Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Integrating deep reinforcement learning with pointer networks for service request scheduling in edge computing

Yuqi Zhao [a,c], Bing Li [a,b,*], Jian Wang [a,**], Delun Jiang [a], Duantengchuan Li [a]

[a] *School of Computer Science, Wuhan University, Wuhan, China*
[b] *Complex Network Research Center, Wuhan University, Wuhan, China*
[c] *Latent Vector Technology Co., Ltd, Wuhan, China*

ARTICLE INFO

Article history:
Received 8 July 2022
Received in revised form 2 October 2022
Accepted 4 October 2022
Available online 10 October 2022

Keywords:
Edge computing
Pointer networks
Deep reinforcement learning
Scheduling algorithm

ABSTRACT

With the increasing popularity of edge computing, service providers are more likely to deploy services at the edge of the network to reduce the latency of service requests. However, the resources offered by edge servers are extremely limited compared to those in the cloud. Therefore, a challenging issue in edge computing is how to sufficiently utilize service resources at the edge to satisfy as many service requests as possible. Existing service request scheduling methods mainly use a single optimization objective, e.g., resource utilization or running time. In this paper, the issue of service request scheduling with multiple requests is modeled as a sequential problem, where multiple optimization objectives, including resource utilization, running time, and waiting time, are involved. A reinforcement learning model with pointer networks is proposed to construct scheduling policies. Experiments conducted on three representative real-world datasets show that our proposed approach outperforms several state-of-the-art methods on the three metrics.

1. Introduction

With the development of edge computing technologies, more and more services are being deployed on edge servers, which can significantly reduce the latency of service requests [1]. However, when a resource-constrained edge server receives multiple user requests simultaneously, these requests will be queued in the server due to the limited resources offered by edge servers [2,3]. Task scheduling with multiple service requests thus becomes an essential topic in edge computing. The service requests scheduling problem refers to arranging the execution order of requests and aims to improve the overall quality of services (QoS) of service requests.

Existing literature on service request scheduling can be roughly divided into meta-heuristic-based and deep learning-based methods. Most meta-heuristic-based methods [4–9] consider single-objective optimization problems, in which only resource utilization or running time are considered, and only a few of them focus on multi-objective optimization objectives. Moreover, meta-heuristic algorithms often require a long-running time for iteration to achieve a better solution, which cannot meet the low latency requirements of service requests in edge environments. Deep learning techniques improve the generalization of models. Some researchers [10,11] utilize deep learning techniques to model task scheduling and predict future loading conditions, which have shown significant improvements to meta-heuristic algorithms. However, these approaches can hardly achieve ideal scheduling performance due to the lack of high-quality tags for model training. On the other hand, many classical operating system scheduling methods [12,13] can be improved to be applied to the edge computing environment. For example, some deep reinforcement approaches [14,15] address the problem by slicing service request sequences into multiple time slices and considering the status of edge servers as the model states in training steps. However, the serialization characteristics of service requests, which are conducive to preventing the server from scheduling across time slices when processing requests, have been rarely explored.

Considering this limitation, this paper models service request scheduling in edge environments as a sequence-to-sequence (Seq2Seq) task. However, traditional Seq2Seq models are less scalable in solving the problem where the output sequence length depends on the input sequence length since its output relies heavily on the dictionary length, which is fixed and cannot handle variable-length sequences. We adopt the pointer networks proposed by Vinyals et al. [16], which can output a probability distribution concerning the input sequence, to handle the inputs with variable lengths. Pointer networks get prediction results

* Corresponding author at: School of Computer Science, Wuhan University, Wuhan, China.
** Corresponding author.
E-mail addresses: bingli@whu.edu.cn (B. Li), jianwang@whu.edu.cn (J. Wang).

by outputting a probability distribution named the pointer. In other words, the traditional Seq2Seq model outputs a probability distribution for each output task sequence, while the pointer networks can output a probability distribution for each input task sequence.

In this paper, we propose a deep reinforcement learning model based on pointer networks to solve the multi-objective optimization problem of service request scheduling.

- Considering that a single objective cannot address the need for service request scheduling in edge computing, this paper proposes a novel multi-objective optimization strategy by integrating resource utilization, running time, and waiting time.
- Since the input sequence of the service request scheduling task is variable, and the number of user service requests allocated to each server is uncertain, a deep reinforcement learning model based on pointer networks (RLPNet) is proposed to schedule service requests.
- We performed extensive experiments on three representative real-world datasets to demonstrate the effectiveness of the proposed method. Experimental results show that our proposed approach outperforms several state-of-the-art methods on all three metrics.

The remainder of this paper is organized as follows. Section 2 discusses the related work. Section 3 presents the scenario and problem description. Section 4 elaborates on the proposed RLPNet model in detail. Section 5 discusses the experimental results on three real datasets. Section 6 concludes this study.

2. Related work

In this section, we briefly review recent works that are directly related to this paper and highlight the differences.

2.1. Task scheduling methods

Service request scheduling or task scheduling in edge computing environments mainly adopts meta-heuristic and deep learning algorithms.

2.1.1. Meta-heuristic scheduling algorithms

Meta-heuristic algorithms, [17–19], can provide an optimal solution for task scheduling in the edge computing environment. Song et al. [5] designed a multi-service task computing offloading algorithm (MTCOA) to achieve an optimal solution for the computational offloading of multi-service tasks. But the algorithm only focuses on the computational cost and resource utilization. To address the problems of low resource utilization, long task processing delay, and unbalanced system load in edge computing, Wang et al. [6] proposed a simulated annealing fusion algorithm based on task processing delay. In [7], the researchers modeled the dependency relationships between application tasks as directed acyclic graphs for mobile edge computing environments. An online approach is used to solve the priority-based application assignments and scheduling problems for the timeout rate of tasks. Sun et al. [8] proposed a series of task assignment policies for the workflow scheduling problem in edge environments. These policies were combined with a greedy policy to construct an improved greedy search algorithm. However, this approach only focuses on optimizing the completion time of workflows. Zhao et al. [9] applied the parallel-batch processing machines (P-BPM) task processing model to edge computing scenarios and proposed an offline task scheduling algorithm based on the ant colony algorithm. These approaches have further accelerated the development of task scheduling research. However, they only consider the optimization problem with a single objective and neglect multiple optimization objectives. Moreover, meta-heuristic algorithms often require a longer running time for iteration to arrive at a better solution, which cannot meet the low latency requirements of service requests in edge environments.

2.1.2. Deep learning-based scheduling algorithm

The development of deep learning provides a new perspective on resource scheduling. In [10], the authors proposed a residual recurrent neural network based on the Asynchronous-Advantage-Actor-Critic(A3C) approach for efficient resource allocation in IoT. Cai et al. [11] proposed a distributed convolutional neural network model to achieve efficient interactions between edge network devices. Based on [11], Zou et al. [14] applied the Markov decision process and used the A3C deep reinforcement learning to further optimize the model. Zheng et al. [15] utilized the Deep-Q-Network (DQN) algorithm to address the complexity and high-dimensional problem of workload scheduling. Deep learning-based scheduling algorithms have shown significant performance improvements over meta-heuristic algorithms. The service request scheduling problem in edge environments can be modeled as a sequential problem, which was ignored by existing methods.

2.2. Multi-objective optimization

Multi-objective optimization problems have been widely studied in Web services and reinforcement learning research. For example, the issue of QoS-aware Web service composition and selection is sometimes modeled as a multi-objective optimization problem. Also, many previous works [20,21] have studied multi-objective reinforcement learning, roughly divided into two categories: single-policy and multiple-policy methods.

2.2.1. Single-policy methods

A multi-objective Q-network is proposed in [22] to describe the dynamic change of the objective weight under the multi-objective optimization problem by combining multiple objectives into a single objective through linear weighting. One optimization objective is defined as the main objective, and others are defined as constraints, which are transformed into single-objective optimization problems. For example, an improved multi-objective reinforcement learning (MORL) algorithm is proposed in [23], which uses the championship algorithm to select an essential objective among multiple optimization objectives as a preference to solve the task offloading problem under edge computing.

2.2.2. Multiple-policy methods

The Pareto optimal solution set is obtained by designing an algorithm to solve the multi-objective optimization problem. For example, McMahan et al. [24] proposed the first federated learning (FL) algorithm known as "Federated Averaging" (FedAvg), which is a synchronous update scheme that proceeds in several rounds. At each round, the central server sends the current global model to a subset of users, using its respective local data to update the received model. Upon receiving the updated local models from users, the server performs aggregation, such as simple averaging, to update the global model. However, FedAvg relies on a coordinate-wise averaging of local models to update the global model. According to Wang et al. [25], in neural network (NN) based models, such coordinate-wise averaging might lead to sub-optimal results due to the permutation invariance of NN parameters. Based on [24], a new algorithm FedMGDA is proposed in [26] to improve the fairness and robustness of federated learning. This algorithm aims at multi-objective optimization, guaranteed to converge to Pareto stationary solutions.

