Cluster Computing
https://doi.org/10.1007/s10586-024-04647-5

Service selection based on blockchain smart contracts in cloud-edge environment

Yingying Ning[1] · Jing Li[1] · Ming Zhu[1] · Chuanxi Liu[1]

Received: 15 February 2024 / Revised: 30 May 2024 / Accepted: 21 June 2024

Abstract
The rapid integration of cloud computing and edge computing has brought the cloud-edge environment into the spotlight in information technology. Within this context, the selection of high-quality and reliable services is crucial to meet the needs of users. However, ensuring the reliability of service information is a challenge due to its vulnerability to tampering. This research paper proposes a method for service selection in the cloud-edge environment based on blockchain smart contracts. By leveraging blockchain technology, this method achieves decentralized and trustworthy service selection. Through smart contracts, user interactions are securely recorded, significantly reducing the risk of information tampering and enhancing information reliability. Additionally, the Arithmetic Optimization Algorithm is improved for service selection on the blockchain by introducing mutation and crossover operations. Experimental results demonstrate that this method effectively prevents tampering with service information and improves the utility value of selected services compared to traditional methods and metaheuristic algorithms mentioned.

Keywords Cloud computing · Edge computing · Blockchain · Smart contract · Service selection · Arithmetic optimization algorithm

1 Introduction

With the rapid advancement of information technology, human society is swiftly entering a new era characterized by digitalization and networking. In this context, cloud computing and edge computing, which are two significant computing paradigms, are profoundly reshaping the way people conduct their lives and work [1]. Cloud computing, achieved through virtualization and resource sharing, offers users highly flexible and efficient computing power, facilitating the swift deployment and expansion of diverse applications and services. However, in cloud computing, where data is processed on remote servers, there may be increased data transmission and response time [2]. Conversely, edge computing moves computing resources and data storage to the periphery of the network. This structure aids in decreasing data transmission delay, resulting in quicker response time. Nevertheless, it is constrained by limited resources, including smaller computing and storage capacities [3]. It is clear that cloud computing and edge computing can complement each other [4]. Consequently, the cloud-edge environment arises with the objective of integrating the advantages of both cloud computing and edge computing, leveraging their combined capabilities while compensating for their individual limitations [5].

With the increasing use of cloud computing and edge computing, several new challenges have arisen. Among these challenges is the selection of the most appropriate service from a vast array of options to fulfill the varying needs of users [6]. In the cloud-edge environment, there are a large number of heterogeneous resources and services, making service selection more complex. Traditional

✉ Ming Zhu
zhu_ming@sdut.edu.cn

Yingying Ning
17852739233@163.com

Jing Li
li_jing@sdut.edu.cn

Chuanxi Liu
853388104@qq.com

[1] School of Computer Science and Technology, Shandong University of Technology, Zibo 255000, China

Published online: 02 July 2024

Table 1 Notations

Expression	Description
sp	Service provider
SP	Set of service providers
s	Service
S	Set of services
S_i	Set of services provided by service provider sp_i
$AddrS$	Address of service provider
$AddrC$	Address of the contract for a specific service
B	Cloud or edge server
F_s	Functional attribute of a service
Q	Finite set of non-functional attributes of a service
U	Service requester
A	Service request
K	Set of tasks
$Task$	Requested number of tasks
M	Structure composition of tasks
C	Maximum cost acceptable to the service requester
N_O	Minimum number of service selection solutions the requester wants to obtain
t_{limit}	Longest time to wait for service selection solutions
t_{real}	Real duration for waiting to choose service solutions
$Cost$	Price of the service
RT	Response time of the service
Av	Availability of the service
p	Non-functional attribute
$Norm$	Normalized value of the non-functional attribute
w	Weight
F	Utility value of the service
$sssp$	Service selection solution provider
$SSSP$	Set of service selection solution providers
CS	Candidate solutions
n	Name of the solution
R	Service selection solution submitted by provider
O	Service selection solution processed by the smart contract
o	Set of service selection solution processed by the smart contract
$AddrSSS$	Address of the service selection solution provider
$Cost_S$	Total cost of selected services
t_{RT}	Total response time of selected services
a	Total availability of selected services
H	Whether the solution meets the requester's cost requirement
ID	Collection of IDs for all services
ID^{sp}	Set of IDs for services that the service provider sp can offer
id	ID of the service
$OptrR$	Optimal service selection solution
$addrC$	Address of the Specific usage contract for the service selected by the optimal solution
$Cost_{total}$	Total cost for the service requester
$Cost_P$	Expense associated with utilizing the smart contract
$Cost_{SSS}$	Expense associated with utilizing the optimal service selection solution
$SSSP^U$	Collection of service selection solution providers that offer solutions for U
$sssp^U$	Service selection solution provider that offers the solution for U

service selection methods may face challenges such as information asymmetry and information tampering [7]. During the service selection process, information asymmetry may prevent users from gaining a comprehensive understanding of the quality and performance of the services offered by service providers. Service providers may tamper with service information through the centralized platform's administrator to gain more revenue, making it difficult for users to obtain reliable information. Therefore, exploring a method that can effectively address these issues becomes crucial.

