TEDC: Temporal-aware Edge Data Caching with Specified Latency Preference

Guobing Zou[1], Ya Liu[2, 1*], Song Yang[1], Shengxiang Hu[1], Yanglan Gan[3*], Bofeng Zhang[4]
[1]School of Computer Engineering and Science, Shanghai University, Shanghai, China
[2]Department of Computer Science and Technology, Tongji University, Shanghai, China
[3]School of Computer Science and Technology, Donghua University, Shanghai, China
[4]School of Computer and Information Engineering, Shanghai Polytechnic University, Shanghai, China
{gbzou, yangsong, shengxianghu}@shu.edu.cn, yaliu0227@tongji.edu.cn, ylgan@dhu.edu.cn, bfzhang@sspu.edu.cn

***Abstract*—Recently, the edge data caching (EDC) problem has received much attention. It aims to appropriately cache data on edge servers. Existing EDC approaches suffer from a series of limitations. First, they often overlook the diverse characteristics of data, including caching costs and latency preferences. In reality, different types of data vary in size and require different storage resources for caching. The impact of specified latency preferences of edge users for different data on the quality of experience should be considered in the EDC problem. Second, the temporal dynamics of edge users' data requests and distributions have been insufficiently addressed. To overcome these limitations systematically, this paper focuses on the problem of temporal-aware edge data caching with specified latency preference (TEDC). We first formulate the TEDC problem and transform it into an optimization problem with multiple objectives and global constraints and prove its $\mathcal{NP}$-hardness. Then, we propose an optimal approach named TEDC-IP to solve this TEDC problem with the Integer Programming technique and a heuristic algorithm named TEDC-A for finding approximate solutions to large-scale TEDC problems efficiently. Extensive experiments are conducted on two widely-used real-world datasets to evaluate the performance of our approach. The results demonstrate that TEDC-IP and TEDC-A significantly outperform state-of-the-art approaches in finding approximate solutions in terms of the trade-off among multiple metrics.**

***Index Terms*—Edge Data Caching, Latency Preferences, Temporal Dynamics, Cache Cost, Data Request**

I. Introduction

With pervasive mobile computing and the Internet of Things, it has witnessed the development of many new compute-intensive and latency-sensitive applications, such as cognitive assistance, mobile gaming, and virtual/augmented reality (VR/AR) [1], [2]. Edge computing has emerged as a novel distributed computing paradigm, extending the capabilities of the traditional cloud computing model. By deploying edge servers at the network edge, closer to users' geographic locations, various resources such as CPU, RAM, storage, and bandwidth can be provided. This paradigm is crucial for enabling the advancement of 5G mobile networks [3] [4]. Service providers can rent computing resources on edge servers and host their applications, ensuring low latency for app users [5]–[7].

In the mobile edge computing environment, every time a user requests social content or video service, the request needs to be sent to the remote cloud center through the edge server to perform data transmission. This process causes tremendous pressure on the network and triggers a sharp decline in user experience. To enhance service efficiency and performance, edge services are deployed on edge servers, providing users with various services such as content delivery, real-time data processing, video stream analysis, and more [8], [9]. Through edge data caching [10], [11], edge services can access the required data more quickly, thereby enhancing service response speed and performance. This is particularly crucial for services such as real-time data processing and content delivery, which necessitate swift handling and transmission of large volumes of data. Furthermore, edge data caching reduces the data transfer volume between edge services and the cloud, lowering network bandwidth usage and data transmission costs, thus improving overall system efficiency.

Edge data caching, as a key technology in future mobile communication systems, has attracted a lot of attention in academia and industry [12]–[15]. Many researchers have investigated network cache from different perspectives, e.g., content popularity, cache allocation and replacement strategies, and social sense cache [16]–[18]. Halalai et al. [19] proposed erasure code into data caching techniques to find the optimal solution to cache data chunks, considering the data popularity and network latency. In [20], Drolia et al. provided a coordinating mechanism for minimizing data retrieval latency, balancing the workloads between edge servers and the remote cloud server. However, existing approaches suffer from a series of limitations. Firstly, they often overlook edge users' specified latency preferences for different types of data and fail to consider the varying cache costs associated with different data. Many approaches assume a uniform cache cost for all data types. In addition, they oversight the relationship between network latency and user experience for different data. For example, people have very high latency requirements for autonomous driving, whereas the network latency requirements for entertainment video are much lower. Secondly, most approaches oversimplify the edge data caching problem by modeling it as a static global optimization challenge. However,

[*]Corresponding author.

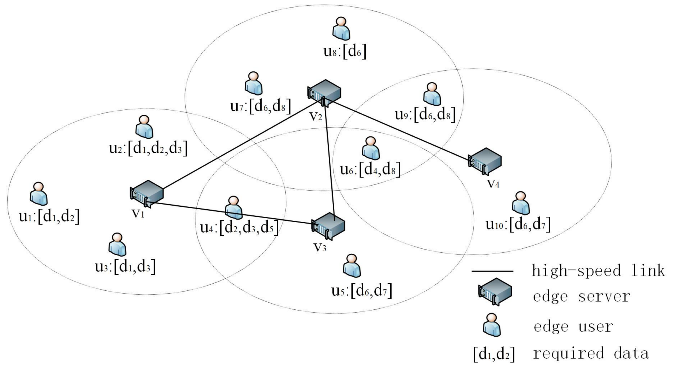


Fig. 1: A motivating example of the TEDC problem.

in reality, the distribution of edge users and their data requests change over time, necessitating consideration in the context of the EDC problem.

In real-world scenarios, from the service provider's perspective, the cost-effective edge data caching problem aims to minimize the total user latency and minimize the data caching cost. Taking the above issues into consideration, the problem is referred to as the temporal-aware edge data caching problem with specified latency preferences (TEDC). In addition to server capacity constraint, server coverage constraint, and server adjacency constraint, we take into account the temporal dynamics and data characteristics including storage resource and latency preference. The main contributions of this paper are summarized as follows:

- We model and formulate the TEDC problem as a constrained optimization problem from the app vendor's perspective, and prove its $\mathcal{NP}$-hardness.
- We develop an optimal approach, namely TEDC-IP, for solving the TEDC problem exactly with the Integer Programming technique. Meanwhile, an approximate approach named TEDC-A for finding solutions to large-scale TEDC problems efficiently.
- Extensive experiments are conducted on two widely used real-world datasets to evaluate the performance of TEDC-IP and TEDC-A against five representative approaches. The results demonstrate the superior performance of the proposed two algorithms on multiple evaluation metrics.

The remainder of the paper is organized as follows. Section II provides a motivating example for this research. Section III formulates the TEDC problem. Section IV proposes the optimal approach TEDC-IP and proves the $\mathcal{NP}$-hardness of the TEDC problem, then proposes the approximation approach TEDC-A for solving TEDC problems in large-scale scenarios. Section V evaluates the proposed approaches based on extensive experiments. Section VI reviews the related work. Section VII concludes the paper and points out future work.

II. MOTIVATING EXAMPLE

Fig. 1 illustrates a simplified scenario of the edge data caching problem. There are four edge servers, i.e., $\{v_1, v_2, v_3, v_4\}$, each covering a specific geographic area. Adjacent edge servers' coverage usually intersects to avoid blank areas not covered by any edge server. Ten users are denoted by $\{u_1, u_2, u_3, \ldots, u_{10}\}$, and eight popular data are denoted by $\{d_1, d_2, \ldots, d_8\}$. Each user has a list of data requests in a time slot. Due to the server coverage constraint and the server adjacency constraint, an app user can retrieve a piece of app data from its local edge servers or neighbor edge servers if the app data is cached on these edge servers. From the service providers' perspective, caching all popular data on every edge server in the area can easily accommodate all users in the area. However, this is not cost-effective nor practical due to the server capacity constraint. Assume the server adjacency constraint is one hop. This allows a user to access any edge servers within 1 hop over the edge server graph for cached app data. Otherwise, it will have to be retrieved from the service provider's remote cloud center. For example, u_{10} can only retrieve cached data from either local edge server v_4 or neighbor edge server v_2. Apparently, multiple data caching strategies fulfill all three constraints.