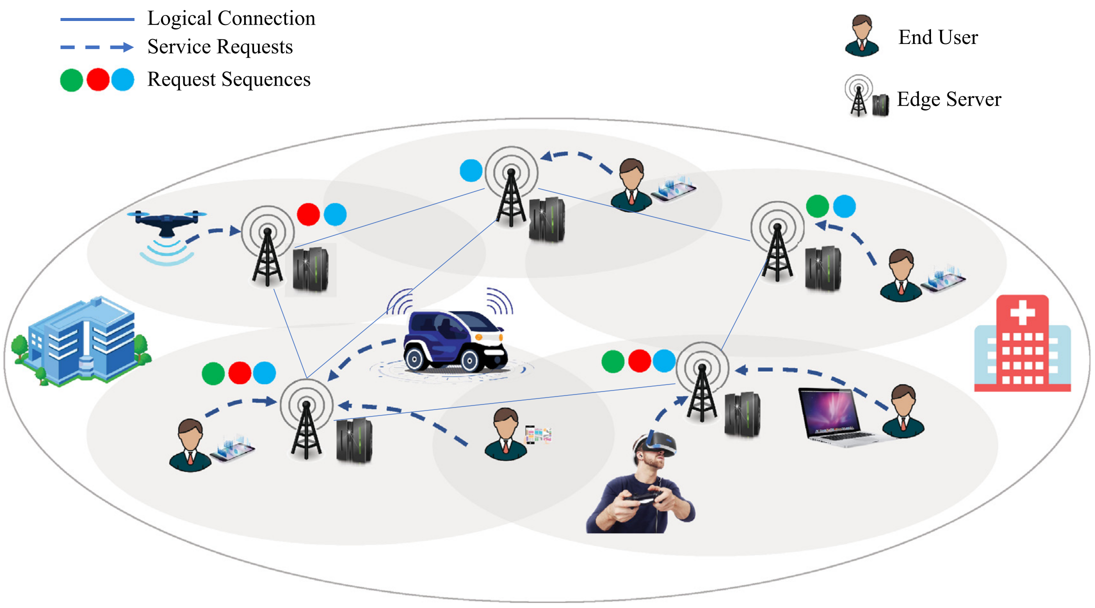

Fig. 1. Typical edge computing scenario.

To summarize, existing work on the problem of scheduling service requests in edge computing environments remains somewhat limited. These approaches tend to consider only a single optimization objective and ignore the impact of other metrics on task scheduling. This paper combines resource utilization, running time, and waiting time in service request scheduling. The multi-objective optimization strategy is used to model the service request scheduling problem. A deep reinforcement learning model based on pointer networks is adopted to model the scheduling sequence, which improves the service quality in edge computing. In particular, for selecting the solution for the multi-objective optimization problem, we consider that the training method of deep reinforcement learning requires a reward function, and a Pareto solution set cannot describe the function. Therefore, this paper uses a linear weighting approach to combine multiple objectives into a single objective.

3. Scenario and problem description

In this section, we present a typical scenario of edge computing request scheduling and formalize the problem formulation.

3.1. Scenario description

We employ the Internet of Vehicles (IoV), a typical kind of application in edge computing [22], to illustrate the service request scheduling problem. Besides realizing communications among vehicles and surrounding things, IoV can provide various value-added services for vehicles, such as finding free parking spaces, sharing road conditions, and diverse entertainment services. With the popularization of automobiles, the number of vehicles connected to the network continues to increase, and a large amount of communication data is generated. Traditional cloud computing methods cannot meet the low-latency requirements of IoV. In this paper, edge computing is integrated with IoV, using onboard sensors to sense the surrounding environment, processing and analyzing the collected data through edge servers deployed on the roadside unit close to the vehicle, and broadcasting it to other surrounding vehicles. Fig. 1 shows a typical scenario of service request scheduling in edge computing.

However, complex computing tasks are placed in the cloud by task offloading during driving operation, and the data transfer latency between remote cloud servers and terminals can significantly increase the response latency of the end terminal. Therefore, it is a very effective and common solution to deploy these high resource-consuming services in the edge environment closer to the end-users and to resort to the ability of edge computing to solve the above problems [27]. Furthermore, effective service request scheduling becomes an urgent issue in this area when a large number of requests are coming simultaneously.

3.2. Problem definition

3.2.1. Edge server

We define the edge server set $\mathcal{S}$:

$$\mathcal{S} = \{s_1, s_2, \ldots, s_m\}\,, \tag{1}$$

where a single server s_j is represented by a four-dimensional vector as:

$$s_j = \left(c_j, o_j, b_j, m_j\right)\,. \tag{2}$$

Each dimension of the vector represents the CPU, I/O, bandwidth, and memory capacity of edge server s_j, respectively.

3.2.2. Service request

Assume that at a certain moment, N service requests are waiting to be executed in a certain edge server. We define the service request set $\mathcal{Q}$ as:

$$\mathcal{Q} = \{q_1, q_2, \ldots, q_N\}\,. \tag{3}$$

Each request q_i is represented by a seven-dimensional vector as:

$$q_i = (c_i, o_i, b_i, m_i, T_i, t_i, \pi_i)\,, \tag{4}$$

where the first four-dimensional vectors (*i.e.*, c_i, o_i, b_i, m_i) represent the CPU, I/O, bandwidth, and memory required to execute request q_i, respectively. T_i represents the timestamp that request q_i arrives at the edge server. t_i represents the time required for the execution of q_i. π_i represents the set of all edge servers that can receive the service request, which is calculated according to the coverage relationship between edge servers and service requests. π_i is expressed as:

$$\pi_i = \left\{s_j \mid r_j \geq \left\| p_i - g_j \right\|_2, s_j \in S\right\}, \tag{5}$$

where p_i represents the coordinates of service request q_i, g_j represents the coordinates of edge server s_j, and r_j represents the coverage radius of edge server s_j.

3.3. Optimization objective

The optimization objectives of this paper are resource utilization, runtime and average waiting time. And the three objectives are combined into one objective using a linear weighting approach.

3.3.1. Resource utilization

Resource utilization exhibits the average utilization efficiency of an edge server in processing service requests. (c_j, o_j, b_j, m_j) denote the CPU, I/O, bandwidth and memory utilization of the jth edge server in processing the entire input sequence, respectively. The optimization objective of resource utilization is represented by $reward_1$.

$$reward_1 = \frac{1}{m}\sum_{j=1}^{m} max(c_j, o_j, b_j, m_j). \tag{6}$$

A higher $reward_1$ value indicates a higher resource utilization of the edge server j.

3.3.2. Running time

The average running time refers to the average time required for all edge servers to execute service requests. Without considering the request timeout, the set of edge servers $\mathcal{S}$ is known. According to an input and output pair $(\mathcal{Q}, C^{\mathcal{Q}})$, the pseudo-code of the algorithm to calculate the running time is shown in Algorithm 1.

The returned result T_{map} is the edge server's running time set corresponding to $(\mathcal{Q}, C^{\mathcal{Q}})$. The optimization objective of running time is expressed as:

$$reward_2 = \frac{1}{m}\sum_{j=1}^{m} T_{map_j}, \tag{7}$$

where T_{map_j} represents the running time of the jth edge server. The optimization goal is to minimize the value of $reward_2$.

3.3.3. Waiting time

The waiting time W_i of service request i represents the time between the arrival of a service request at an edge server and the completion of the request. The optimization objective of the waiting time is defined by $reward_3$.

$$reward_3 = \frac{1}{m}\sum_{j=1}^{m}\left(\frac{1}{N_i}\sum_{i=1}^{N_i} W_i\right), \tag{8}$$

where N_i represents the total number of service requests in edge server j.

3.3.4. Normalization

Classical methods for multi-objective optimization problems include weighted sum method (also called weighted linear combination) [28], interactive methods [29], ϵ-constraint method [29], and Pareto-dominated methods [30]. Among them, the interactive approaches are less applicable, and it is generally used in specific scenarios and is not suitable for resource scheduling problems in edge environments. ϵ-constraint method converts

Algorithm 1: Running time calculation

```
Input: $m$ (the number of servers), $n$ (the number of
       service requests), $\mathcal{Q}$ (model input)
Output: C: model output
1  map ← []·$m$   /* Record the service requests running in the server */
2  $T_{map}$ ← [0]·$m$   /* Record the time spent by each server */
3  for $i$=0, $i \leq n-1$ do
4      idx ← $C_i$
5      $q$ ← $\mathcal{Q}$[idx]
6      /* Randomly select server $s_j$ according to feature $\pi_i$ of $q_i$ */
7      if $s_j[cpu] < q[cpu]$ then
8          while $MIN(t)>0$ do
9              /* $t[j]$ is the collection of time required for the
                  execution of all service requests in $map[j]$ */
10             $T_{map}[j] \leftarrow T_{map}[j] + 1$
11             $t[j] \leftarrow t[j] - 1$
12         end
13         while $MIN(t) \leq 0$ do
14             $k \leftarrow ARGMIN(t)$
15             $s_j \leftarrow s_j$+map[$k$]
16             map[$j$].REMOVE($k$)
17         end
18     else
19         $s_j \leftarrow s_j$-[q[cpu]]
20         map[$j$].ADD($q$) /* Add $q$ into map */
21     end
22 end
23 $T_{map}[j] \leftarrow T_{map}[j] + MAX(t[j])$
24 return $T_{map}$
```

the original multi-objective optimization problem into a single-objective optimization problem by converting the objectives into a form of constraints, after which the problem can be solved by the single-objective optimization method. However, considering the specificity of our scenario, the existence of interconnections among the objectives makes it difficult to convert into a single constraint. In addition, Pareto is the classical model in multi-objective optimization, and it is based entirely on the original data, without transforming the problem into a single-objective problem. However, the optimal solution of the Pareto model is a set, which contains more than one optimal solution, so it is difficult to exhaust and find all the Pareto optimal solutions.