Blockchain technology, as a distributed and decentralized solution, has the potential to address the issues of information asymmetry and data tampering in service selection [8]. It can record information about the services provided by service providers. Its characteristic of being public and transparent allows users to obtain more accurate information, thereby mitigating the problems caused by information asymmetry. Once data is recorded in the blockchain, it is difficult to tamper with. Each block contains information from the previous block, and any tampering with the data can be quickly detected. This characteristic safeguards the integrity of the data, preventing any malicious alterations or counterfeiting, and ensuring that the data accessed by users is reliable [9]. The decentralized nature of the blockchain enables multiple nodes to verify and record information, establishing a higher level of reliability. Smart contracts implemented on the blockchain enable users to conduct secure transactions without the involvement of third parties. By utilizing a blockchain-based service selection method, a fair and dependable mechanism for selecting services in cloud-edge environments can be established, providing users with more reliable services.

Additionally, when it comes to selecting services in the cloud-edge environment based on blockchain, there is an important aspect to take into account: whether users can find the best service selection solution. This paper explores a method that enables users to achieve automated service selection through blockchain smart contracts. This method involves comparing solutions generated by various service selection algorithms to guarantee that users receive the most optimal outcome. Furthermore, we propose an enhanced Arithmetic Optimization Algorithm (AOA) [10] for the service selection algorithm.

The key contributions of this paper include:

1. An architecture for service selection based on Ethereum smart contracts in cloud-edge environments is proposed. This ensures transparency and prevents tampering of information during transactions between service providers, service requesters, and service selection solution providers, assisting users in establishing a reliable service selection environment. A smart contract is designed specifically for this architecture, allowing users to utilize it for service selection. Users in this architecture do not need to write smart contracts themselves, and at the same time, service selection solution providers are encouraged to offer superior solutions in competition.
2. A proposal is put forward for an algorithm to select services on the blockchain. The algorithm is called the Differential Evolution Arithmetic Optimization Algorithm (DEAO). The DEAO combines the mutation and crossover operations of the Differential Evolution (DE) [11] with the AOA. Comparative experiments indicate that the DEAO is more effective than the AOA and other referenced studies in avoiding local optima, showcasing its exceptional capability to discover global optimal solutions.

In this paper, we discuss related work in Sect. 2. The preliminary knowledge is introduced in Sect. 3. In Sect. 4, the proposed method is presented. we report experimental results in Sect. 5. Finally, conclusion is drawn in Sect. 6.

2 Related work

As a decentralized distributed ledger technology, blockchain offers new possibilities for service selection in cloud-edge environments. In this section, we introduce the applications of service selection in cloud-edge environments, blockchain-based service selection, and the application of blockchain in cloud-edge environments.

2.1 Service selection in cloud-edge environments

With the rapid development of cloud computing and Internet of Things technology, an increasing number of application scenarios require services to be provided in the cloud-edge environment. Cloud-edge computing extends the capabilities of computing, storage, and processing from the cloud to edge devices, enabling faster and real-time data processing. However, due to the wide and dynamically changing distribution of resources in the cloud-edge environment, selecting the best service has become a challenging problem. Researchers have proposed different methods to address this challenge. Zhu et al. introduced an extended Grey Wolf Algorithm [12]. This algorithm incorporates roulette wheel selection, solving the service

selection problem based on edge and cloud computing. This algorithm introduces roulette wheel selection into the Grey Wolf Algorithm, resolving the issue of service selection based on edge and cloud computing. In a dynamic and heterogeneous edge-cloud collaborative service environment, Wang et al. proposed a model based on Petri nets for selecting the optimal combination of services [13]. Jian et al. put forward an improved Bee Algorithm [14]. This algorithm effectively addresses the challenge of combining cloud services while meeting the local Quality of Service (QoS) requirements of edge users, maximizing the overall QoS value of composite services.