TABLE I: Latency preferences and cache cost of different popular data.

$Data$	$latency\ preferences$	$cache\ cost$
d_1	1	2
d_2	3	4
d_3	0	3
d_4	2	5
d_5	1	3
$\ldots$	$\ldots$	$\ldots$

Data Cache Cost and Latency Preferences. Existing studies neglect the different characteristics of data, including data cache cost and latency preferences. In real-world scenarios, different popular data will consume different storage resources when caching them on edge servers, which is an important factor that service providers should consider. Assume the cache cost and latency preference of each data is shown in Table I. For example, assume that edge server v_1 has 8 storage resources. Since the total storage resources required by data d_1 and d_2 is 6, it does not exceed the remaining storage resources on v_1. Assume that edge server v_1 caches data d_1, d_3 and d_5, which also does not exceed the remaining storage resources of v_1. As a result, we can find that caching data d_1, d_3 and d_5 on edge server v_1 will meet more data requests for edge users than caching data d_1, d_2. Additionally, in reality, edge users have varying requirements for response latency depending on the type of data. For instance, autonomous driving systems are highly sensitive to network delays. Any delay in data transmission could potentially endanger the driver's life and safety. Therefore, the latency requirement for data response is dictated by the nature of the service itself, directly impacting user satisfaction that app vendors are concerned about.

Temporal Dynamics. Existing studies treat an EDC problem as a static global optimization problem and aim at finding an optimal or near-optimal solution. However, they are ineffective in dynamic EDC scenarios because they cannot handle the variations in the distribution of users and their data requests

TABLE II: Variations on the data requests of users at different time slots.

U \ T	t_1	t_2	t_3	...
u_1	$\{d_1, d_4, d_7\}$	$\{d_1, d_4\}$	$\{d_1, d_2, d_4\}$	...
u_2	$\{d_3, d_4, d_8\}$	$\{d_3, d_4\}$	$\{d_4, d_8\}$	...
u_3	$\{d_1, d_5\}$	$\{d_1, d_5\}$	$\{d_1, d_3, d_5\}$	...
u_4	$\{d_2, d_4, d_6, d_7\}$	$\{d_4, d_6, d_7\}$	/	...
u_5	/	/	$\{d_5, d_8\}$	...
...	...	...	...	...

TABLE III: Notations

Notation	Description
$T = \{t_1, t_2, \ldots, t_p\}$	a set of time slots
$D_t = \{d_1, d_2, \ldots, d_l\}$	a set of data to be cached in time slot t
d_k	the $k-$th data in D_t.
$V = \{v_1, v_2, \ldots, v_m\}$	a set of edge servers
v_j	the $j-$th edge server in V
$U_t = \{u_1, u_2, \ldots, u_n\}$	a set of edge users in time slot t
u_i	the $i-$th user in U_t
$D_t(u_i)$	a set of data which a user u_i requests in time slot t
$l(d_k)$	latency preference for a data d_k
$c(d_k)$	data caching cost (required storage resources) for a data d_k
$a(v_j)$	available capacity of an edge server v_j
$l_{i,j}$	latency from server v_i to v_j
R_t	a collection of binary variables $r_{d_k}^j$ representing the data caching strategy at time t
$r_{d_k}^j$	a boolean indicator of whether d_k is cached on v_j
$l_{u_i}^{d_k}$	the data delay experienced by user u_i to get data d_k
$\theta_{i,k}$	a boolean indicator of whether user u_i access data d_k with a limit latency
G	a graph of a particular geographical area
$E = \{e_1, e_2, \ldots, e_q\}$	a finite set of links between edge servers

over time. Table II demonstrates the data requests submitted by edge users, for processing across three time slots, i.e., t_1, t_2, and t_3. Take u_1 for example. It submits three data requests $\{d_1, d_4, d_7\}$ in time slot t_1, two service requests $\{d_1, d_4\}$ in t_2 and three service requests $\{d_1, d_2, d_4\}$ in t_3. Given the temporal variation in users' distribution and data requests, it is essential to consider these factors in the EDC problem to achieve the optimization objectives of the service provider.

III. Problem Formulation

In this section, we first define the concept of multi-data caching and the server capacity constraint. Then, we model data retrieval latency and latency preferences. Finally, we formally model the TEDC problem. Table III summarizes the important notations and descriptions used in this paper.

A. Multi-data Caching and Capacity Constraint

In this study, we conceptualize the edge server network within a specific geographical area as a graph, denoted as $G(V, E)$, where $V = \{v_1, v_2, \ldots, v_m\}$ represents the set of nodes and $E = \{e_1, e_2, \ldots, e_q\}$ represents the set of edges within the graph. Each node $v_i \in V$ signifies an edge server, while each edge $e_q \in E$ denotes a connection between two nodes within the graph. Throughout the remainder of this paper, we will use the terms edge server and node interchangeably, both indicated by v.

Definition 1 ***(Multi-data Caching)***. Assumed that in time slot t, a finite set of data $D_t = \{d_1, d_2, \ldots, d_l\}$ is to be cached on edge servers. A multi-data caching strategy is represented as $R_t = \{\langle r_{d_1}^1, \ldots, r_{d_1}^m \rangle, \ldots, \langle r_{d_l}^1, \ldots, r_{d_l}^m \rangle\}$, where $r_{d_k}^j \in \{0, 1\}$ denotes whether data d_k is cached on edge server v_j.

$$r_{d_k}^j = \begin{cases} 1, & \text{if } d_k \text{ is cached on edge server } v_j \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

In edge computing, many service providers may need to hire storage capacities on edge servers in the same area for caching their data. This causes fierce competition among service providers and makes it practically impossible for every service provider to cache all app data on every edge server. Furthermore, the storage resources on an edge server are usually limited. Thus, the storage resources of data cached on each edge server v_j cannot violate its available server capacity constraint in any time slot t.

Definition 2 ***(Capacity Constraint)***. The storage resources on an edge server are usually limited. The total amount of resources required by all data cached on an edge server must not exceed its capacity available in time slot t:

$$\sum_{k=1}^{|D_t|} c(d_k) * r_{d_k}^j \le a(v_j), \forall j \in \{1, 2, \ldots, m\} \tag{2}$$

Take Fig. 1 for example. Since the storage resources required by data d_1, d_2, d_3 and d_5 are 2, 4, 3, and 3, respectively, the total amount of resources required by d_1 and d_2 is 6. It does not exceed v_1's available capacity, which is 8. However, if we continue to cache data d_3 or d_5 on v_1, v_1's capacity will not suffice to cache all of them because their total storage resource demand is 9, exceeding v_1's available resource capacity of 8.

B. Data Retrieval Latency and Latency Preferences

The data retrieval latency within the edge server network comprises two main components: the latency between the device and its nearby edge server, and the latency between the local edge server and neighboring edge servers. Given that the latency between the device and its nearby edge server is typically negligible in the 5G network and remains unaffected by the data caching strategy, it is excluded from the formulation of the caching strategy.

Definition 3 ***(Data Retrieval Latency)***. As the hop count between edge servers is a measure of data retrieval latency, the network delay in retrieving data d_k for the app user u_i in time slot t is calculated as follows:

$$\begin{gathered} l_{u_i}^{d_k} = min\left\{l_{i,j}, r_{d_k}^j = 1, v_j \in V\right\}, \\ \forall i \in \{1, 2, \ldots, n\}, d_k \in D_t(u_i) \end{gathered} \tag{3}$$

where v_j is the edge server caching data d_k and v_i is the edge server covering the edge user u_i, and $l_{i,j}$ is the number of hops between v_i and v_j.