The weighted linear combination, which has the advantage of being simple to implement, uses only the scaled values to represent the original objective and is relatively easy to solve. The weighted linear combination is applicable to the case where multiple evaluation indicators are independent of each other. Therefore, we use here a weighted linear combination (WLC) method to solve the resource scheduling problem in the edge environment. The key to achieving better results is selecting appropriate weights for each optimization objective. In response to the problems raised in this paper, we believe that the above three optimization objectives have the same importance. Based on this, we choose to normalize these three optimization objectives so that the change of each optimization objective will identically affect the model. Enumeration experiments in Section 5.2 will discuss the settings of weights. The final optimization objective is defined as a reward:

$$reward = \alpha * reward_1 + \beta * reward_2 + \gamma * reward_3. \tag{9}$$

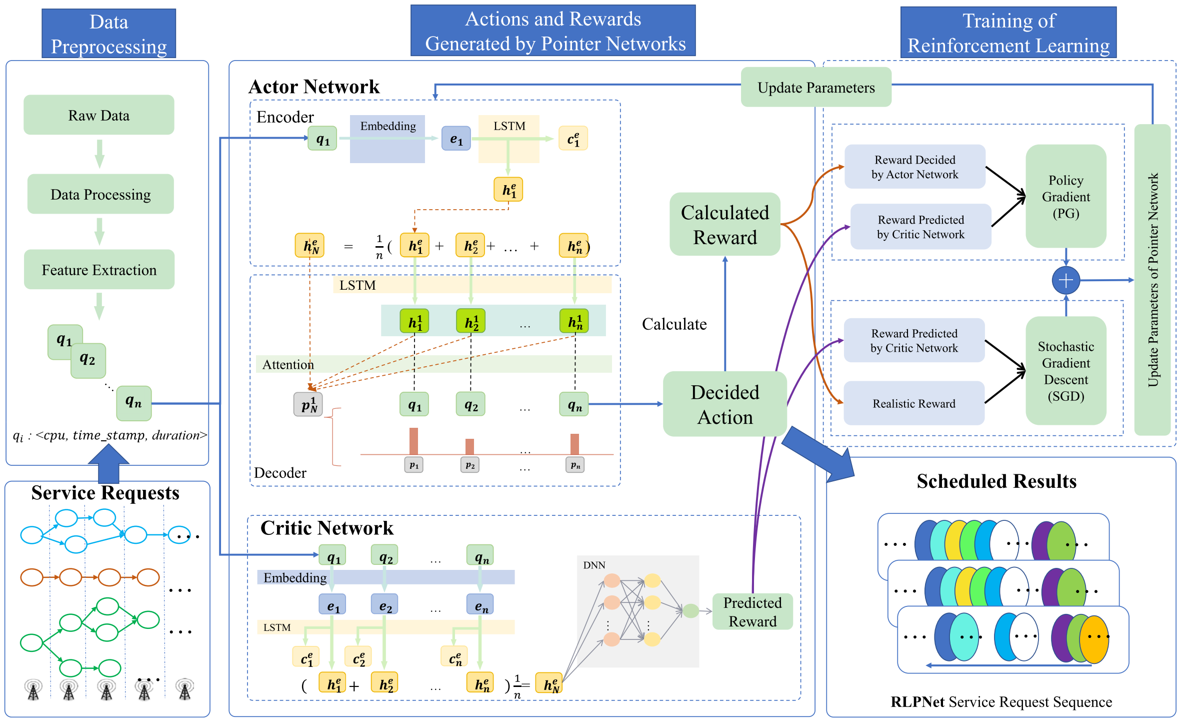
Data Preprocessing
Raw Data
Data Processing
Feature Extraction
q_i : <cpu, time_stamp, duration>
Service Requests
Actions and Rewards Generated by Pointer Networks
Actor Network
Encoder
Embedding
LSTM
Attention
Decoder
Critic Network
DNN
Predicted Reward
Calculated Reward
Calculate
Decided Action
Update Parameters
Training of Reinforcement Learning
Reward Decided by Actor Network
Reward Predicted by Critic Network
Policy Gradient (PG)
Realistic Reward
Stochastic Gradient Descent (SGD)
Update Parameters of Pointer Network
Scheduled Results
RLPNet Service Request Sequence

Fig. 2. Overview of the RLPNet framework.

4. Method

4.1. Overview of RLPNet

The overall process of the RLPNet model consists of three main parts, as shown in Fig. 2. The first part is data preprocessing, which transforms the input data from different datasets into the canonical format defined in the paper.

Followed by the Actions and Rewards Generation by Pointer Networks module of the model, the formatted data is inputted to an Actor-network and a Critic-network, respectively. The Actor-network is essentially a pointer network. In the Actor-network, the input data is encoded into an intermediate vector by an encoder composed of long short-term memory (LSTM) as the input of the decoder. After performing the attention mechanism of the pointer network, the output becomes a probability distribution, and the final output sequence is decided according to the probability distribution. Moreover, the model calculates the reward value from this output sequence. For the Critic-network, it will make an encode operation similar to the one in the Actor-network on the input data and then output a reward prediction value after DNN.

Finally, we discuss the reinforcement learning training process of the model. For Actor-network, it takes the predicted reward of the Critic-network as the baseline and uses the policy gradient descent for model training. For the Critic-network, it takes the difference between its predicted reward and the reward output of the Actor-network as the optimization objective. It uses stochastic gradient descent for model training.

4.2. Data preprocessing

4.2.1. Input

Based on the π_i characteristics of service request q_i, the edge server closest to q_i in the set of π_i is selected in priority, and the service request is sent to this edge server for processing by combining the coordinate information of service requests and edge servers.

4.2.2. Output

The result obtained through the model scheduling strategy is displayed with the subscript sequence of the service request as the output. The output result represents the execution sequence of the edge server set $\mathcal{S}$ for the service request set $\mathcal{Q}$. It should be noted that when the remaining resources of the edge server are sufficient, multiple service requests can be run simultaneously; that is, the service requests are executed in parallel in the server. The output $C^{\mathcal{Q}}$ is defined as:

$$C^{\mathcal{Q}} = \{C_1, C_2, \dots, C_n\}\,. \tag{10}$$

The model scheduling strategy is executed independently in the edge server without interference in a multi-server environment. Each server has its output $C^{\mathcal{Q}}$. $C^{\mathcal{Q}}$ represents the sequence of solutions associated with the input request set $\mathcal{Q}$, and its length is the same as the length of $\mathcal{Q}$. Each element in $C^{\mathcal{Q}}$ represents the subscript of a service request, which is similar to the traveling salesman problem (TSP), where the city subscript sequence is the output result.

4.3. Actions and rewards generation by pointer networks

Since both input and output can be serialized in this scenario, the Seq2Seq model can be used to solve this type of problem effectively. Similar to the work [31], two long short-term memory networks (LSTM) are used as the encoding and decoding stages of the sequence to solve the language-translation problem. There are many options for the Seq2Seq model. The disadvantage of the traditional Seq2Seq model is that its output sequence length depends on the dictionary length defined in advance. For example, in solving the TSP problem, the city ids $city_1, city_2, \dots, city_n$ are used as the dictionary, and the method will fail when the number of input cities is greater than n. Therefore, the traditional Seq2Seq model is less scalable in solving the problem where the output sequence length depends on the input sequence length.

In this paper, we employ the pointer network model, which improves the attention mechanism of the traditional Seq2Seq

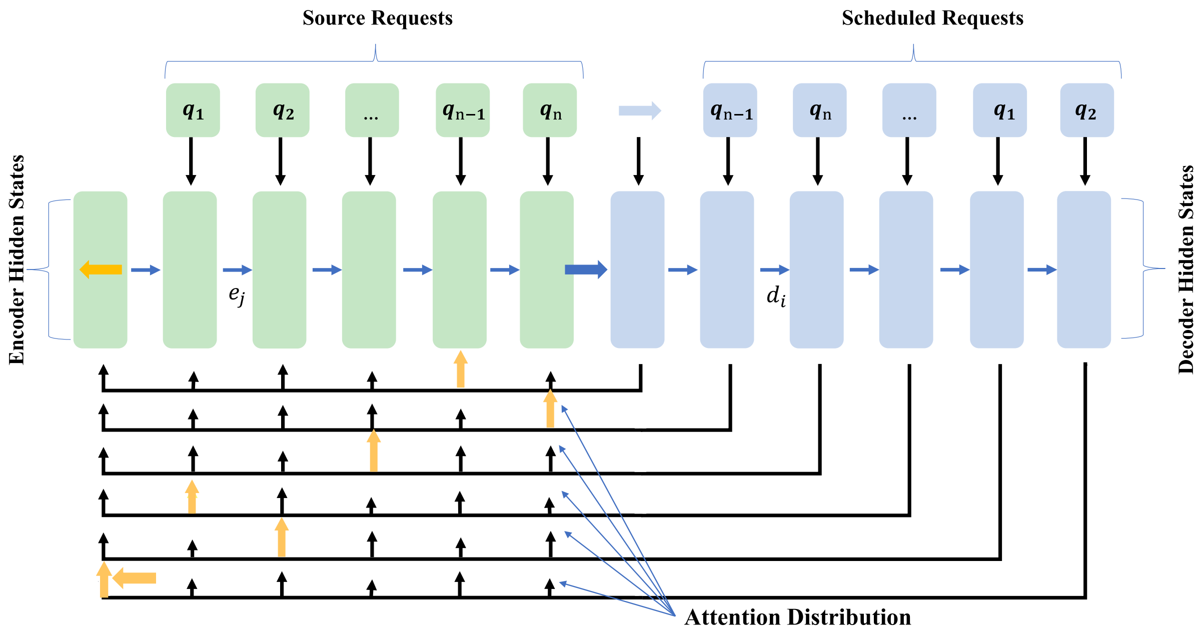

Fig. 3. Overview of the pointer network.

model so that the output of the pointer network can effectively point to a specific position of the input sequence. In addition, it differs from other Seq2Seq models in that the standard Seq2Seq output is a probability distribution for the dictionary. In contrast, the pointer network output is a probability distribution for the input sequence. Thus, the output sequence no longer depends on the dictionary but is related to the input sequence.