The researchers above considered various service selection approaches to solve the problem of service selection in the cloud-edge environment, resulting in optimized outcomes. However, in practical applications, there is a risk of tampering with service information. Directly applying these methods in practice may lead to the selection of tampered services, thereby causing service requesters to utilize services with the poorer QoS. To overcome this issue, some scholars have turned their attention to blockchain-based service selection.

2.2 Blockchain-based service selection

To prevent the tampering of service information, some researchers have explored the application of blockchain technology in addressing the problem of service selection. Wang et al. proposed a smart contract-based method for QoS-aware service selection, which ensures transaction reliability and data authenticity while reducing transaction costs [15]. However, this method involves creating a smart contract for each task to select services, which is time-consuming and not suitable for addressing the problem of selecting services when there are too many tasks. Sridevi et al. introduced a blockchain-based architecture and algorithm for QoS-aware semantic service selection [9]. They use blockchain as a management tool for Service Level Agreement, allowing participants to interact in a trustless environment. However, when handling multi-task service selection, this method requires deploying multiple smart contracts to complete the service selection, which consumes a substantial amount of storage resources. Lyu et al. developed a system that combines cloud platforms with blockchain, using a Genetic Algorithm (GA) [16] to solve the service selection problem and achieve immutable transaction records and traceable information storage [8]. However, the GA is prone to getting stuck in local optima.

In summary, there are several challenges in current research on blockchain-based service selection. For instance, considering the cost of time and storage resource consumption, the more tasks there are, the more smart contracts need to be created, which is not conducive to solving the problem of complex service selection. Additionally, the algorithm employed for service selection is prone to getting trapped in local optimal solutions, which is unfavorable for finding global optimal solutions.

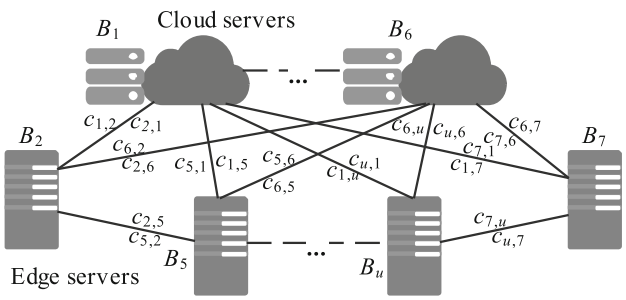

Fig. 1 Cloud-edge environment

2.3 Application of blockchain in cloud-edge environments

Blockchain technology in the cloud-edge environment can provide a secure and reliable mechanism for data exchange, resource management, and service delivery. It can enhance the scalability and establish trust for both cloud and edge computing, driving the development and application of these technologies. To address the resource selection problem, Kaur et al. proposed a multi-criteria statistical approach [17]. This method effectively leverages the advantages of both edge and cloud computing, meeting the application requirements of the Internet of Things and blockchain in Industry 4.0. The method has a positive impact on enhancing the service provisioning rate, optimizing throughput, and minimizing transmission latency of both edge and cloud servers. Paper [18] introduced a blockchain-based service orchestrator. This method enables cross-service communication in the edge and cloud environments, facilitating the full utilization of network markets. Duan et al. designed a distributed virtual machine agent to prevent the integrity of data in edge servers from being compromised [19]. And they established a blockchain-based data protection system for edge and cloud. This method can defend against attacks from cloud service providers and ensure the integrity of data.

However, relevant studies have primarily focused on the issue of service selection in cloud-edge environments, as well as blockchain-based service selection problems and the application of blockchain in cloud-edge environments.