Take Fig. 1 as an example. Assume edge server v_1 is selected to cache data d_5, the latency $l_{u_4}^{d_5}$ in user u_4's retrieval of data d_5 is 0, and the latency $l_{u_5}^{d_5}$ in user u_5's retrieval of data d_5 is 1.

Definition 4 ***(Latency Preferences)***. Given a finite set of data $D_t = \{d_1, d_2, \dots, d_l\}$ in time slot t, each type of data has a different data response latency preference $l(d_k)$. Once a user cannot retrieve data d_k from edge servers within $l(d_k)$, the user experience will be impacted significantly. Here, $\theta_{i,k}$ is denoted to indicate whether user u_i access data d_k within a limit latency:

$$\theta_{i,k} = \begin{cases} 1, & \text{if data retrieval latency } l_{u_i}^{d_k} \le l(d_k) \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

where $l_{u_i}^{d_k}$ is minimum data retrieval latency form edge servers, $l(d_k)$ is the data response latency limit of data d_k.

In the real world, popular app data have different requirements for data response latency. For instance, autonomous driving is particularly sensitive to data response latency. Once an overtime reaction occurs, it is likely to endanger the driver's life and safety. While data response latency requirements for entertainment video is much lower. Thus, the different data has different data response latency requirements, which is related to edge users' satisfaction.

C. TEDC Problem Formulation

Based on the above definition of multi-data caching, server capacity constraint, data retrieval latency, and latency preferences, the TEDC problem can be formally defined as follows:

Definition 5 ***(Temporal-aware Edge Data Caching with Specified Latency Preference)***. An TEDC problem can be defined as a five-tuple $TEDC =< T, U_t, V, D_t, E >$, where

(1) $T = \{t_1, t_2, \dots, t_p\}$ is a set of time slots;
(2) $U_t = \{u_1, u_2, ..., u_n\}$ is a set of users in time slot t and each user has a set of data requests;
(3) $V = \{v_1, v_2, ..., v_m\}$ is a set of edge servers;
(4) $D_t = \{d_1, d_2, \dots, d_l\}$ is a set of data in time slot t;
(5) $E = \{e_1, e_2, \dots, e_q\}$ is a finite set of links between edge servers.

From the service providers' perspective, the first optimization objective is to minimize the users' overall data retrieval latency produced by its data caching strategy R:

$$minimize \frac{1}{|T|} \sum_{t \in T} \sum_{i=1}^{|U_t|} \sum_{k=1}^{D_t(u_i)} l_{u_i}^{d_k} * \theta_{i,k} \tag{5}$$

the secondary objective is to minimize the total data caching cost incurred by R:

$$minimize \frac{1}{|T|} \sum_{t \in T} \sum_{j=1}^{|V|} \sum_{k=1}^{|D_t|} c(d_k) * r_{d_k}^{j} \tag{6}$$

IV. Approach

In this section, we first model the TEDC problem as a constrained optimization problem to find its optimal solution. Then, an approximation algorithm named TEDC-A is proposed for finding approximate solutions to large-scale TEDC problems efficiently.

A. Optimal Approach

Given a TEDC problem TEDC=$\langle T, U_t, V, D_t, E \rangle$, there are two optimization objectives: (1) minimizing the users' overall data retrieval latency, and (2) minimizing the data caching cost across multiple time slots in that area. In this section, we present an approach named TEDC-IP for finding the optimal solution to a TEDC problem. It model the TEDC problem as a Lexicographic Goal Programming (LGP) problem as follows:

$$minimize \frac{1}{|T|} \sum_{t \in T} \sum_{i=1}^{|U_t|} \sum_{k=1}^{D_t(u_i)} l_{u_i}^{d_k} * \theta_{i,k} \tag{7}$$

$$minimize \frac{1}{|T|} \sum_{t \in T} \sum_{j=1}^{|V|} \sum_{k=1}^{|D_t|} c(d_k) * r_{d_k}^{j} \tag{8}$$

s.t:

$$\sum_{k=1}^{|D_t|} c(d_k) * r_{d_k}^{j} \le a(v_j), \forall j \in \{1, 2, \dots, m\} \tag{9}$$

$$\begin{gathered} l_{u_i}^{d_k} = min\left\{ l_{i,j}, r_{d_k}^{j} = 1, v_j \in V \right\}, \\ \forall i \in \{1, 2, \dots, n\}, d_k \in D_t(u_i) \end{gathered} \tag{10}$$

where $r_{d_k}^{j}$ and $\theta_{i,k}$ are two binary variables indicating that,

$$r_{d_k}^{j} = \begin{cases} 1, & \text{if data } d_k \text{ is cached on } v_j \text{ in time t} \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

$$\theta_{i,k} = \begin{cases} 1, & \text{if data retrieval latency } l_{u_i}^{d_k} \le l(d_k) \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

The objective function (7) minimizes the users' overall data retrieval latency across all the time slots. The objective function (8) minimizes the data caching cost in multiple time slots. Constraint (9) makes sure that the aggregate storage resources required for the data cached on the edge server must not exceed its available upper bound capacity in time slot t. Constraint (10) ensures that each app user u_i retrieves data d_k from the nearest edge server where the data is cached in time slot t and calculates the network delay. There are two groups of binary variables, i.e., $r_{d_k}^{j}$ (11) and $\theta_{i,k}$ (12).

B. TEDC Hardness

Based on the modeled LGP optimization problem, we now prove that the TEDC problem is $\mathcal{NP}$-hard. First, we introduce the Knapsack problem, a classic $\mathcal{NP}$-hard problem.

Definition 6 ***(Knapsack Problem).*** Given n items with associated weights and values, the objective of the Knapsack problem is to choose a subset of items to maximize total value while adhering to the constraint of a maximum total weight W. This problem can be formally represented as follows:

$$\max \sum_{i=1}^{n} v_i x_i \tag{13}$$

s.t.:

$$\sum_{i=1}^{n} w_i x_i \leq W \tag{14}$$

$$x_i \in \{0, 1\} \tag{15}$$

Here, v_i and w_i represent the value and weight of item i, respectively. W denotes the total weight constraint for all selected items, and x_i is a binary variable indicating whether the i-th item is chosen.

Theorem 1. *The $\mathcal{NP}$-hard Knapsack is reducible from the TEDC problem, i.e., Knapsack $\leq_p$ TEDC. Thus, the TEDC problem is $\mathcal{NP}$-hard.*

Based on the definition of the Knapsack problem, we now prove that the TEDC problem is $\mathcal{NP}$-hard by reducing a Knapsack problem to a specialization of a TEDC problem. For ease of exposition, we make the following assumptions: 1) For each user u_i, its requirements for different data are the same, i.e., $D_t(u_1) = D_t(u_2) = \ldots = D_t(u_n)$; 2) For each type of data d_k, its latency preference is the same, i.e., $l_{d_1} = l_{d_2} = \ldots = l_{d_l}$; 3) For any edge server s_j, its storage capacity is equal, i.e., $a(v_1) = a(v_2) = \ldots = a(v_m)$; 4) The coverage of each edge server is infinite, i.e., an edge user can retrieval of any data from any of the edge servers in the area. 5) All edge servers can communicate with each other.