4.3.1. The structure of Pointer network

The Pointer network shown in Fig. 3 consists of two recurrent neural networks (RNN), that is an encoder and a decoder. Each recurrent neural network comprises multiple LSTM units.

As shown in Fig. 3, the input is serialized and read by the encoder, and one service request is read at each step. q_i indicates the $i-$th service request, and the encoder acts as an RNN to transform the input sequence into an intermediate vector expression. The decoder uses the attention mechanism in each step to select the service request with the largest weight in the input sequence, and when a service request is selected, it is used as the input to the decoder in the next step.

4.3.2. Attention mechanism of pointer network

For the attention mechanism used in the traditional Seq2Seq model, the principle is to add the hidden layer output of the encoder according to a certain weight and then splice it into the hidden layer output of the decoder. In this way, it plays the so-called "soft alignment" role, thereby improving the entire model's prediction accuracy. The traditional attention mechanism is calculated as follows:

$$u_j^i = v^T \tanh\left(W_1 e_j + W_2 d_i\right) \quad j \in (1, \ldots, n), \tag{11}$$

$$a_j^i = softmax\left(u^i\right) \quad j \in (1, \ldots, n), \tag{12}$$

$$d_i' = \sum_{j=1}^{n} a_j^i e_j. \tag{13}$$

Parameters v^T, W_1 and W_2 in Eq. (11) can be trained. a_j^i represents the weight assigned to the input sequence by the decoder in the ith step. Eq. (13) calculates the weighted summation using the weight a_j^i and output e_j of the hidden layer in the encoder at the jth step, and the obtained d_i' is added to output d_i of the hidden layer in the decoder at the ith step. Finally, we can obtain the output of the model.

It can be seen from Eq. (12) that a_j^i itself is the weight for the input sequence. Therefore, the pointer network simplifies the traditional attention mechanism and regards a_j^i as a pointer to the input sequence, and the output of the decoder at each step selects the element according to the weight sequence. The attention mechanism of the pointer network is defined as follows.

$$u_j^i = v^T \tanh\left(W_1 e_j + W_2 d_i\right) \quad j \in (1, \ldots, n), \tag{14}$$

$$p\left(C_i \mid C_1, \ldots, C_{i-1}, \mathcal{Q}\right) = softmax\left(u^i\right). \tag{15}$$

4.4. Reinforcement learning module

We define the service request scheduling problem in the edge computing environment as a multi-objective optimization problem. Two ways, including deep learning and reinforcement learning, can be used to train the model. The training effect of deep learning depends on the label quality of the dataset, but the current research in this field lacks labeled datasets. At the same time, pure reinforcement learning usually requires a very large state–action space. Therefore, this paper proposes a deep reinforcement learning model based on pointer networks to solve the multi-objective optimization problem of service request scheduling.

4.4.1. Strategy gradient

This paper uses the policy gradient-based deep reinforcement learning method proposed in [32] to optimize the parameters of the pointer network. We define the parameters in the pointer network as θ and use $reward(C^{\mathcal{Q}}|\mathcal{Q})$ to denote the value of the reward function when the policy $C^{\mathcal{Q}}$ is adopted for a known set $\mathcal{Q}$ of service requests.

The expectations of reward$(C^{\mathcal{Q}}|\mathcal{Q})$ are defined as follows:

$$J(\theta \mid \mathcal{Q}) = \mathrm{E}_{C \sim p\theta(.|\mathcal{Q})} \text{ reward } \left(C^{\mathcal{Q}} \mid \mathcal{Q}\right), \tag{16}$$

where $J(\theta|\mathcal{Q})$ represents the optimization objective of the pointer network, i.e., minimizing the expectation of rewards. The execution process of the strategy gradient uses the reinforcement

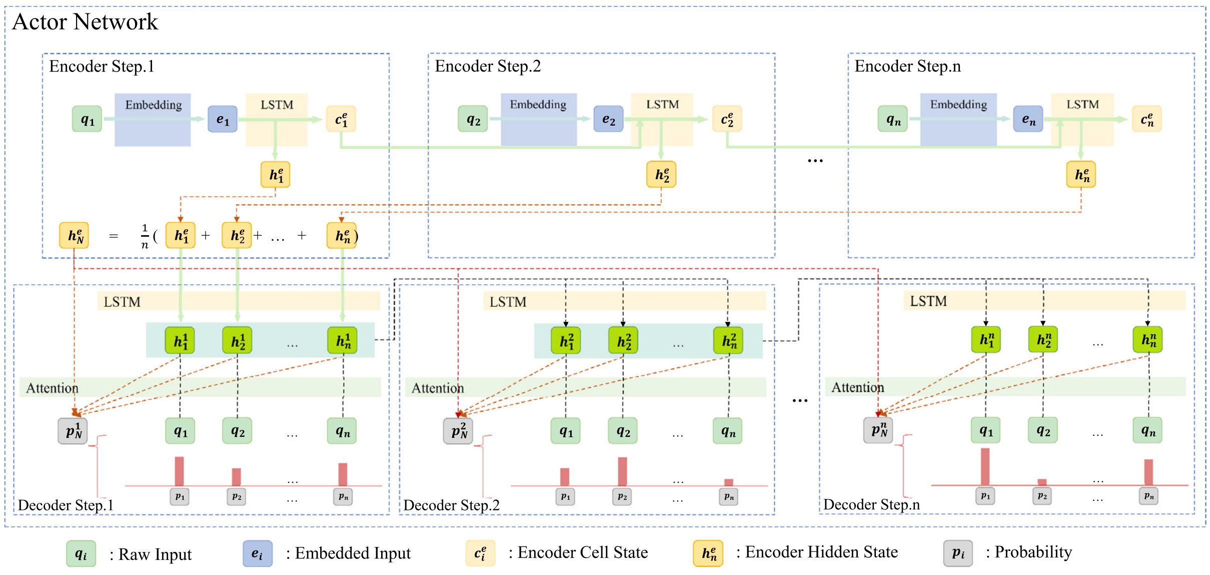

Fig. 4. Outline of the Actor-network.

learning algorithm proposed in [33] to optimize θ:

$$\nabla_\theta J(\theta \mid \mathcal{Q}) = \mathrm{E}_{C\sim p\theta(.\mid\mathcal{Q})}\left[\left(\text{ reward } (C \mid \mathcal{Q}) - b(\mathcal{Q})\right)\nabla_\theta \log_{p\theta}(C \mid Q)\right], \tag{17}$$

where $b(\mathcal{Q})$ is a baseline function irrelevant to the adopted strategy $C^{\mathcal{Q}}$. Its function is to reduce the variance of the gradient by selecting the exponential moving average (EMA) of the rewards obtained during the training of the pointer network.

Considering that the actual training of the pointer network is to combine multiple input sequences into a batch for parallel training, the batch size of the training data is defined as B, and Eq. (17) can be approximately equal to:

$$\nabla_\theta J(\theta) \approx \frac{1}{B}\sum_{i=1}^{B}\left(reward\left(C_i^{\mathcal{Q}} \mid \mathcal{Q}_i\right) - b\left(\mathcal{Q}_i\right)\right)\nabla_\theta \log_{p\theta}\left(C_i^{\mathcal{Q}} \mid \mathcal{Q}_i\right). \tag{18}$$

4.4.2. Actor-Critic model

We choose the Actor–Critic model to train the pointer network. The model is divided into two main parts: an Actor-network and a Critic-network.

Algorithm 2: Training process

```
Input: $\mathcal{Q}$ (the training sample of each round), $N$ (the
       number of training rounds), $B$ (the batch size of
       each round input)
Output: RLPNet model parameters
1  Initialize Actor-network parameters $\theta$ and Critic-network
     parameters $\theta_v$
2  for round=1, round$\leq N$ do
3      idx $\leftarrow C_i$
4      $\mathcal{Q}_i \leftarrow$ SAMPLE($\mathcal{Q}$) for $i \in \{1,\dots,B\}$
5      $C_i^{\mathcal{Q}} \leftarrow$ SAMPLE($p\theta(. \mid \mathcal{Q})$) for $i \in \{1,\dots,B\}$
6      $b_i \leftarrow b_{\theta_v}(\mathcal{Q}_i)$ for $i \in \{1,\dots,B\}$
7      $j_\theta \leftarrow$
         $\frac{1}{B}\sum_{i=1}^{B}\left(\text{reward } \left(C_i^{\mathcal{Q}} \mid \mathcal{Q}_i\right) - b_i\right)\nabla_\theta \log_{p\theta}\left(C_i^{\mathcal{Q}} \mid \mathcal{Q}_i\right)$
8      $l_{\theta_v} \leftarrow \frac{1}{B}\sum_{i=1}^{B}\|b_{\theta_v}(\mathcal{Q}_i) - \text{ reward } \left(C_i^{\mathcal{Q}} \mid \mathcal{Q}_i\right)\|_2^2$
9      $\theta \leftarrow$ ADAM $(\theta, j_\theta)$
10     $\theta_v \leftarrow$ ADAM $\left(\theta_v, \nabla_{\theta_v}, l_{\theta_v}\right)$
11 end
```

In the scenario of this paper, the Actor-network is a basic pointer network (see Fig. 4). The Critic-network shown in Fig. 5 judges the behavior scores based on the behaviors generated by the Actor-network and guides the Actor-network to update network parameters. This paper designs the Critic-network as an RNN and deep neural network (DNN). RNN is similar to the encoder of the pointer network structure. The parameter of the Critic-network is represented as θ_v. The network predicts the reward value of the adopted strategy $C^{\mathcal{Q}}$ according to the final state of the Actor-network when the set of input service requests $\mathcal{Q}$ is known. The Critic-network takes the mean square error between the predicted reward and the reward decided by the Actor-network of strategy $C^{\mathcal{Q}}$ as the optimization objective and uses stochastic gradient descent for model training.

$$l\left(\theta_v\right) = \frac{1}{B}\sum_{i=1}^{B}\left\|b_{\theta_v}\left(\mathcal{Q}_i\right) - reward\left(C_i^{\mathcal{Q}} \mid \mathcal{Q}_i\right)\right\|_2^2, \tag{19}$$

where $b_{\theta_v}(\mathcal{Q}_i)$ is the reward value predicted by the Critic-network, and $reward\left(C_i^{\mathcal{Q}}|\mathcal{Q}_i\right)$ is the reward decided by the Actor-network. The pseudo-code of the training process is shown in Algorithm 2.

