for dynamic edge-server deployment. We also intend to investigate the runtime complexity and convergence of the proposed algorithm and to design more efficient variants to improve scalability and practical applicability.

Author Contributions: Conceptualization, S.N. and B.Z.; methodology, S.N. and K.L.; software, X.Z. and S.W.; validation, X.Z., S.W. and K.L.; formal analysis, G.Z.; investigation, S.N.; resources, B.Z.; data curation, G.Z.; writing—original draft preparation, X.Z.; writing—review and editing, X.Z. and S.W.; visualization, K.L.; supervision, G.Z. and B.Z.; project administration, S.N.; funding acquisition, G.Z. and B.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by Natural Science Foundation of Xinjiang Uygur Autonomous Region under Grant No. 2022D01A236, National Natural Science Foundation of China under Grants 62272290, Shanghai Central Guide Local Science and Technology Development Fund Projects under Grant No. YDZX20253100004004005.

Data Availability Statement: The original contributions presented in the study are included in the article; further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to thank Qiang He for the contributors of the EUA dataset, which were essential for model validation. We also appreciate all of the anonymous reviewers for their insightful suggestions and useful comments that will significantly improve the quality of our manuscript.

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

References

1. Reinsel, D.; Gantz, J.; Rydning, J. Data Age 2025: The Evolution of Data to Life-Critical. Report, 2017. Available online: https://www.seagate.com/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf (accessed on 12 April 2022).
2. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
3. Kong, L.; Tan, J.; Huang, J.; Chen, G.; Wang, S.; Jin, X.; Zeng, P.; Khan, M.; Das, S.K. Edge-computing-driven internet of things: A survey. *ACM Comput. Surv.* **2022**, *55*, 1–41. [CrossRef]
4. Chinnasamy, P.; Rojaramani, D.; Praveena, V.; Annlin Jeba, S.; Bensujin, B. Data Security and Privacy Requirements in Edge Computing: A Systemic Review. *Cases Edge Comput. Anal.* **2021**, 171–187.
5. Wang, Z.; Zhou, Y.; Jin, X.; Chen, Y.; Lu, C. An edge server deployment approach for delay reduction and reliability enhancement in the industrial internet. *Wirel. Netw.* **2024**, *30*, 5743–5757. [CrossRef]
6. Ghasemzadeh, A.; Aghdasi, H.S.; Saeedvand, S. Edge server placement and allocation optimization: A tradeoff for enhanced performance. *Clust. Comput.* **2024**, *27*, 5783–5797. [CrossRef]
7. Li, Q.; Wang, S.; Zhou, A.; Ma, X.; Yang, F.; Liu, A.X. QoS driven task offloading with statistical guarantee in mobile edge computing. *IEEE Trans. Mob. Comput.* **2020**, *21*, 278–290. [CrossRef]
8. Mondal, S.; Ruffini, M. Optical front/mid-haul with open access-edge server deployment framework for sliced O-RAN. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 3202–3219. [CrossRef]
9. Jasim, A.M.; Al-Raweshidy, H. Optimal intelligent edge-servers placement in the healthcare field. *IET Netw.* **2024**, *13*, 13–27. [CrossRef]
10. Yin, L.; Sun, J.; Zhou, J.; Gu, Z.; Li, K. ECFA: An Efficient Convergent Firefly Algorithm for Solving Task Scheduling Problems in Cloud-Edge Computing. *IEEE Trans. Serv. Comput.* **2023**, *16*, 3280–3293. [CrossRef]
11. Asghari, A.; Sayadi, M.; Azgomi, H. Energy-aware edge server placement using the improved butterfly optimization algorithm. *J. Supercomput.* **2023**, *79*, 14954–14980. [CrossRef]
12. Havas, S.; Azizi, S.; Abdollahpouri, A. A Multistart Power of d Choices Strategy for Edge Server Placement Problem. In Proceedings of the 2023 7th International Conference on Internet of Things and Applications (IoT), Xining, China, 25–27 August 2023; pp. 1–6.
13. Ye, H.; Cao, B.; Liu, J.; Li, P.; Tang, B.; Peng, Z. An edge server deployment method based on optimal benefit and genetic algorithm. *J. Cloud Comput.* **2023**, *12*, 148. [CrossRef]
14. Cui, G.; He, Q.; Chen, F.; Jin, H.; Yang, Y. Trading off between user coverage and network robustness for edge server placement. *IEEE Trans. Cloud Comput.* **2020**, *10*, 2178–2189. [CrossRef]
15. Song, H.; Gu, B.; Son, K.; Choi, W. Joint optimization of edge computing server deployment and user offloading associations in wireless edge network via a genetic algorithm. *IEEE Trans. Netw. Sci. Eng.* **2022**, *9*, 2535–2548. [CrossRef]
16. Lovén, L.; Lähderanta, T.; Ruha, L.; Leppänen, T.; Peltonen, E.; Riekki, J.; Sillanpää, M.J. Scaling up an Edge Server Deployment. In Proceedings of the 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Austin, TX, USA, 23–27 March 2020; pp. 1–7.
17. Raeisi-Varzaneh, M.; Dakkak, O.; Habbal, A.; Kim, B.S. Resource scheduling in edge computing: Architecture, taxonomy, open issues and future research directions. *IEEE Access* **2023**, *11*, 25329–25350. [CrossRef]
18. Elgendy, I.A.; Meshoul, S.; Hammad, M. Joint Task Offloading, Resource Allocation, and Load-Balancing Optimization in Multi-UAV-Aided MEC Systems. *Appl. Sci.* **2023**, *13*, 2625. [CrossRef]
19. Shruthi, G.; Mundada, M.R.; Supreeth, S.; Gardiner, B. Deep learning-based resource prediction and mutated leader algorithm enabled load balancing in fog computing. *Int. J. Comput. Netw. Inf. Secur.* **2023**, *15*, 84–95. [CrossRef]
20. Kasi, S.K.; Kasi, M.K.; Ali, K.; Raza, M.; Afzal, H.; Lasebae, A.; Naeem, B.; Ul Islam, S.; Rodrigues, J.J. Heuristic edge server placement in industrial internet of things and cellular networks. *IEEE Internet Things J.* **2020**, *8*, 10308–10317. [CrossRef]
21. Luo, F.; Zheng, S.; Ding, W.; Fuentes, J.; Li, Y. An Edge Server Placement Method Based on Reinforcement Learning. *Entropy* **2022**, *24*, 317. [CrossRef]
22. Li, B.; Hou, P.; Wu, H.; Hou, F. Optimal edge server deployment and allocation strategy in 5G ultra-dense networking environments. *Pervasive Mob. Comput.* **2021**, *72*, 101312. [CrossRef]
23. Asghari, A.; Sohrabi, M.K. Multiobjective edge server placement in mobile-edge computing using a combination of multiagent deep q-network and coral reefs optimization. *IEEE Internet Things J.* **2022**, *9*, 17503–17512. [CrossRef]
24. Lähderanta, T.; Leppänen, T.; Ruha, L.; Lovén, L.; Harjula, E.; Ylianttila, M.; Riekki, J.; Sillanpää, M.J. Edge server placement with capacitated location allocation. *arXiv* **2019**, arXiv:1907.07349. [CrossRef]
25. Cao, B.; Fan, S.; Zhao, J.; Tian, S.; Zheng, Z.; Yan, Y.; Yang, P. Large-scale many-objective deployment optimization of edge servers. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 3841–3849. [CrossRef]