Based on the above assumptions, the objective function (7) and constraints (10)(12) can be omitted. Thus, there is no data retrieval latency and latency preferences in the TEDC problem. So, the hidden goal in the objective function (7) is to satisfy more user data requests, e.g., more users can get the cached data they want from the edge server, which can be defined as follows:

$$max \frac{1}{|T|} \sum_{t \in T} \sum_{i=1}^{|U_t|} \sum_{k=1}^{D_t(u_i)} \beta_{i,k} \tag{16}$$

where $\beta_{i,k}$ is a binary variables indicating that,

$$\beta_{i,k} = \begin{cases} 1, & \text{if } u_i \text{ retrieval } d_k \text{ from edge servers} \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

Since the coverage of each edge server is infinite, data acquisition always meets the delay limit, and the objective function (7) can mean maximizing the number of data requests satisfied. For the simplified special case, objective (16) can be projected to (13). Constraints (9)(11) can be combined and projected to (14)(15) since the storage capacities of all edge servers can be aggregated as an overall resource limit. Clearly, there is a solution to the TEDC problem if and only if there is a solution to the corresponding Knapsack problem. Thus, the TEDC problem is $\mathcal{NP}$-hard.

However, in a real-world TEDC problem, there are a set of time slots $T = \{t_1, t_2, \ldots, t_p\}$. In each time slot t, there is a set of users $U_t = \{u_1, u_2, \ldots, u_n\}$, where each user delivers a set of data requests that always dynamically changes as time goes by. Simultaneously, the different latency preferences for different app data are of vital importance to the user experience. Furthermore, different popular data caches on edge servers will consume different storage resources, which is an important factor that service providers should consider when caching data. Therefore, a TEDC problem is more complicated than a Knapsack problem, since all the parameters of a Knapsack problem are known except the optimization variables. Thus, it is a challenging task to solve a TEDC problem optimally in polynomial time.

Algorithm 1 TEDC-A algorithm

```
Input: An TEDC problem $\langle T, U_t, V, D_t, E \rangle$.
Output: An edge data caching strategy $f : d_k \rightarrow v_j$.
 1: $C \leftarrow \varnothing$
 2: $M\,[|V|]\,[|D_t|] = \{0\}$
 3: for each $c_j \in C$ do
 4:     av = copy($a(v_j)$)
 5:     $M \leftarrow$ UpdateServerData($c_j$) # Algorithm 2
 6:     $datas$ = GetBenefitData($M$) # Algorithm 3
 7:     $k = 0$
 8:     while av != 0 & data != null do
 9:         if datas[k].benefit $\leq \beta$ then
10:             break
11:         end if
12:         $c_j \leftarrow c_j \cup$ datas[k].data
13:         $av = av - a(d_k)$
14:         $k = k + 1$
15:     end while
16: end for
17: return the generated edge data caching strategy $f$
```

C. Approximation Algorithm

Since the TEDC problem is $\mathcal{NP}$-hard, finding the optimal solution is intractable in large-scale TEDC scenarios. This section presents an approximation algorithm, named TEDC-A, for finding approximate solutions to large-scale TEDC problems efficiently.

Given $V = \{v_1, v_2, \ldots, v_m\}$, $U_t = \{u_1, u_2, \ldots, u_n\}$ and $D_t = \{d_1, d_2, \ldots, d_l\}$ at time slot t, TEDC-A comprehensively considers the number of requests for various data in each edge server, the caching cost of different data, the data response latency requirement for each data, and the unique constraints in the edge computing environment, including the coverage and capacity constraint. Algorithm 1 shows the pseudo-code of TEDC-A, while the functions used in Algorithm 1 are presented in Algorithms 2 and 3.

In this algorithm, we select the appropriate data to be cached for each edge server. It starts with the initialization of an empty data caching strategy C and an empty matrix $M_{|V|*|D_t|}$ in lines 1-2, where the matrix M is used to store the result of function *UpdateServerData* and the value $m_{j,k}$ indicates the number of data requests of d_k in the coverage of edge server v_j. Then, TEDC-A always selects the candidate data with the

Algorithm 2 UpdateServerData

```
Input: an server $v_j$; edge users $U_t$; data requests $D_t(u_i)$
Output: a server-data Matrix $M$.
1: A $\leftarrow$ $U(v_j)$ # get the edge users for $v_j$, satisfying proximity constraints
2: for each $u_i \in A$ do
3:     for each $d_k \in D_t(u_i)$ do
4:         $M[v_j][d_k] \leftarrow M[v_j][d_k] + 1$
5:     end for
6: end for
7: return $M$
```

Algorithm 3 getBenefitData

```
Input: a server-data Matrix $M$; data cache cost $c(d_k)$; data latency preference $l_{d_k}$
Output: data.
1: data = null
2: i = 0
3: for each $d_k \in M[v_j]$ do
4:     datas[i].data = $d_k$
5:     datas[i].benefit = $M[v_j][d_k]/c(d_k)$
6:     datas[i].latency = $l_{d_k}$
7:     i = i+1
8: end for
9: length $\leftarrow$ $|datas|$
10: for i $\leftarrow$ 0 to length-1 do
11:     for j $\leftarrow$ 0 to length-1-i do
12:         if datas[i].benefit $<$ datas[j].benefit then
13:             swap(datas[i],datas[j])
14:         else if datas[i].benefit $=$ datas[j].benefit then
15:             if datas[i].latency $>$ datas[j].latency then
16:                 swap(datas[i],datas[j])
17:             end if
18:         end if
19:     end for
20: end for
21: return data
```

maximum benefit for the service provider to be cached in each edge server (line 3-16), and each $c_j \in C$ indicates which edge data is cached on an edge server v_j. The pivotal iteration steps of TEDC include *UpdateServerData* and *GetBenefitData*. The function *UpdateServerData* calculates the number of requests for various data by users within the coverage of each edge server. Using the matrix M obtained by *UpdateServerData*, *GetBenefitData* provides the candidate data list, each element is a triple $(data, benefit, latency)$, where $data$ is the type of data, $benefit$ is computed by the number of the requests for data d_k from edge users within the edge server v_j divided by the data cache cost of data d_k, and $latency$ is the data response latency requirement of d_k. Furthermore, the candidate data list will be sorted in descending order by the value of $benefit$, and then sorted in ascending order by $latency$. In this way, the data with many requests, low cache cost, and high data response latency requirements will be cached first. Then, based on the sorted candidate data list, we cache the selected data while meeting the capacities of edge servers (lines 8-15). Finally, f is returned as the approximate solution to a TEDC problem (line 17).

For the computational cost of the TEDC-A algorithm, since the computation of matrix M and the candidate data list can be calculated offline, the time complexity of *UpdateServerData* and *GetBenefitData* are both $\mathcal{O}(1)$. Thus, the total time computational complexity is $\mathcal{O}(|V|)$ for caching data on edge servers, where $|V|$ is the number of edge servers. More specifically, it consists of calculating the number of requests for various data within the coverage of each edge server, sorting data according to the number of requests per unit cost and the data response latency requirement of each data, and selecting data to be cached on edge servers under the capacity constraint. Consequently, TEDC-A can efficiently obtain a solution to the edge data caching problem in polynomial time.

V. Experimental Evaluation

A. Experimental Setup and Datasets

In this section, we experimentally evaluate our approaches' effectiveness and efficiency. All the experiments are conducted on a machine featuring an Intel(R) Xeon(R) Gold 6130 CPU@2 and 192GB RAM. The LGP model is solved with Gurobi.

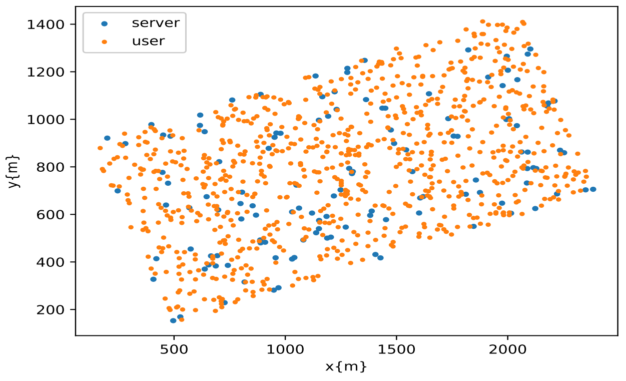
The experiments are conducted on two widely recognized benchmark datasets in the field of edge computing: the EUA dataset [1] and the Shanghai-Telecom dataset[2]. The EUA dataset comprises 125 edge servers and 816 mobile users situated in the Melbourne central business district area of Australia, as depicted in Fig. 2(a). The Shanghai-Telecom dataset consists of 3,233 base stations located throughout Shanghai, China. In the experiments, we focused on edge servers within the Lujiazui Finance and Trade Zone, generating users based on a Gaussian distribution $N(u, \sigma)$. This resulted in 345 edge servers and 1,000 users, as shown in Fig. 2(b). To ensure the connectivity of the edge servers, we randomly generate links between them. Additionally, the available storage resources on each edge server are distributed according to a normal distribution $N(u, \sigma)$, where σ is 1.