5. Experimental evaluation

In this section, we conducted a series of experiments from the perspective of service providers using a real-world trajectory dataset. Three groups of experiments were designed to answer the three research questions:

- RQ1: How do different reward weights affect multi-objective optimization results?
- RQ2: Can the proposed RLPNet outperform existing state-of-the-art approaches under different data scales?
- RQ3: Can RLPNet perform better than metaheuristic algorithms?

5.1. Datasets

We experimented with three publicly accessible datasets.

EUA-dataset [27]**:** The EUA-dataset contains geographic locations of 125 base stations in the Melbourne CBD and 816 mobile users around these base stations. In this scenario, we

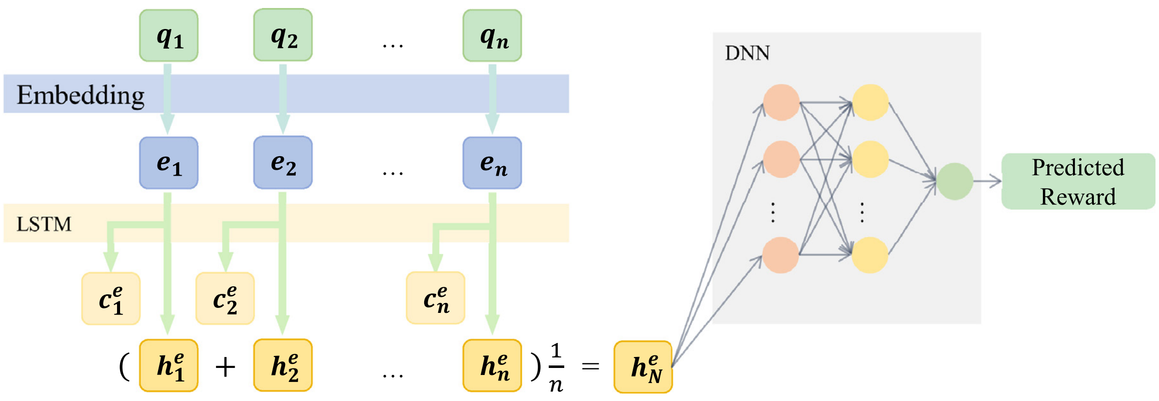

Fig. 5. Outline of the Critic-network, which can generate predicted reward.

regard base stations and mobile users as edge servers and service requests. Since the EUA-dataset contains only edge server and user information, the experimental data is constructed using the actual edge server and user information together with simulated service request data. For each feature $(c_i, o_i, b_i, m_i, T_i, t_i, \pi_i)$ of the service requests defined in Eq. (4), we randomly generate the remaining features except for π_i, which is counted according to the coverage relationship between the selected edge server and the service request (Eq. (5)). For the first four dimensions of the service request regarding resource occupation (c_i, o_i, b_i, m_i), we assign them to random numbers between 1 and 10; for the total running time T_i, we assign it to random numbers between 0 and 4; and for the running time t_i, we set it to random numbers between 1 and 20.

Google Cluster Trace [34]**:** This dataset comes from a real edge-cloud collaboration system. It is originally designed for cloud environments, but has now been extended to cover data in edge environments. The dataset records the lifecycle of tasks in servers, including creation, waiting, interruption, and end, as well as the characteristics of tasks, such as resource occupancy (including CPU, RAM, and Disk), and the timestamp when the task arrives at the server.

Alibaba Cluster Trace [35]**:** This dataset records the information of multiple edge servers, such as the number of CPU cores, memory size, and disk size. The dynamic information of servers, e.g., the resource utilization rate at a certain time, is recorded in the form of timestamps.

For the Google cluster trace and the Alibaba cluster trace datasets, request data are derived from real recorded values. In each experiment, we randomly select 400,000 records from each dataset.

5.2. Parameter determination

In the RLPNet model, three parameters need to be determined. It can be seen that the values of hyperparameters α, β and γ will directly affect the performance of the model, so it is necessary to explore the influence of different values on the model. Given that the resource utilization takes values between 0 and 1, and the running time and waiting time are positively related to the number of tasks n. We use different penalties to regularize the constraints on resource utilization, running time and waiting time, respectively. To offset the impact of different task numbers on the objective function, we take values of 1,1/n, $1/n^2$ for β and γ. In addition, considering that resource utilization is not necessarily related to the number of tasks n, a grid search-like approach is used to select the most suitable parameter for parameter α. We insert different values of parameter $\alpha \in [0.1, 0.2, 0.4, 0.8, 1.0, 2.0, 4.0, 8.0, 10.0]$ to observe the impact of different values of parameter α on resource utilization, running time, and waiting time.For other parameters in the training process, we set the batch size to 128, and each batch selects 100 tasks as input sequences, with a total of 100 epochs.

Table 1
Regularization parameter discussion in the proposed RLPNet method.

α	β	γ	Resource utilization	Running time (ms)	Waiting time (ms)
0.1	$1/n^2$	$1/n^2$	0.861	351	345
0.1	$1/n^2$	$1/n$	0.833	362	314
0.1	$1/n^2$	1	0.820	367	320
0.1	$1/n$	$1/n^2$	0.917	330	422
0.1	$1/n$	$1/n$	0.848	355	327
0.1	$1/n$	1	0.825	365	313
0.1	1	$1/n^2$	0.913	331	420
0.1	1	$1/n$	0.888	340	358
0.1	1	1	0.845	356	327
1	$1/n^2$	$1/n^2$	0.918	329	412
1	$1/n^2$	$1/n$	0.841	358	331
1	$1/n^2$	1	0.826	365	308
1	$1/n$	$1/n^2$	0.918	330	413
1	**1**$/n$	**1**$/n$	**0.851**	**354**	**331**
1	$1/n$	1	0.824	365	313
1	1	$1/n^2$	0.915	331	412
1	1	$1/n$	0.916	330	415
1	1	1	0.847	353	365
10	$1/n^2$	$1/n^2$	0.919	329	423
10	$1/n^2$	$1/n$	0.883	334	353
10	$1/n^2$	1	0.823	366	308
10	$1/n$	$1/n^2$	0.918	329	413
10	$1/n$	$1/n$	0.850	342	348
10	$1/n$	1	0.835	361	310
10	1	$1/n^2$	0.919	329	413
10	1	$1/n$	0.918	329	419
10	1	1	0.845	356	325

To select the best values for the parameters, several algorithms, such as the L-curve method [36], generalized cross-validation (GCV) [37], and discrepancy principle [38], have been proposed. In this paper, the GCV algorithm is leveraged to validate the parameter values in a large range and determine the best ones automatically. In other words, the three regularization parameters are determined heuristically. More parameter details will be discussed in the next section.

5.3. Reward weights analysis (RQ1)

In this experiment, we transformed the multi-objective optimization problem into a single-objective optimization one through the WLC model. We carried out 27 groups of experiments by using the enumeration method, where each group of experiments was run ten times and averaged. Thus the possible values

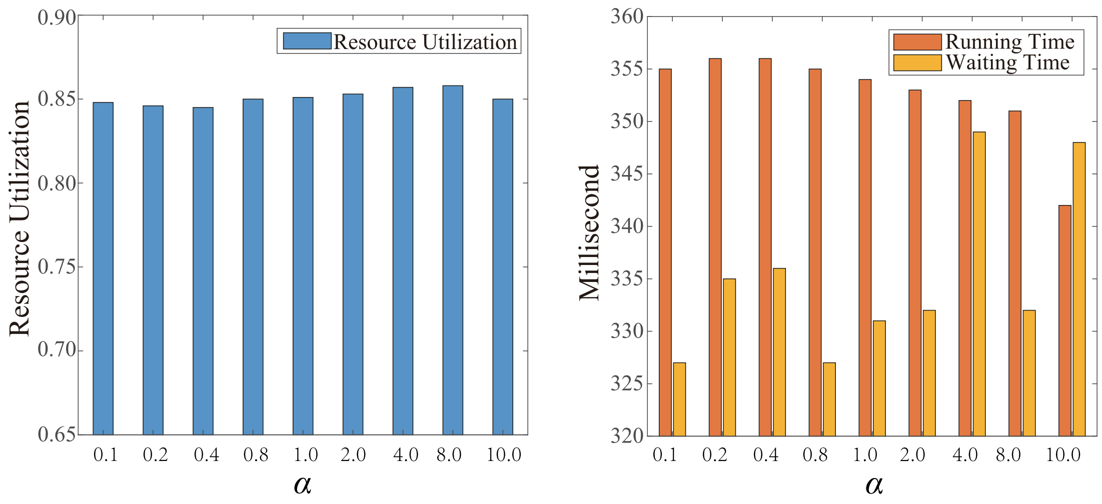

Fig. 6. Influence of parameter α on resource utilization, running time, and waiting time.

of α, β and γ were fully considered. α is the weight of resource utilization, which is a value between [0, 1]. Therefore, we choose one option from the range of 0.1, 1 and 10. The running time is within milliseconds, so the value of its weight β is in the range of $[1/n^2, 1/n, 1]$. The unit of the average waiting time is also the millisecond, so the value of its weight γ is also in the same range.