26. Chang, L.; Deng, X.; Pan, J.; Zhang, Y. Edge server placement for vehicular ad hoc networks in metropolitans. *IEEE Internet Things J.* **2021**, *9*, 1575–1590. [CrossRef]
27. Zhang, X.; Li, Z.; Lai, C.; Zhang, J. Joint edge server placement and service placement in mobile-edge computing. *IEEE Internet Things J.* **2021**, *9*, 11261–11274. [CrossRef]
28. Ling, C.; Feng, Z.; Xu, L.; Huang, Q.; Zhou, Y.; Zhang, W.; Yadav, R. An edge server placement algorithm based on graph convolution network. *IEEE Trans. Veh. Technol.* **2022**, *72*, 5224–5239. [CrossRef]
29. Cui, G.; He, Q.; Xia, X.; Chen, F.; Jin, H.; Yang, Y. Robustness-oriented k Edge Server Placement. In Proceedings of the 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melboune, Australia, 11–14 May 2020; pp. 81–90.
30. Chen, Y.; Lin, Y.; Zheng, Z.; Yu, P.; Shen, J.; Guo, M. Preference-Aware Edge Server Placement in the Internet of Things. *IEEE Internet Things J.* **2022**, *9*, 1289–1299. [CrossRef]
31. Liu, Z.; Xu, X.; Han, F.; Zhao, Q.; Qi, L.; Dou, W.; Zhou, X. Secure edge server placement with non-cooperative game for internet of vehicles in web 3.0. *IEEE Trans. Netw. Sci. Eng.* **2023**, 11, 4020–4031. [CrossRef]

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.