B. Competing Methods and Evaluation Metrics

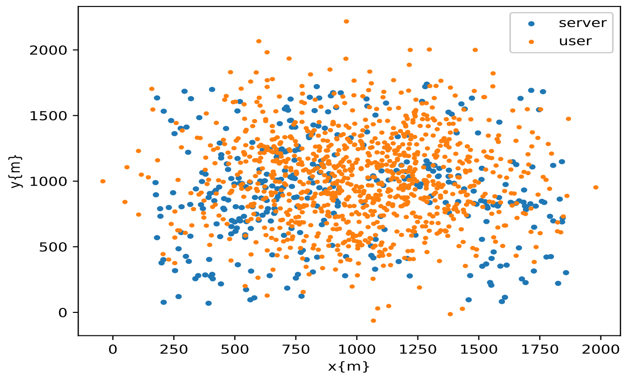
In the experiments, we evaluate the performance of TEDC-IP and TEDC-A against five representative approaches:

- Delay-optimal Cooperative Caching (DCC) [21]: This approach aims to minimize the total user latency by serving the most users from the nearest edge servers, where each edge server caches data with the highest benefits within its coverage area.
- Genetic Algorithm-based Approach (TEDC-GA): This approach is a variation of GA [22], using a crossover

[1]https://sites.google.com/site/heqiang/eua-respository
https://github.com/swinedge/eua-dataset
[2]http://www.sguangwang.com/TelecomDataset.html



(a) EUA Dataset



(b) Shanghai-Telecom Dataset

Fig. 2: EUA dataset and Shanghai-Telecom dataset

operator, a mutation operator, a tournament selection operator, where the Greedy-Covered-Users approach is used in the mutation operation.

- Greedy-Covered-Users (GU): This approach always selects the edge server that covers the most app users to cache data until the capacity constraint (2) is fulfilled.
- Greedy-Connection (GC): This approach always selects the edge server that has the most neighbor edge servers to cache data under Constraint (2).
- Random: This approach always selects the edge server randomly to cache data under Constraint (2).

In our experiments, four metrics are employed to compare and analyze the results: three for assessing effectiveness and one for evaluating efficiency.

- Data Caching Cost (cost), measured by the storage resources utilized for caching data, aiming for minimal expenditure.
- Benefit per Cache Cost (bpc), measured by the ratio of total hops reduced to data cache cost, striving for maximal efficiency.
- Served Request Ratio per Cache Cost (SRRpc), measured by the proportion of served data requests to allocated data cache cost, seeking optimal utilization.
- Computational Overhead (time), measured by the duration required to derive the solution, with a preference for minimal processing time.

C. Experiment Results and Analyses

In the experiments, the coverage radius of edge servers obeys a Gaussian distribution with $u = 50$ and the number of data per user required follows a Gaussian distribution with $u = 3$. Moreover, an edge server's available storage capacities follow a Gaussian distribution $N(10, 1)$ in both EUA datasets and Shanghai-Telecom datasets. The total number of time slots is 200 in the experiments.

Fig. 3 and Table IV present the experimental results on four evaluation metrics on the EUA dataset, where we compare TEDC-IP and TEDC-A with five competing methods. In Fig. 3(a), the results demonstrate that TEDC-IP achieves the lowest data caching cost, and TEDC-A achieves the second lowest data caching cost. It is a pretty good performance, take Time Slot #1 in Table IV as an example, TEDC-IP is superior to Random, GC, GU, DCC, and TEDC-GA with an advantage of 40.62%, 38.70%, 39.36%, 31.32%, and 36.66%, respectively, while TEDC-A is superior to Random, GC, GU, DCC, and TEDC-GA with an advantage of 26.04%, 23.56%, 24.46%, 14.45%, and 21.11%, respectively. Fig. 3(b) demonstrates the benefit per cache cost of the seven approaches over individual time slots. Note that TEDC-IP achieves the highest benefit per cache cost, followed by TEDC-A. Take Time Slot #2 for instance, the benefit per cache cost of all the approaches is 1.96 for TEDC-IP, 1.43 for TEDC-A, 1.07 for TEDC-GA, 1.18 for DCC, 1.05 for GU, 1.07 for GC and 1.05 for Random. TEDC-IP focuses on covering the maximum number of users with minimal cache cost. Thus, it achieves the highest benefit per cache cost. TEDC-A decides whether to cache certain data according to the popularity of the data and the cache cost of the data. In Fig. 3(c), the served request ratio per cache cost of TEDC-IP is again the highest, and TEDC-A is the second highest. Note that methods have the same trend in the benefit per cache cost and the served request ratio per cache cost. The reason is that the higher the data response rate, the higher the revenue of data caching. Regarding CPU time, TEDC-GA takes the most time to find a solution, followed by TEDC-IP. Considering the $\mathcal{NP}$-hard of the TEDC problem and the multi-objective optimization of TEDC-IP, its solution is not unique and it needs to compromise among multiple optimization objectives. The computation time of TEDC-A is similar to that of Random, GC, GU, and DCC. Thus, TEDC-A can accommodate TEDC scenarios on a large scale. In general, TEDC-IP and TEDC-A outperform other methods and achieve superior performance in the data cache cost, the benefit per cache cost, and the served request ratio per cache cost. However, TEDC-IP takes more time to find a solution than Random, GC, GU, and DCC.

Fig. 4 and Table V show the experimental results on the Shanghai-Telecom dataset among TEDC-IP, TEDC-A, and

TABLE IV: Experimental results of data caching among competing approaches on EUA datasets.

Methods	Time Slot t_1				Time Slot t_2				Time Slot t_3			
	cost	bpc	SRRpc	CPU Time	cost	bpc	SRRpc	CPU Time	cost	bpc	SRRpc	CPU Time
Random	96	1.03	30.19	0.0003	95	1.05	29.87	0.0002	95	1.00	29.24	0.0002
GC	93	1.07	31.32	0.0009	93	1.07	30.64	0.0010	94	1.06	31.08	0.0009
GU	94	1.06	30.94	0.0002	94	1.05	30.08	0.0001	93	1.06	31.30	0.0002
DCC	83	1.19	34.95	0.0002	83	1.18	33.97	0.0002	83	1.19	34.97	0.0002
TPEDC-GA	90	1.10	32.20	0.1022	92	1.07	30.64	0.1884	90	1.10	32.13	0.2487
TPEDC-A	71	1.39	40.80	0.0003	68	1.43	41.01	0.0002	71	1.39	40.83	0.0002
TPEDC-IP	57	1.75	51.15	0.0315	51	1.96	55.84	0.0307	59	1.69	49.57	0.0313

TABLE V: Experimental results of data caching among competing approaches on Shanghai-Telecom datasets.