To reveal the effects of different weight values on multi-objective optimization results, we conducted some comparative experiments and used the enumeration method to test them separately. Experimental results are demonstrated in Table 1, which shows the impact of parameters α, β and γ on the resource utilization, running time, and waiting time.

Firstly, a Pareto-optimal set is extracted from Table 1 according to the principle of mutual non-dominance involved in [19]. In determining the final solution, a high priority will be given to the elements that perform better under the more important optimization objectives, corresponding to the elements in Pareto-optimal front. The elements at the edge of the Pareto-optimal front will be selected in preference. Moreover, because the three metrics,including resource utilization, running time, and waiting time are equally important in the edge computing environment, we select the relatively centered element in the Pareto-optimal front, i.e., the combination of $(1, 1/n, 1/n)$ as the final weight combination. Experimental results in Table 1 also show that the model trained with this combination can effectively optimize the three metrics, and the results are better than those of other baseline methods.

To analyze the effect of different values of parameter α on resource utilization, running time, and waiting time, we further set different values of α, $\alpha \in [0.1, 0.2, 0.4, 0.8, 1.0, 2.0, 4.0, 8.0, 10.0]$. The experimental results are presented in Fig. 6, which demonstrate that different values of α have little effect on resource utilization. Furthermore, we found that when α is set as 1.0, the model performance achieves good results on resource utilization, running time, and waiting time. Considering that resource utilization, running time, and waiting time are equally important, we set α as 1.0.

5.4. Performance comparison (RQ2)

The following four approaches are selected for comparison with our proposed RLPNet. The multi-level feedback queue scheduling algorithm is a typical CPU processor scheduling algorithm adopted by the UNIX operating system. It has the advantage of expediting the response to high-priority jobs and the completion of short jobs (processes). Since the problem discussed in this paper is also essentially a task scheduling problem, we used two multi-level feedback queueing scheduling algorithms, FCFS and HRRN, for comparison. Two latest machine learning-based approaches, OnPQ and OnDisc, were also leveraged for comparison.

- FCFS [39] (First-Come, First-Served): Service requests are executed according to the order they arrive at the edge server. That is, service requests that come first are executed first, and those that arrive later are executed later.
- HRRN [40] (High Response Ratio Next): Considering the tasks with longer waiting time and tasks with shorter running time, the higher the response ratio, the first it will be executed.
- OnPQ [34]: The task scheduling process is modeled as a Markov decision process. The Q-Learning algorithm is used to reduce energy consumption and average waiting time of service requests.
- OnDisc [41]: A weighted response time represents its latency sensitivity. The weighted response time is divided by the time required to run, which is defined as the residual density of the service request. During the scheduling process, the principle of highest residual density first (HRDF) is followed.

The numbers of service requests (n) and edge servers (m) are two variables in experimental design. According to the experimental results, it can be seen that whether n or m is changed, RLPNet performs better than other comparison algorithms in terms of resource utilization, running time, and waiting time. Because OnDisc aims to minimize weighted response time, the performance on the latency is often better than other algorithms except for RLPNet. However, the performance of OnDisc on other objectives is relatively not outstanding. Since OnPQ needs to combine multiple service requests into one batch and then hand it over to the edge server for execution, the next batch is allowed to be executed only after one batch is wholly executed. Since the execution time required by different service requests is different, an execution strategy will result in more idle time inside an edge server, leading to the unsatisfactory performance of OnPQ on each optimization objective. For HRRN and FCFS, two traditional job scheduling algorithms within operating systems, their performance is quite satisfactory and within the expected range. HRRN is slightly better than FCFS in terms of waiting time due to the consideration of the response ratio.

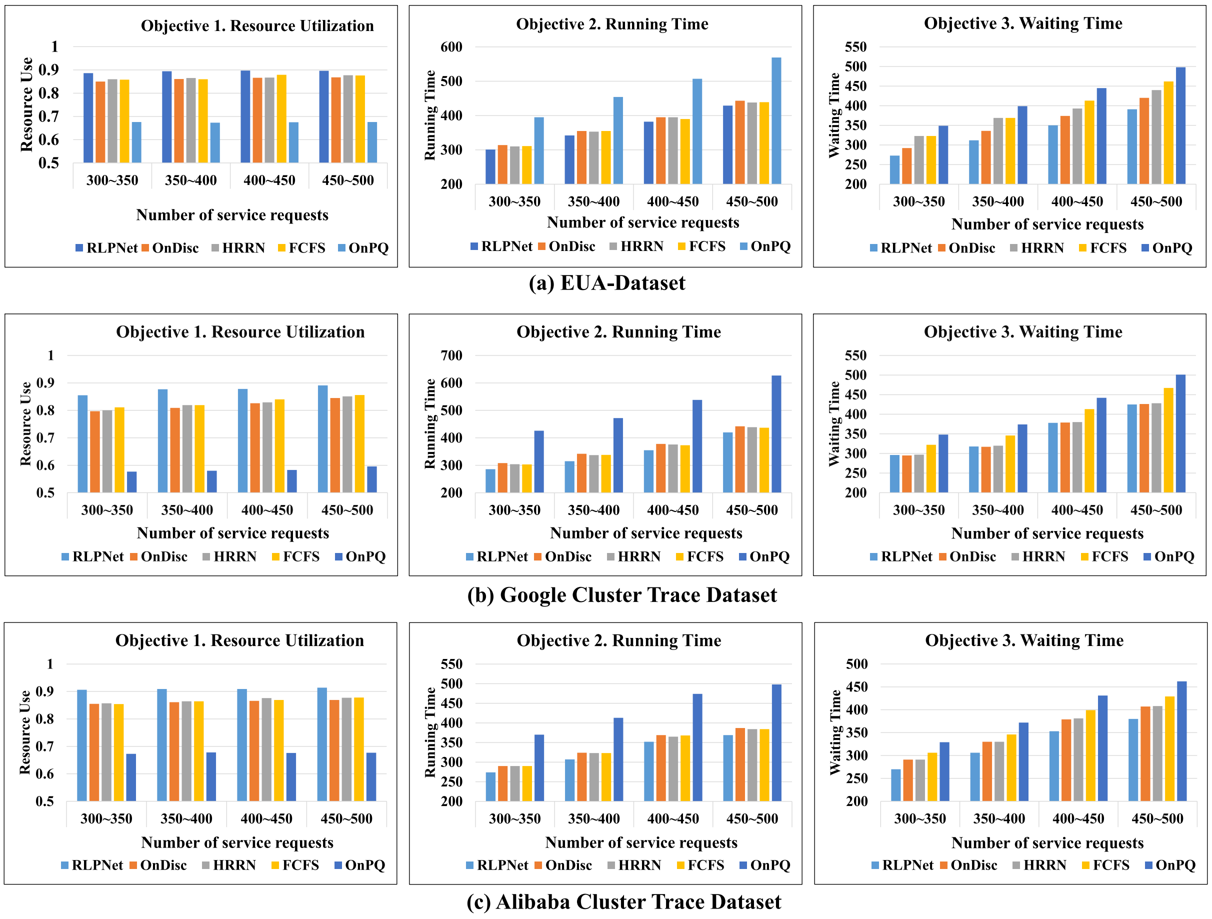

Fig. 7. Impact analysis of the service request number on (a) EUA-Dataset, (b) Google Cluster Trace Dataset and (c) Alibaba Cluster Trace Dataset.

5.4.1. *Impact analysis of the service request number*

The first group of experiments fixed the number of edge servers (m) as five and set four intervals according to the service request quantity: (300∼350), (350∼400), (400∼450), and (450∼500). In each interval, an experiment was conducted for every five additional service requests, and the experimental observations were averaged as the result of this interval. We tested the performance of different algorithms under various optimization objectives over three datasets. Experimental results are shown in Fig. 7. With the increase in service request quantity, the proposed RLPNet achieves almost the best results in all three evaluation metrics over three datasets.

Resource utilization analysis: We can see that the server resource utilization of these methods is getting higher and higher as the number of service requests increases. Our proposed RLPNet achieves the highest resource utilization with 88.5% on EUA-Dataset, 85.6% on Google Cluster Trace, and 91% on Alibaba Cluster Trace. Among them, it is the best performer on the Alibaba Cluster Trace with more than 90%, which is 12.5% higher than OnDisc, the best comparison method.

Running time analysis: The running time of several methods shows an increasing trend as the number of service requests increases, indicating that the more the service requests, the more time it takes for the server to process. Our proposed RLPNet takes the shortest time on average over all three datasets. Particularly, RLPNet performs the best on the Alibaba Cluster Trace and improves nearly 10% over the comparison methods.