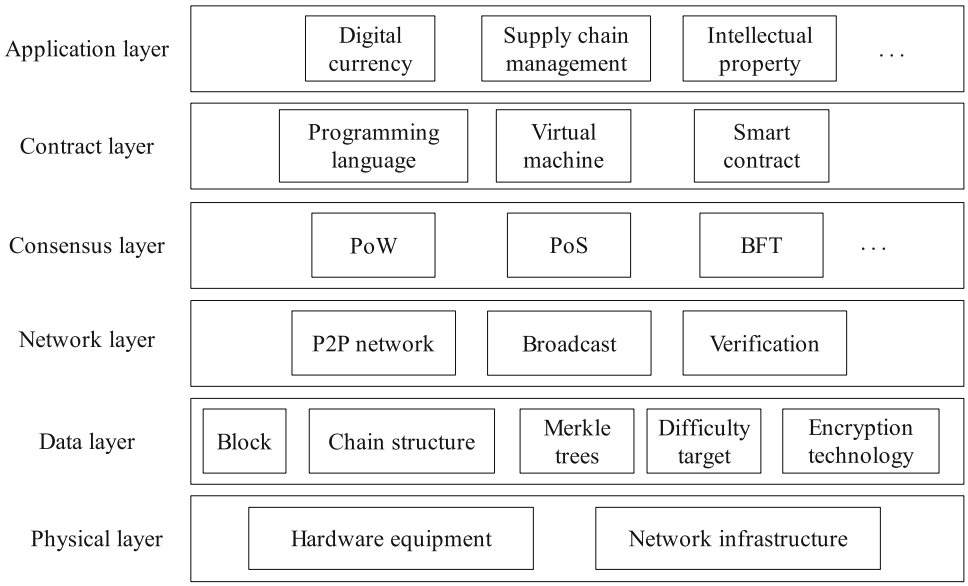

Fig. 2 Basic architecture of blockchain

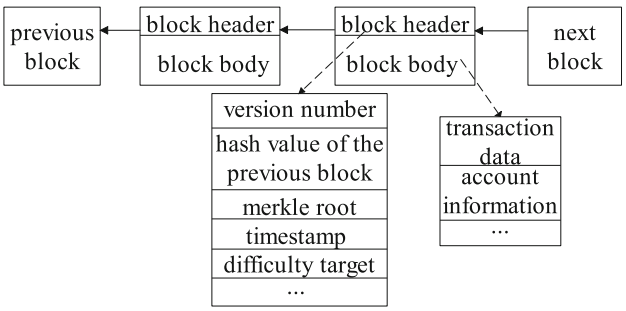

Fig. 3 Data structure of blockchain

There has been relatively little exploration of the problem of service selection in cloud-edge environments using blockchain and smart contracts. Therefore, further research work is still necessary.

3 Preliminary knowledge

In this section, we offer an overview of cloud-edge environments, blockchain technology, the issue of service selection, and metaheuristic algorithms, while also presenting the definitions pertaining to service selection based on blockchain.

3.1 Cloud-edge environments

The cloud-edge environment is a computing model that combines edge computing with cloud computing. The cloud-edge environment brings computing resources and services closer to the source of data generation, aiming to reduce data transmission latency, enhance application performance, and support real-time responsiveness and distributed data processing [20].

$$\begin{array}{c} \\ B_1 \\ B_2 \\ \dots \\ B_i \\ \dots \\ B_u \end{array} \begin{array}{c} \begin{array}{cccccc} B_1 & B_2 & \dots & B_j & \dots & B_u \end{array} \\ \begin{pmatrix} 0 & c_{1,2} & \dots & c_{1,j} & \dots & c_{1,u} \\ c_{2,1} & 0 & \dots & c_{2,j} & \dots & c_{2,u} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ c_{i,1} & c_{i,2} & \dots & c_{i,j} & \dots & c_{i,u} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ c_{u,1} & c_{u,2} & \dots & c_{u,j} & \dots & 0 \end{pmatrix} \end{array} \tag{1}$$

As shown in Fig. 1, each edge server can transfer data and establish connections with other servers over the Internet. Similarly, each cloud server is capable of transferring data and establishing connections with other servers over the Internet. The network topology between edge servers and cloud servers can be represented by an adjacency matrix, as shown in formula (1). Where, B represents cloud or edge servers, totaling u servers, and the unit $c_{i,j}$ represents the time taken for data transmission between two servers B_i and B_j.

3.2 Blockchain

Blockchain is a special type of distributed database that combines various technologies like peer-to-peer communication, encryption algorithms, and consensus mechanisms. It embodies features like multi-party consensus, decentralization, distribution, and tamper resistance. Blockchain is particularly effective in facilitating communication and transactions between entities that have a relatively low level of trust but maintain close business ties. Through the establishment of a trust mechanism, it enables secure and verifiable transaction verification

among business entities that may not possess inherent trust in one another [21].

The basic architecture of blockchain is illustrated in Fig. 2, including the physical layer, data layer, network layer, consensus layer, contract layer, and application layer [22]. We provide a detailed explanation of the functions and relationships of each layer.