Methods	Time Slot #1				Time Slot #2				Time Slot #3			
	cost	bpc	SRRpc	CPU Time	cost	bpc	SRRpc	CPU Time	cost	bpc	SRRpc	CPU Time
Random	131	0.65	49.24	0.0003	128	0.67	51.92	0.0003	132	0.64	49.35	0.0003
GC	128	0.66	50.21	0.0018	128	0.66	50.81	0.0017	125	0.68	52.96	0.0017
GU	127	0.69	52.39	0.0002	127	0.69	53.10	0.0002	130	0.66	51.20	0.0002
DCC	137	0.63	47.72	0.0002	137	0.62	47.56	0.0002	136	0.62	47.99	0.0002
TPEDC-GA	121	0.71	54.26	0.4981	117	0.73	55.88	0.6196	120	0.72	56.03	0.5848
TPEDC-A	97	0.90	68.25	0.0003	99	0.87	66.72	0.0004	99	0.87	68.80	0.0003
TPEDC-IP	95	0.94	71.08	0.2297	103	0.87	66.54	0.2284	97	0.92	71.00	0.2265

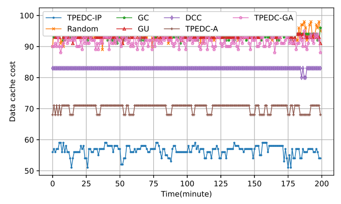
other five representative approaches, including TEDC-GA, DCC, GU, GC and Random. As can be seen from Table V, TEDC-IP and TEDC-A outperform the other methods in terms of data cache cost, benefit per cache cost, and served request ratio per cache cost. Take Time Slot #1 as an example, TEDC-IP outperforms Random, GC, GU, DCC, and TEDC-GA with an advantage of 27.48%, 25.78%, 25.19%, 30.65%, and 21.48%, respectively. TEDC-A achieves data caching cost with an advantage of 25.95% over Random, 24.21% over GC, 23.62% over GU, 29.19% over DCC, and 19.83% over TEDC-GA. In terms of benefit per cache cost, TEDC-IP, TEDC-A, TEDC-GA, DCC, GU, GC, and Random achieve 0.94, 0.90, 0.71, 0.63, 0.69, 0.66 and 0.65, respectively. In terms of served request ratio per cache cost, TEDC-IP performed the best, followed by TEDC-A, and the worst was DCC. From Fig. 4, it is worth noting that the performance of TEDC-A has caught up with the performance of TEDC-IP. This is because TEDC-A weighs the two indicators of data latency and cache cost to choose the right edge server to cache data. Whereas other approaches will cache as much data as possible on edge servers to approach the capacity limit of edge servers, regardless of whether the data caching is reasonable. Therefore, TEDC-A will perform better in large-scale edge data caching scenarios. Except for TEDC-IP and TEDC-A, the performance of TEDC-GA, DCC, GU, GC, and Random was approximately the same.

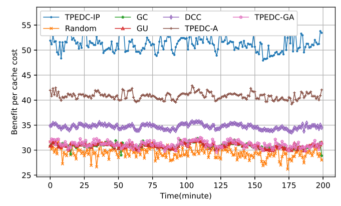
D. Performance Impact of Parameters

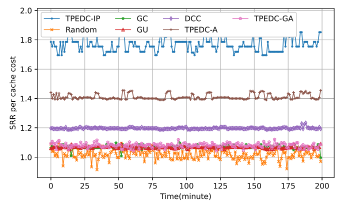
To evaluate the performance of our approaches comprehensively, we vary the following three parameters in the experiments to observe their performance in different TEDC scenarios. In each set of simulations, we change one setting parameter and fix the other two. This way, we observe how the changes in the setting parameters impact the performance of methods on the EUA dataset. Each time a setting parameter varies as follows, the simulation is repeated 100 times and the results are averaged:

- Number of edge servers (m). This parameter impacts the size of graph G and varies from 4 to 9.
- Capacity of edge servers ($a(v_j)$). Edge servers' capacity is generated following a Gaussian distribution with $\sigma = 1$. The storage capacity of each edge server ranges from u = 2, 4, ..., to 12 in steps of 2.
- Number of data request ($|D_t(u_i)|$). The number of data required by edge users follows a Gaussian distribution with $\sigma = 1$. The u is set from 1 to 6.

1) Impact of number of edge servers: Fig.5 shows the performance comparison of each method in the experiments, where the number of edge servers m varies from 4 to 9. From Fig.5(a), it can be seen that as m increases, the total data caching cost required by different methods also increases. In the beginning, there is little difference in the data caching cost among the seven methods because each edge server must cache popular data to satisfy user data requests. As m increases, the data caching costs of Random, GC, GU, and TEDC-GA increase significantly. Comparatively, TEDC-IP and TEDC-A have less increase in data caching cost, especially TEDC-A gets the minimum caching cost in all methods. This is


(a) Data cache cost


(b) Benefit per cache cost


(c) SRR per cache cost

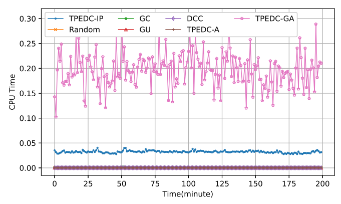
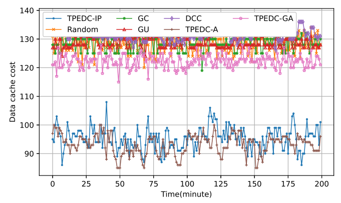
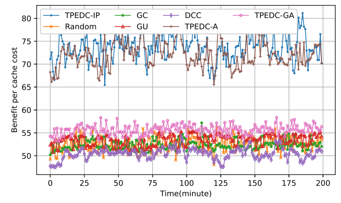
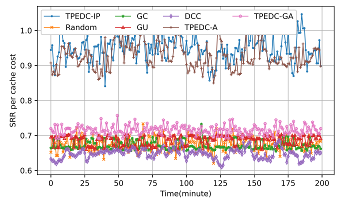

(d) CPU Time

Fig. 3: Experimental results of data cache among competing approaches on EUA datasets.


(a) Data cache cost


(b) Benefit per cache cost


(c) SRR per cache cost

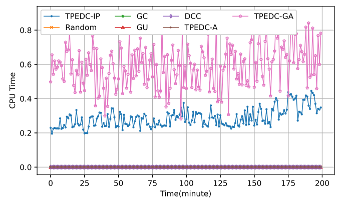
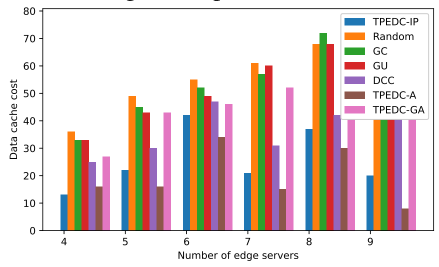
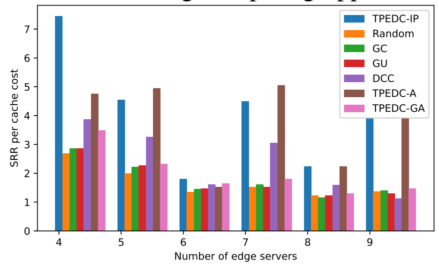

(d) CPU Time

Fig. 4: Experimental results of data cache among competing approaches on Shanghai-Telecom datasets.


(a) Data cache cost


(b) SRR per cache cost

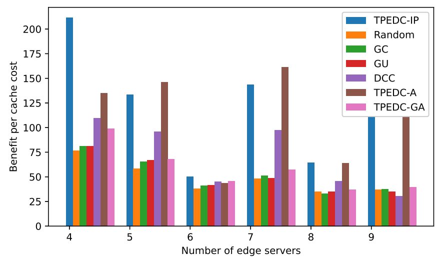
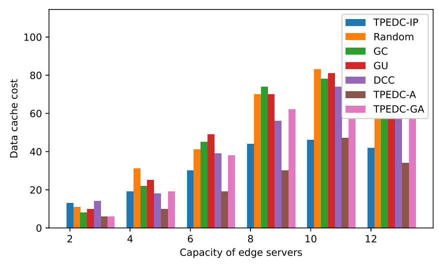
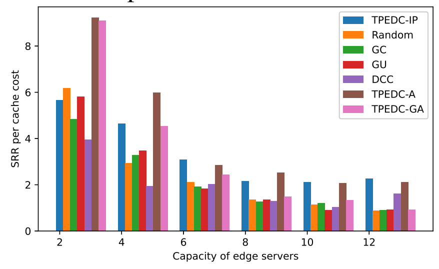

(c) Benefit per cache cost

Fig. 5: Performance comparisons on the variations of edge servers.