Waiting time analysis: The average waiting time for service requests is getting longer as the number of service requests increases. The reason is that the resources of the edge server are limited, and the probability of thread blocking increases as the number of service requests increases. The proposed RLPNet spends the least average waiting time on three datasets. Notably, RLPNet performs the best on EUA-Dataset and Alibaba Cluster Trace. However, the advantage is not apparent on the Google Cluster Trace, probably because the server configuration used for the Google Cluster Trace is higher than the environment in which the other two datasets are running.

By continuously and dynamically reordering the service requests, RLPNet can reorder the tasks executed at each moment, improving the server's efficiency and reducing the cost. This result supports the conjecture that RLPNet can solve the scheduling problem through pointer networks and reinforcement learning. Furthermore, as the number of service requests increases, the performance of RLPNet performs better than other methods, making the average waiting time for service requests shorter.

5.4.2. *Impact analysis of the number of edge servers*

The second group of experiments controls the number of service requests (n) as 500 and changes the number of edge servers $m=$ 5, 7, 9, 11, 13, 15. We tested the performance of different algorithms under various optimization objectives. The experimental results are shown in Fig. 8. As the number of edge servers increases, the proposed RLPNet can consistently achieve optimal results in all three evaluation metrics over three different datasets.

Resource utilization analysis: When the number of service requests is fixed, the resource utilization of the edge servers gradually decreases as the number of edge servers increases. Even so, our proposed RLPNet method still has the highest resource

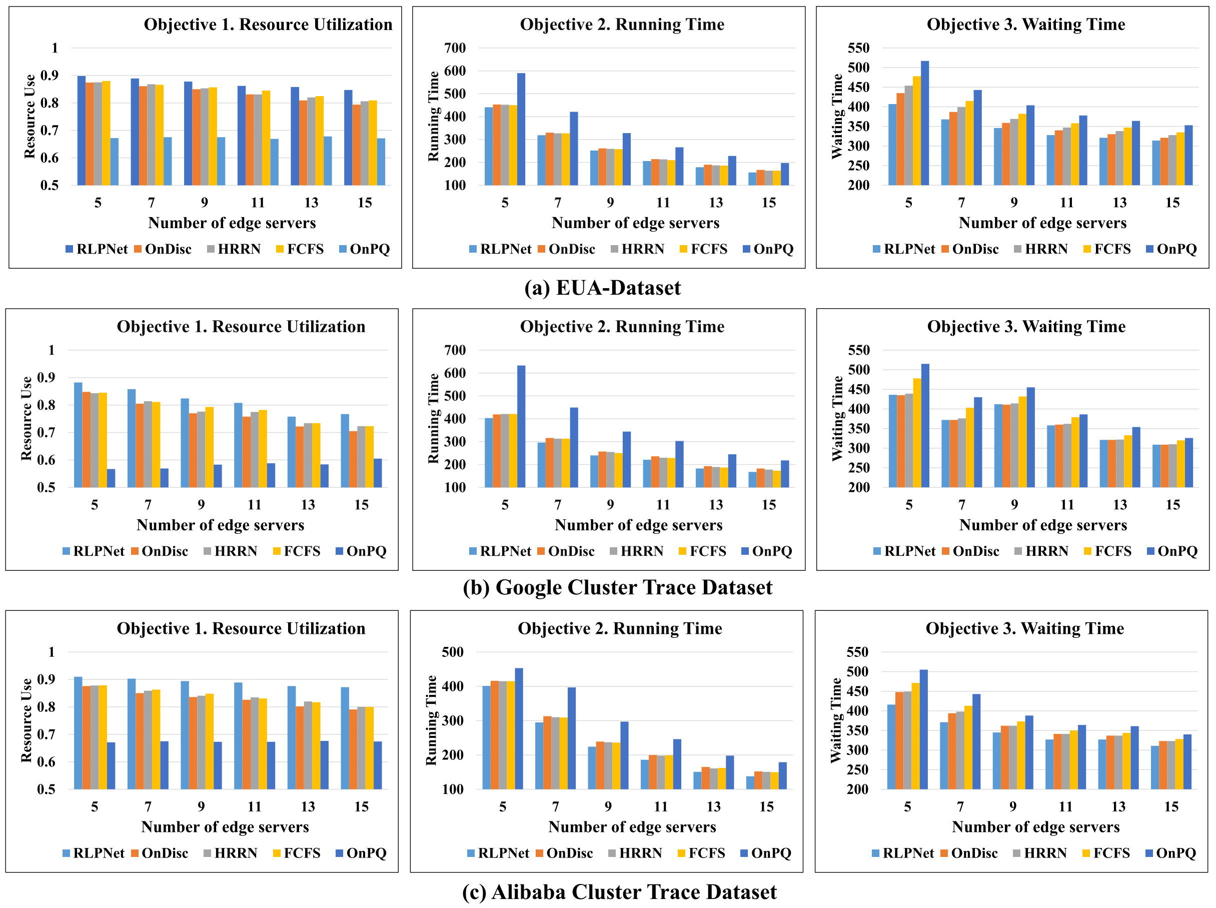

Fig. 8. The impact of different number of edge servers on (a) EUA-Dataset, (b) Google Cluster Trace Dataset and (c) Alibaba Cluster Trace Dataset.

utilization on all three different datasets. For example, when the number of edge servers is only five, the resource utilization of RLPNet is around 90%, while the best of the other methods is 87.5%. When the number of edge servers increases to 15, the resource utilization of RLPNet remains at about 80%.

Running time analysis: When the number of service requests is fixed, the running time of service requests keeps decreasing as the number of edge servers increases, as shown in Fig. 8. As the number of servers increases, RLPNet outperforms other methods on all three datasets. For example, when the number of edge servers is five, the average waiting time of RLPNet is about 400 ms. RLPNet takes more than 400 ms on the EUA-Dataset dataset, which is slightly inferior to the other two datasets. The possible reason is that the data volume of the EUA-Dataset is smaller, and RLPNet is more suitable to handle the large-scale service request scheduling problem.

Waiting time analysis: When the number of service requests is fixed, the average waiting time for service requests decreases as the number of edge servers increases. As the number of servers increases, RLPNet performs the best on both EUA-Dataset and Alibaba Cluster Trace. When the number of edge servers is five, the average waiting time of RLPNet is 400 ms. When the number of servers is expanded to 15, the average waiting time of RLPNet decreases significantly.

To summarize, when the number of servers increases, service requests can be more efficiently assigned to edge servers that are more suitable for execution, which allows the average waiting time to be significantly reduced. The performance improvement of RLPNet is more apparent when the number of servers increases, indicating that the RLPNet approach can adapt to large-scale service request scenarios. Also, when the resources of edge servers are limited, RLPNet can execute tasks more efficiently.

5.5. Comparison with Meta-heuristic algorithms (RQ3)

As mentioned above, the service request scheduling problem in edge computing environments is a combinatorial optimization problem with multiple constraints and variables. Meta-heuristic algorithms are widely used to solve this type of combinatorial optimization problem. To verify the effectiveness of our method, we select two classical meta-heuristic algorithms for comparison.

Comparison algorithms:

- PSO [18] (Particle Swarm Optimization) aims to find an optimal solution through cooperation and information sharing among individuals in the group. Since PSO can only perform single-objective optimization, we use the same weight combination as our approach selected in Table 1.
- NSGA-II [19]: An improved genetic algorithm for multi-objective optimization. NSGA-II uses fast non-dominated sorting with an elite strategy to reduce time complexity.

Meta-heuristic algorithm settings: For PSO, we used 50 particles and 100 iterations. For NSGA-II, we set the population size and the iteration number as 50 and 300, respectively.

Experimental results and analysis: As shown in Table 2, with the same experimental settings, RLPNet is only a little ahead of the meta-heuristic algorithms regarding the effectiveness of the three optimization objectives. However, the advantage of the running time in RLPNet is huge. In the face of complex and variable user requests and resource-constrained edge environments,

Table 2
Comparison between RLPNet and meta-heuristic algorithms.

Dataset	Algorithm	Resource utilization	Running time (ms)	Waiting time (ms)	Algorithm running time (ms)
EUA Dataset	RLPNet	**0.900**	**357**	**373**	**110**
	PSO [18]	0.889	360	377	140,000
	NSGA-II [19]	0.867	371	389	101,000
Google cluster trace	RLPNet	**0.882**	**326**	**368**	**100**
	PSO [18]	0.859	335	370	124,000
	NSGA-II [19]	0.833	345	386	98,000
Alibaba cluster trace	RLPNet	**0.901**	**334**	**359**	**80**
	PSO [18]	0.889	339	361	118,000
	NSGA-II [19]	0.868	346	373	76,000

meta-heuristic algorithms are significantly worse than RLPNet in terms of comprehensive execution efficiency, demonstrating our proposed method's effectiveness.

RLPNet deployed in the edge environment is pre-trained offline. In this way, the decision time of RLPNet can often be controlled within 1 s or even 0.1 s each time. Compared with the long iteration time needed by meta-heuristic algorithms, RLPNet can better meet the requirements of high latency sensitivity under edge computing.

6. Conclusion and future work

Service request scheduling, aiming to sufficiently utilize service resources at the edge to satisfy as many service requests as possible, has become a challenging topic in edge computing in recent years. Toward sthe issue, this paper comprehensively considers the indicators that need to be optimized for a reasonable scheduling strategy from the perspective of servers and users. We focus on scheduling the execution order of queued service requests by optimizing three objectives: resource utilization, running time, and waiting time. We model the scheduling problem as a sequential problem and use a reinforcement learning model with pointer networks to address the issue. We conduct experiments with three representative real-world datasets, which show that our proposed approach outperforms several state-of-the-art methods over three metrics.