1. Physical Layer. The physical layer serves as the fundamental basis of the blockchain, encompassing hardware components and network infrastructure such as computers, servers, storage devices, sensors, etc. It is responsible for providing the necessary computational power and storage resources for the blockchain network, while also offering essential support to the higher layers.
2. Data Layer. In the realm of blockchain technology, the fundamental component is known as a block. Each block houses a cluster of transaction records and associated metadata. Crucially, each block contains a hash pointer pointing to the previous block, thus creating a linked structure. A hash pointer is computed based on the previous block. The advantage of this back-to-front pointing is that it is possible to know if someone has tampered with the data in the blockchain by simply saving the hash of the last block. This is because if someone tampers with the contents of a block in the blockchain, then the hash pointer of the block after that block has to be modified as well, and so on until the pointer to the last block has been modified. The data structure of the blockchain is shown in Fig. 3, with each block comprising a block header and a block body [23]. The dashed arrow points to the specific content of the block. The block header contains important information about the blockchain. It includes different elements such as the version number, the hash value of the previous block, the Merkle root, the timestamp, and the difficulty target, among other details. The block body comprises transaction data and account information, alongside other particulars. Various blockchain systems may adopt different approaches for implementation. Furthermore, the data layer incorporates hash encryption algorithms and asymmetric encryption algorithms to guarantee the integrity and security of data. It can be perceived as a distributed and tamper-proof database.
3. Network Layer. The blockchain consists of a decentralized network where all nodes participate. Whenever a node generates a new block, it shares this information with its neighboring nodes through broadcasting. Once the neighboring nodes successfully validate the block, they pass on the data to other nodes. Eventually, when the majority of nodes in the system validate the block, it becomes officially added to the blockchain.
4. Consensus Layer. The consensus mechanism refers to a collection of regulations and algorithms that enable nodes in a blockchain network to reach an agreement. It ensures that all nodes unanimously acknowledge the state and transactions of the blockchain without centralized control. Popular consensus mechanisms include Proof of Work (PoW), Proof of Stake (PoS), Byzantine Fault Tolerance (BFT), and various others.
5. Contract Layer. The contract layer provides programming languages, virtual machines, and execution environments that allow developers to create and deploy smart contracts. These smart contracts, which run on the blockchain, operate independently of third-party control. They automatically execute according to predetermined conditions, adhering to the concept of "code is law" [21]. In general, smart contracts are activated by external transactions or specific events. Once the predetermined triggering conditions are satisfied, the smart contracts automatically carry out their functions based on the predefined logic and operations. The execution of smart contracts occurs on nodes. These smart contracts can access on-chain data while they are being executed. Lastly, each node verifies transactions, records the outcome of smart contract executions, and updates the state to the latest block.
6. Application Layer. The top layer of the blockchain is known as the application layer, which encompasses a wide range of application scenarios and business domains. These include digital currencies, supply chain management, Internet of Things, healthcare, intellectual property, and more. By utilizing the underlying structure and technology, the application layer is responsible for creating specific blockchain applications that offer users a variety of functionalities and services.

3.3 Service selection

Definition 1 (Service provider) The service provider is denoted as sp, $SP = \{sp_1, sp_2, \ldots, sp_i, \ldots, sp_{|SP|}\}$ represents the collection of service providers, and sp_i refers to the i^{th} service provider in the collection.

Definition 2 (Service) The service, denoted as s, is a six-tuple $\{id, AddrS, AddrC, B, F_s, Q\}$. The id is a unique identifier for the service, $AddrS$ represents the address of the service provider, $AddrC$ is the contract address used for accessing the service, B denotes either an edge server or a cloud server responsible for deploying and running the service, F_s represents the functional attributes of the service, and Q is a finite set of the QoS parameters.

The set of services is denoted as $S = \{s_1, s_2, \ldots, s_i, \ldots, s_{|S|}\}$, where s_i refers to the i^{th} service.

The collection of service identifiers is represented as $ID = \{id_1, id_2, \ldots, id_i, \ldots, id_{|ID|}\}$, with id_i denoting the ID of the i^{th} service.

$S_i (S_i \subseteq S)$ encompasses all the services that can be provided by the service provider sp_i.

The set of IDs for all services that can be provided by service provider sp_i is represented by ID_i^{sp}.

The requester needs to complete a set of tasks, and each task requires invoking a service for completion. The requester hopes to find a solution where the selected services have functionalities that meet the task requirements and possess the optimal QoS [24]. In this paper, we use cost, response time, and availability to measure the QoS. The QoS, also referred to as non-functional attributes. Cost and response time are negative attributes—the higher the value, the lower the QoS. Availability is a positive attribute—the higher the value, the higher the QoS [25].

Definition 3 (QoS)

Cost (*Cost*): The expense incurred by the requester for using services.

Response Time (*RT*): The total time taken from receiving the requester's service request to the completion of the service execution.

Availability (*Av*): The probability of successfully invoking a service. It is expressed as the ratio of successful service accesses to the total number of service accesses, i.e., $Av = As/An$. Where, *As* represents the number of successful service responses, and *An* represents the total number of service accesses.