(a) Data cache cost


(b) SRR per cache cost

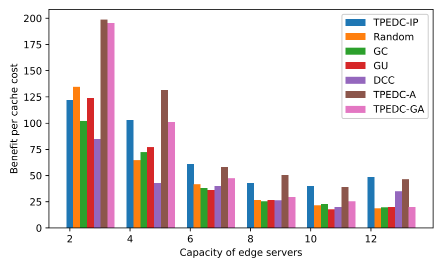
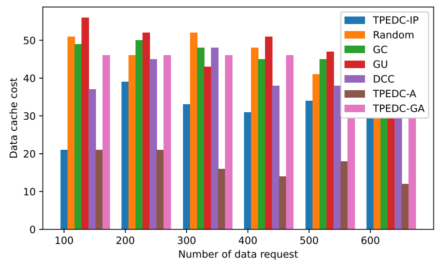
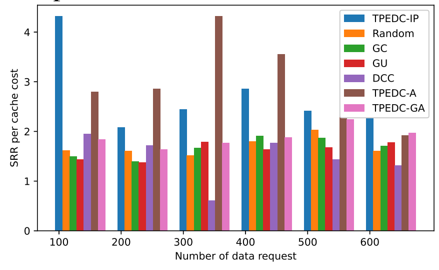

(c) Benefit per cache cost

Fig. 6: Performance comparisons on the variations of server's available capacity.


(a) Data cache cost


(b) SRR per cache cost

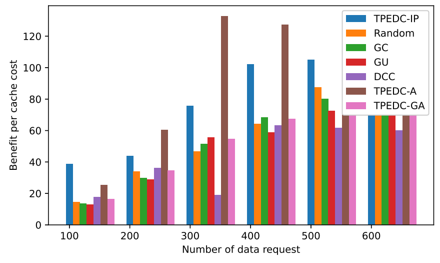

(c) Benefit per cache cost

Fig. 7: Performance comparisons on the variations of data request.

because TEDC-A weighs the data latency and caching cost to choose the right edge server, instead of exhausting the server's storage capacity as much as possible. Therefore, TEDC-A is a good choice for large-scale EDC problems. In Fig.5(b), as the number of edge servers increases, the service request rate per cache cost achieved by all methods decreases. The reason is that when the number of users is fixed, the total number of requests is also fixed. Therefore, with a fixed number of users, the service request rate per cache cost decreases when the service provider rents more cache storage resources. The benefit per unit cache cost for all methods has the same trend in Fig.5(c) as in Fig.5(b).

2) Impact of edge server capacities: Fig.6 shows the performance comparison of each method in the experiments with different server storage capacities. As the server's available storage capacity $a(v_j)$ varies from 2 to 12, the costs spent on data caching of all methods increase in Fig.6(a). It is worth noting that the caching cost for TEDC-IP and TEDC-A increases slowly compared to the other methods. Because TEDC-IP and TEDC-A select only the necessary edge servers to cache data, whereas the other algorithms cache as much data as possible on the edge servers if the edge servers have storage resources. Fig.6(b) shows the trend that the served request ratio per cache cost of all methods decreases as $a(v_j)$ increases. This is because as the storage capacity of the servers increases, these methods still cache more data on the edge servers with no change in user requests, resulting in higher data caching costs. As for the benefit per cache cost in Fig.6(c), it shows the same trend as Fig.6(b).

3) Impact of number of data requests: Fig.7 shows the performance comparison of the various methods in the experiments with variations of data request. As shown in Fig.7(a) and Fig.7(b), when the number of data requests per user increased from 1 to 6, the performance of methods in terms of data caching cost and service request rate per unit caching cost does not change significantly. In Fig.7(c), the benefit per unit caching cost increases with the number of data requests. The reason is that when the edge server capacity is fixed, the more data requests from users, the higher the gain per unit cost.

VI. Related Work

With the widespread use of the Internet, network congestion is becoming a major issue when handling a large number of user requests [23]–[25]. In recent years, researchers have begun to focus on edge computing's caching architecture as a solution to this challenge [26]–[29]. Edge data caching can effectively alleviate network load, improve data access speed, and provide users with a more stable and faster service experience.

Existing research has explored various methods for deploying and distributing edge data, including minimizing cache costs, maximizing data hit ratio, minimizing data latency, and reducing energy consumption [30]–[32]. Zhang et al. [33] focused on enhancing the overall energy efficiency of base stations by devising a caching strategy that incorporates user association and power allocation within the cellular network. Cao et al. [34] addressed the expenses incurred during data delivery by proposing an auction mechanism aimed at identifying the most efficient data caching solution. The authors of [35] introduced a novel edge caching framework that integrates caches on intelligent vehicles into the network cache infrastructure. This innovative approach notably enhanced both the resource utilization and the overall effectiveness. Zhang et al. [36] proposed a hierarchical caching mechanism in the edge computing environment to maximize the hitting rate, with consideration of wireless communication. In [30], the authors proposed a Lyapunov-based algorithm to minimize the overall data retrieval latency for the budgeted service placement problem. Deng et al. [37] provided a primal-dual algorithm named IDA4ReE to solve service deployment problems. Yang et al. in [38] formulated the cost-aware energy-efficient data offloading problem as a discrete-time optimal control problem from an energy-efficiency perspective. In [39], Yu et al. integrated social relationships into multi-access edge computing and proposed a Monte-Carlo-based method named TA-MCTS to minimize energy consumption. Zhang et al. [40] jointly considered the interference between mobile nodes, caching state, link scheduling, and routing, and proposed an online algorithm based on the Lyapunov drift-plus-penalty theory to minimize energy consumption. In [21], Xia et al. addressed the unique server capacity constraint and edge servers' communication capabilities in the Constrained Edge Data Caching (CEDC) problem.

However, these studies failed to consider the characteristics of different types of application data, including the varying storage resource requirements and latency preferences associated with each data type. Specifically, users have distinct latency demands for different types of data, which directly influence the quality of their app usage experience. Furthermore, real-world edge computing environments exhibit temporal dynamics, particularly in the fluctuations of user distribution and data requests over time. Therefore, we address these practical challenges by introducing a temporal-aware edge data caching problem with specified latency preferences.

VII. Conclusion

In this paper, we formulated the problem of temporal-aware data caching with specified latency preference (TEDC) in edge computing. We first proved that the TEDC problem is $\mathcal{NP}$-hard. To solve this problem, an optimal approach named TEDC-IP based on integer programming is proposed to minimize the users' overall data retrieval latency and minimize the data caching cost in multiple time slots. Then, an approximate approach named TEDC-A is proposed for finding approximate solutions to large-scale TEDC problems efficiently. Extensive experiments were conducted on two widely-used real-world data sets, e.g. EUA-dataset and Shanghai-Telecom datasets, to evaluate the performance of the proposed approaches. The results showed that our approaches significantly outperformed the state-of-the-art approaches in various TEDC scenarios.

In future work, we will consider the mobility of edge users, data sharing between users, security constraints on data regulation, and social information.

Acknowledgment

This work was supported by National Natural Science Foundation of China (No. 61772128, 62172088), and Shanghai Natural Science Foundation (No. 21ZR1400400).