In the future, we plan to improve the paper in three aspects. First, the limitations of the weighted sum as a multi-objective optimization problem will be considered, and the reward calculation strategy will be improved to further improve the training effect. Secondly, the scheduling strategy proposed in this paper is designed only for the internal scheduling of edge servers, and does not consider the coordination and loading balance among servers. When the edge servers cross, service requests sent by users located at the range of multiple edge servers are randomly selected. We plan to consider the user's selection strategy for edge servers by balancing server load. Thirdly, the current scheduling strategy is static scheduling, since an edge server takes the queued service requests for the next scheduling after the current one. We will investigate the dynamic scheduling while retaining static scheduling.

CRediT authorship contribution statement

Yuqi Zhao: Methodology, Software, Writing – original draft. **Bing Li:** Conceptualization, Writing – review & editing, Supervision. **Jian Wang:** Conceptualization, Methodology, Writing – review & editing. **Delun Jiang:** Conceptualization, Writing – review & editing. **Duantengchuan Li:** Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Nos. 62032016 and 61972292) and the Key Research and Development Program of Hubei Province (No. 2021BAA031).

References

[1] L. Yuan, Q. He, F. Chen, J. Zhang, L. Qi, X. Xu, Y. Xiang, Y. Yang, CSEdge: Enabling collaborative edge storage for multi-access edge computing based on blockchain, IEEE Trans. Parallel Distrib. Syst. 33 (8) (2022) 1873–1887.
[2] J. Deng, B. Li, J. Wang, Y. Zhao, Microservice pre-deployment based on mobility prediction and service composition in edge, in: 2021 IEEE International Conference on Web Services, ICWS, 2021, pp. 569–578.
[3] Q. Peng, C. Wu, Y. Xia, Y. Ma, X. Wang, N. Jiang, DoSRA: A decentralized approach to online edge task scheduling and resource allocation, IEEE Internet Things J. 9 (6) (2022) 4677–4692.
[4] X. Xia, H. Qiu, X. Xu, Y. Zhang, Multi-objective workflow scheduling based on genetic algorithm in cloud environment, Inform. Sci. 606 (2022) 38–59.
[5] S. Song, S. Ma, J. Zhao, F. Yang, L. Zhai, Cost-efficient multi-service task offloading scheduling for mobile edge computing, Appl. Intell. 52 (4) (2022) 4028–4040.
[6] S. Wang, W. Wang, Z. Jia, C. Pang, Flexible task scheduling based on edge computing and cloud collaboration, Comput. Syst. Sci. Eng. 42 (3) (2022) 1241–1255.
[7] H. Liao, X. Li, D. Guo, W. Kang, J. Li, Dependency-aware application assigning and scheduling in edge computing, IEEE Internet Things J. 9 (6) (2022) 4451–4463.
[8] J. Sun, L. Yin, M. Zou, Y. Zhang, T. Zhang, J. Zhou, Makespan-minimization workflow scheduling for complex networks with social groups in edge computing, J. Syst. Archit. 108 (2020) 101799.
[9] X. Zhao, X. Guo, Y. Zhang, W. Li, A parallel-batch multi-objective job scheduling algorithm in edge computing, in: IEEE International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2018, pp. 510–516.
[10] S. Tuli, S. Ilager, K. Ramamohanarao, R. Buyya, Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks, IEEE Trans. Mob. Comput. 21 (3) (2022) 940–954.
[11] S. Cai, D. Wang, H. Wang, Y. Lyu, G. Xu, X. Zheng, A.V. Vasilakos, Dynacomm: Accelerating distributed CNN training between edges and clouds through dynamic communication scheduling, IEEE J. Sel. Areas Commun. 40 (2) (2022) 611–625.
[12] A.S. Tanenbaum, Modern Operating Systems, third ed., Pearson Prentice-Hall, 2009.
[13] W. Stallings, Operating Systems - Internals and Design Principles, seventh ed., Pitman, 2011.
[14] J. Zou, T. Hao, C. Yu, H. Jin, A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario, IEEE Trans. Comput. 70 (2) (2021) 228–239.
[15] T. Zheng, J. Wan, J. Zhang, C. Jiang, Deep reinforcement learning-based workload scheduling for edge computing, J. Cloud Comput. 11 (2022) 3.
[16] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, in: Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems, 2015, pp. 2692–2700.
[17] N. Goren, E. Fogel, D. Halperin, Area optimal polygonization using simulated annealing, ACM J. Exp. Algorithmics 27 (2022) 2.3:1–2.3:17.

[18] T. Li, J. Shi, W. Deng, Z. Hu, Pyramid particle swarm optimization with novel strategies of competition and cooperation, Appl. Soft Comput. 121 (2022) 108731.
[19] W. Deng, X. Zhang, Y. Zhou, Y. Liu, X. Zhou, H. Chen, H. Zhao, An enhanced fast non-dominated solution sorting genetic algorithm for multi-objective problems, Inform. Sci. 585 (2022) 441–453.
[20] G. Wang, C. Li, Y. Huang, X. Wang, Y. Luo, Smart contract-based caching and data transaction optimization in mobile edge computing, Knowl.-Based Syst. (2022) 109344.
[21] L. Zhu, J. Lin, Y.-Y. Li, Z.-J. Wang, A decomposition-based multi-objective genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem, Knowl.-Based Syst. 225 (2021) 107099.
[22] X. Kong, G. Duan, M. Hou, G. Shen, H. Wang, X. Yan, M. Collotta, Deep reinforcement learning-based energy-efficient edge computing for internet of vehicles, IEEE Trans. Ind. Inform. 18 (9) (2022) 6308–6316.
[23] F. Song, H. Xing, X. Wang, S. Luo, P. Dai, K. Li, Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach, Future Gener. Comput. Syst. 128 (2022) 333–348.
[24] M. Huang, H. Li, B. Bai, C. Wang, K. Bai, F. Wang, A federated multi-view deep learning framework for privacy-preserving recommendations, 2020, CoRR abs/2008.10808.
[25] H. Wang, M. Yurochkin, Y. Sun, D.S. Papailiopoulos, Y. Khazaeni, Federated learning with matched averaging, in: 8th International Conference on Learning Representations, ICLR, 2020.
[26] Z. Hu, K. Shaloudegi, G. Zhang, Y. Yu, FedMGDA+: Federated learning meets multi-objective optimization, 2020, CoRR abs/2006.11489.
[27] P. Lai, Q. He, M. Abdelrazek, F. Chen, J.G. Hosking, J.C. Grundy, Y. Yang, Optimal edge user allocation in edge computing with variable sized vector bin packing, in: C. Pahl, M. Vukovic, J. Yin, Q. Yu (Eds.), Service-Oriented Computing - 16th International Conference, ICSOC, Vol. 11236, 2018, pp. 230–245.
[28] H. Falsafi, A. Zakariazadeh, S. Jadid, The role of demand response in single and multi-objective wind-thermal generation scheduling: A stochastic programming, Energy 64 (2014) 853–867.
[29] Y. Cui, Z. Geng, Q. Zhu, Y. Han, Review: Multi-objective optimization methods and application in energy saving, Energy 125 (2017) 681–704.
[30] J. Schaffer, Multiple objective optimization with vector evaluated genetic algorithms, in: G.J.E. Grefensette, J.J.L. Erlbraum (Eds.), Proceedings of the First Int. Conference on Genetic Algortihms, 1985, pp. 93–100.
[31] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems, 2014, pp. 3104–3112.
[32] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, 2017.
[33] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (1992) 229–256.
[34] J. Lu, X. Guo, X. Zhao, H. Zhou, A parallel tasks scheduling algorithm with Markov decision process in edge computing, in: Green, Pervasive, and Cloud Computing - 15th International Conference, GPC, Vol. 12398, 2020, pp. 362–375.
[35] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, X. Liu, From cloud to edge: a first look at public edge platforms, in: IMC '21: ACM Internet Measurement Conference, Virtual Event, USA, 2021, pp. 37–53.
[36] P.C. Hansen, Analysis of discrete ill-posed problems by means of the L-curve, SIAM Rev. 34 (4) (1992) 561–580.
[37] G.H. Golub, M. Heath, G. Wahba, Generalized cross-validation as a method for choosing a good ridge parameter, in: R.H. Chan, C. Greif, D.P. O'Leary (Eds.), Milestones in Matrix Computation - Selected Works of Gene H. Golub, with Commentaries, Oxford University Press, 2007, pp. 202–212.
[38] H.W. Engl, Discrepancy principles for Tikhonov regularization of ill-posed problems leading to optimal convergence rates, J. Optim. Theory Appl. 52 (2) (1987) 209–215.
[39] R. Chen, L. Cui, Y. Zhang, J. Chen, K. Yao, Y. Yang, C. Yao, H. Han, Delay optimization with FCFS queuing model in mobile edge computing-assisted UAV swarms: A game-theoretic learning approach, in: 2020 International Conference on Wireless Communications and Signal Processing (WCSP), 2020, pp. 245–250.
[40] L. Zeng, J. Sun, J. Ma, Q. Liu, Task scheduling based on multi-level hashing and HRRN in cloud computing, in: IEEE Intl Conf on Dependable, Autonomic and Secure Computing, 2021, pp. 667–672.
[41] H. Tan, Z. Han, X. Li, F.C.M. Lau, Online job dispatching and scheduling in edge-clouds, in: 2017 IEEE Conference on Computer Communications, INFOCOM, 2017, pp. 1–9.