When presented with two services, s_i and s_j, there are three options for selecting these services: sequential invocation $s_i; s_j$, parallel invocation $s_i | s_j$, and selection invocation $s_i || s_j$ [26].

Definition 4 (Determining the cost of choosing services s_i and s_j)

$$Cost(s_i; s_j) = Cost(s_i) + Cost(s_j) \tag{2}$$

$$Cost(s_i | s_j) = Cost(s_i) + Cost(s_j) \tag{3}$$

$$Cost(s_i || s_j) = min\{Cost(s_i), Cost(s_j)\} \tag{4}$$

where $min\{Cost(s_i), Cost(s_j)\}$ refers to taking the smaller value between $Cost(s_i)$ and $Cost(s_j)$.

Definition 5 (Calculation of response time for selecting services s_i and s_j)

$$T(s_i; s_j) = \begin{cases} T(s_i) + T(s_j), & \text{if } s_i \in B_k,\ s_j \in B_k \\ T(s_i) + T(s_j) + T(B_k, B_p), & \text{if } s_i \in B_k,\ s_j \in B_p \end{cases} \tag{5}$$

$$T(s_i | s_j) = \begin{cases} max\{T(s_i), T(s_j)\}, & \text{if } s_i \in B_k,\ s_j \in B_k \\ max\{T(s_i) + T(B_k, B_x), T(s_j) + T(B_p, B_y)\}, & \text{if } s_i \in B_k,\ s_j \in B_p \end{cases} \tag{6}$$

$$T(s_i || s_j) = \begin{cases} min\{T(s_i), T(s_j)\}, & \text{if } s_i \in B_k,\ s_j \in B_k \\ min\{T(s_i) + T(B_k, B_x), T(s_j) + T(B_p, B_y)\}, & \text{if } s_i \in B_k,\ s_j \in B_p \end{cases} \tag{7}$$

where B_k and B_p are two servers, $s_i \in B_k$ represents that s_i is deployed on B_k, $T(B_k, B_p)$ indicates the time for data transfer from B_k to B_p. In equations (6) and (7), we assume that the successor service of s_i is located on B_x, and the successor service of s_j is located on B_y.

Where $max\{T(s_i), T(s_j)\}$ denotes taking the larger value between $T(s_i)$ and $T(s_j)$, $max\{T(s_i) + T(B_k, B_x), T(s_j) + T(B_p, B_y)\}$ represents taking the larger value between $T(s_i) + T(B_k, B_x)$ and $T(s_j) + T(B_p, B_y)$, $min\{T(s_i), T(s_j)\}$ refers to taking the smaller value between $T(s_i)$ and $T(s_j)$, and $min\{T(s_i) + T(B_k, B_x), T(s_j) + T(B_p, B_y)\}$ denotes taking the smaller value between $T(s_i) + T(B_k, B_x)$ and $T(s_j) + T(B_p, B_y)$.

Definition 6 (Calculation of availability for selecting services s_i and s_j)

$$Av(s_i; s_j) = Av(s_i) * Av(s_j) \tag{8}$$

$$Av(s_i | s_j) = Av(s_i) * Av(s_j) \tag{9}$$

$$Av(s_i || s_j) = max\{Av(s_i), Av(s_j)\} \tag{10}$$

$max\{Av(s_i), Av(s_j)\}$ refers to taking the larger value between $Av(s_i)$ and $Av(s_j)$.

Definition 7 (Calculation of the QoS value normalization)

Because different non-functional attributes have different units, it is necessary to standardize them when measuring the values of different attributes, that is, normalization [27]. In this paper, the values of negative attributes are normalized according to equation (11). The values of positive attributes are normalized according to equation (12).

$$Norm(p) = \begin{cases} \dfrac{p^{max} - p}{p^{max} - p^{min}}, & \text{if } p^{max} \neq p^{min} \\ 1, & \text{if } p^{max} = p^{min} \end{cases} \tag{11}$$

$$Norm(p) = \begin{cases} \dfrac{p - p^{min}}{p^{max} - p^{min}}, & \text{if } p^{max} \neq p^{min} \\ 1, & \text{if } p^{max} = p^{min} \end{cases} \tag{12}$$

where p^{max} and p^{min} respectively represent the maximum and minimum values of attribute p. After normalization,

the attribute values of the service are within the range of [0,1] and can be uniformly processed thereafter.