References

[1] A. Beheshti, "Empowering generative ai with knowledge base 4.0: Towards linking analytical, cognitive, and generative intelligence," in *IEEE International Conference on Web Services (ICWS)*. IEEE, 2023, pp. 763–771.

[2] J. Peng, Q. Li, X. Ma, Y. Jiang, Y. Dong, C. Hu, and M. Chen, "Magnet: cooperative edge caching by automatic content congregating," in *Proceedings of the ACM Web Conference*, 2022, pp. 3280–3288.

[3] T. X. Tran, M.-P. Hosseini, and D. Pompili, "Mobile edge computing: Recent efforts and five key research directions," *IEEE COMSOC MMTC Communications - Frontiers*, 2017.

[4] J. Yang, A. K. Bashir, Z. Guo, K. Yu, and M. Guizani, "Intelligent cache and buffer optimization for mobile vr adaptive transmission in 5G edge computing networks," *Digital Communications and Networks*, 2023.

[5] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[6] S. Wu, C. Niu, J. Rao, H. Jin, and X. Dai, "Container-based cloud platform for mobile computation offloading," in *IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE, 2017, pp. 123–132.

[7] G. Zou, Y. Liu, Z. Qin, J. Chen, Z. Xu, Y. Gan, B. Zhang, and Q. He, "TD-EUA: Task-decomposable edge user allocation with QoE optimization," in *International Conference on Service-Oriented Computing (ICSOC)*. Springer, 2020, pp. 215–231.

[8] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2020–2033, 2018.

[9] T. Shi, Z. Cai, J. Li, H. Gao, J. Chen, and M. Yang, "Services management and distributed multihop requests routing in mobile edge networks," *IEEE/ACM Transactions on Networking*, vol. 31, no. 2, pp. 497–510, 2022.

[10] Z. Xue, C. Liu, C. Liao, G. Han, and Z. Sheng, "Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems," *IEEE Transactions on Vehicular Technology*, 2023.

[11] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundant placement for microservice-based applications at the edge," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1732–1745, 2020.

[12] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE International Conference on Computer Communications (IEEE INFOCOM)*. IEEE, 2019, pp. 10–18.

[13] X. Xia, F. Chen, Q. He, G. Cui, P. Lai, M. Abdelrazek, J. Grundy, and H. Jin, "Graph-based optimal data caching in edge computing," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 477–493.

[14] Y. Liu, Y. Han, A. Zhang, X. Xia, F. Chen, M. Zhang, and Q. He, "Qoe-aware data caching optimization with budget in edge computing," in *IEEE International Conference on Web Services (ICWS)*. IEEE, 2021, pp. 324–334.

[15] Z. Ni, M. Yuan, and H. Tang, "Qoe-aware data caching optimization in edge computing environment," in *IEEE International Conference on Services Computing (SCC)*. IEEE, 2022, pp. 65–73.

[16] S. Deng, Z. Xiang, P. Zhao, J. Taheri, H. Gao, J. Yin, and A. Y. Zomaya, "Dynamical resource allocation in edge for trustable internet-of-things systems: A reinforcement learning method," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6103–6113, 2020.

[17] J. Wu, Z. Chen, and M. Zhao, "Information cache management and data transmission algorithm in opportunistic social networks," *Wireless Networks*, vol. 25, no. 6, pp. 2977–2988, 2019.

[18] Z. Li, C. Yang, X. Huang, W. Zeng, and S. Xie, "Coor: collaborative task offloading and service caching replacement for vehicular edge computing networks," *IEEE Transactions on Vehicular Technology*, 2023.

[19] R. Halalai, P. Felber, A.-M. Kermarrec, and F. Taïani, "Agar: A caching system for erasure-coded data," in *International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 23–33.

[20] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017, pp. 276–286.

[21] X. Xia, F. Chen, J. Grundy, M. Abdelrazek, H. Jin, and Q. He, "Constrained app data caching over edge server graphs in edge computing environment," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2635–2647, 2021.

[22] H. Song, B. Gu, K. Son, and W. Choi, "Joint optimization of edge computing server deployment and user offloading associations in wireless edge network via a genetic algorithm," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 4, pp. 2535–2548, 2022.

[23] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "Adaptsize: Orchestrating the hot object memory cache in a content delivery network," in *USENIX Symposium on Networked Systems Design and Implementation)*, 2017, pp. 483–498.

[24] A. S. Ali, K. R. Mahmoud, and K. M. Naguib, "Optimal caching policy for wireless content delivery in D2D networks," *Journal of Network and Computer Applications*, vol. 150, p. 102467, 2020.

[25] P. Zhang, M. Sun, Y. Tu, X. Li, Z. Yang, and R. Wang, "Device-edge collaborative differentiated data caching strategy towards aiot," *IEEE Internet of Things Journal*, 2023.

[26] K. Ma, B. Yang, Z. Yang, and Z. Yu, "Segment access-aware dynamic semantic cache in cloud computing environment," *Journal of Parallel and Distributed Computing*, vol. 110, pp. 42–51, 2017.

[27] M. Reiss-Mirzaei, M. Ghobaei-Arani, and L. Esmaeili, "A review on the edge caching mechanisms in the mobile edge computing: A social-aware perspective," *Internet of Things*, vol. 22, p. 100690, 2023.

[28] S. Tamoor-ul Hassan, S. Samarakoon, M. Bennis, M. Latva-Aho, and C. S. Hong, "Learning-based caching in cloud-aided wireless networks," *IEEE Communications Letters*, vol. 22, no. 1, pp. 137–140, 2017.

[29] Y. Wang, Y. Zhang, X. Han, P. Wang, C. Xu, J. Horton, and J. Culberson, "Cost-driven data caching in the cloud: an algorithmic approach," in *IEEE International Conference on Computer Communications (IEEE INFOCOM)*. IEEE, 2021, pp. 1–10.

[30] J. Zhou, J. Fan, J. Wang, and J. Jia, "Dynamic service deployment for budget-constrained mobile edge computing," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 24, p. e5436, 2019.

[31] J. Zhou, F. Chen, Q. He, X. Xia, R. Wang, and Y. Xiang, "Data caching optimization with fairness in mobile edge computing," *IEEE Transactions on Services Computing*, vol. 16, no. 3, pp. 1750–1762, 2023.

[32] J. Yang, Z. Guo, J. Luo, Y. Shen, and K. Yu, "Cloud-edge-end collaborative caching based on graph learning for cyber-physical virtual reality," *IEEE Systems Journal*, 2023.

[33] H. Zhang, S. Huang, C. Jiang, K. Long, V. C. Leung, and H. V. Poor, "Energy efficient user association and power allocation in millimeter-wave-based ultra dense networks with energy harvesting base stations," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 9, pp. 1936–1947, 2017.

[34] X. Cao, J. Zhang, and H. V. Poor, "An optimal auction mechanism for mobile edge caching," in *International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 388–399.

[35] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative content caching in 5G networks with mobile edge computing," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 80–87, 2018.

[36] X. Zhang and Q. Zhu, "Collaborative hierarchical caching over 5G edge computing mobile wireless networks," in *IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[37] S. Deng, Z. Xiang, J. Taheri, M. A. Khoshkholghi, J. Yin, A. Y. Zomaya, and S. Dustdar, "Optimal application deployment in resource constrained distributed edges," *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, pp. 1907–1923, 2020.

[38] C. Yang and R. Stoleru, "Ceo: cost-aware energy efficient mobile data offloading via opportunistic communication," in *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2020, pp. 548–554.

[39] S. Yu, B. Dab, Z. Movahedi, R. Langar, and L. Wang, "A socially-aware hybrid computation offloading framework for multi-access edge computing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 6, pp. 1247–1259, 2019.

[40] X. Zhang, P. Huang, L. Guo, and Y. Fang, "Social-aware energy-efficient data offloading with strong stability," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1515–1528, 2019.