For example, suppose we have a dataset containing two types of QoS: response time and availability. Our goal is to normalize them so that their values range from 0 to 1. First, we need to find the minimum and maximum values for response time and availability. Supposing the minimum response time is 42ms, and the maximum response time is 3347ms; the minimum availability is 12%, and the maximum availability is 100%. Next, we normalize for 246ms response time and 88% availability. Since response time is a negative attribute, according to formula (11), the normalized response time for 246ms is (3347-246)/(3347-42)=0.94. The smaller the response time, the larger the normalized value. Since availability is a positive attribute, according to formula (12), the normalized availability for 88% is (88-12)/(100-12)=0.86. The greater the availability, the larger the normalized value. Through this normalization process, we scale the values of different attributes to range from 0 to 1, and the higher the value, the higher the quality of service. This allows for better comparison and comprehensive analysis, avoiding biases caused by differences in numerical ranges.

Definition 8 (Utility value of the service)

The utility value of a service is the final evaluated value of the quality of service. The formula is as follows:

$$F = \sum_{i=1}^{v} Norm(p_i) \times w_i \tag{13}$$

Where v represents the number of attributes. $Norm(p_i)$ represents the normalized value of the i^{th} attribute, and w_i represents the weight of the i^{th} attribute, satisfying the condition $\sum_{i=1}^{v} w_i = 1$. As mentioned earlier, this paper considers three attributes: price, response time, and availability. Additionally, their weights are all set to 1/3 in the experiments.

Definition 9 (Service request) The service request A consists of six components: $\{K, M, Q, C, N_O, t_{limit}\}$. K is a set of tasks, denoted as $K = \{k_1, k_2, \ldots, k_{Task}\}$, with a total quantity of $Task$. M defines the structure of the tasks. C stands for the maximum cost that the service requester is willing to accept. N_O indicates the minimum number of service selection solutions desired by the requester. Lastly, t_{limit} specifies the longest time the service requester is prepared to wait for service selection solutions.

In addition, the variable t_{real} represents the real waiting time for the service requester to obtain solutions for service selection.

Definition 10 (Service selection solution provider) The provider of service selection solutions is referred to as $sssp$. $SSSP = \{sssp_1, sssp_2, \ldots, sssp_i, \ldots, sssp_{|SSSP|}\}$ denotes the collection of service selection solution providers, where $sssp_i$ indicates the i^{th} provider within the set.

Definition 11 (Name of the service selection solution) The name of the service selection solution is denoted as n. The set of candidate solutions CS comprises all the solution names that can be offered by providers of service selection solutions, represented as $\{n_1, n_2, \ldots, n_i, \ldots, n_{|CS|}\}$, where n_i denotes the name of the i^{th} solution.

Definition 12 (Service selection solution provided by the provider) The service selection solution provided by the provider is denoted as $R = \{del_1, r_1, del_2, r_2, \ldots, del_i, r_i, \ldots, del_{Task}, r_{Task}\}$, where r_i denotes the service chosen for the i^{th} task, del_i represents the time of data transfer between the i^{th} service and the server where the predecessor service is located, and the value of del_1 is 0.

R is the solution submitted by the service selection solution provider through the smart contract.

Definition 13 (Service selection solution processed by the smart contract) The service selection solution processed by the smart contract, referred to as o, consists of six elements: $\{AddrSSS, Cost_S, t_{RT}, a, H, ID_o\}$. $AddrSSS$ represents the provider's address for the solution, $Cost_S$ denotes the cost of the selected services, t_{RT} indicates the total response time of the selected services, a represents the overall

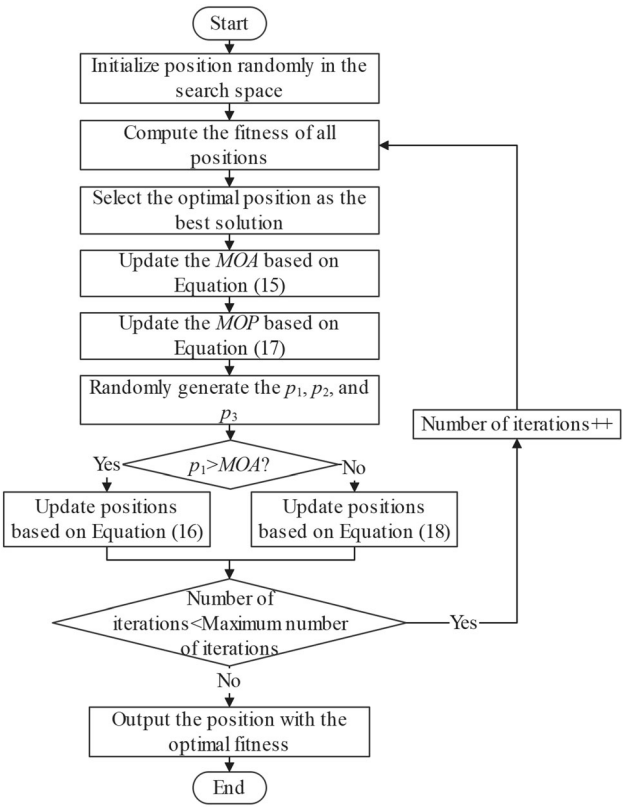

Fig. 4 The flow chart of the AOA

availability of the selected services, H signifies whether the solution meets the requester's cost requirement, and ID_o is a set of selected service IDs within the solution.

The collection of service selection solutions executed by the smart contract is represented as $O = \{o_1, o_2, \ldots, o_i, \ldots, o_{|O|}\}$. Each o_i denotes the i^{th} solution for service selection processed by the smart contract.

Once the service selection solution provider submits the solution R, the smart contract initiates a sequence of processing. The processed solution is represented as o.

Definition 14 (Optimal solution) The optimal solution is represented by $OptrR = \{addrC_1, addrC_2, \ldots, addrC_i, \ldots, addrC_{Task}\}$. Each $addrC_i$ denotes the address of the contract corresponding to chosen service for the i^{th} task in the final outcome.

Definition 15 (Total cost incurred by the service requester for invoking services)

$$Cost_{total} = Cost_P + Cost_{SSS} + Cost_S \tag{14}$$

Where the cost that needs to be paid by the requester to the smart contract provider for the provided smart contract is denoted as $Cost_P$, the cost that needs to be paid by the requester to the optimal service selection solution provider is represented as $Cost_{SSS}$, and $Cost_S$ represents the cost of the selected services.

3.4 Metaheuristic algorithms

Metaheuristic algorithms are a method for searching for the optimal solution in a specific space in a spontaneous, efficient, and flexible manner. They are used to tackle combinatorial optimization problems and typically aim to find solutions within a given time frame. The core idea of metaheuristic algorithms is to mimic heuristic processes in nature, such as biological evolution, simulated annealing, social behavior, etc., and abstract these processes into a problem-solving approach. By introducing randomness, diversity, and fitness evaluation, metaheuristic algorithms can explore complex search spaces to find solutions that meet specific objectives. These algorithms have excellent applications in various fields, providing powerful tools for solving practical problems. Next, some common algorithms are introduced.

3.4.1 Arithmetic optimization algorithm

The AOA is a novel metaheuristic algorithm proposed by Abualigah et al. in 2021 [10]. The AOA is characterized by its fast convergence speed and high precision. The inspiration for this algorithm's design comes from the concept of arithmetic operations in the field of mathematics. In the AOA, multiplication and division operations are used to enhance the breadth of global search, while addition and subtraction operations are employed to improve the accuracy of local search. This enables the algorithm to effectively iterate through the solution space of the problem in search of optimized solutions. Each position represents a solution, and the position is updated after each iteration.

The selection of exploration and exploitation stages is determined by the mathematical optimizer acceleration function MOA, and the random number $p_1 \in [0, 1]$. MOA is computed according to equation (15):

$$MOA = Min + C\_Iter \times \left(\frac{Max - Min}{M\_Iter}\right) \tag{15}$$

where C_Iter represents the current iteration number, Min and Max denote the minimum and maximum values of MOA, and M_Iter is the maximum number of iterations. Following the suggestion of Abualigah et al., this paper sets Min and Max to be 0.2 and 0.9, respectively.

When $p_1 gt MOA$, AOA switches to the exploration stage. The following are the updating rules for the positions x_i in the exploration stage:

$$x_i = \begin{cases} x_{best} \div (MOP + \epsilon) \times ((UB - LB) \times \mu + LB), & \text{if } p_2 < 0.5 \\ x_{best} \times MOP \times ((UB - LB) \times \mu + LB), & \text{otherwise} \end{cases} \tag{16}$$

where x_{best} represents the best position obtained so far, ϵ is a very small value to prevent division by zero, UB and LB represent the upper and lower bounds of the positions, μ is a control parameter for adjusting the search process, set to 0.499, and $p_2 \in [0, 1]$ is a random number. MOP is calculated based on equation (17):

$$MOP = 1 - \frac{C\_Iter^{1/\alpha}}{M\_Iter^{1/\alpha}} \tag{17}$$

The parameter α is set as 5, representing the sensitivity factor.

When $p_1 \leq MOA$, AOA enters the exploitation stage. The following are the updating rules for the positions x_i in the exploitation stage:

$$x_i = \begin{cases} x_{best} - MOP \times ((UB - LB) \times \mu + LB), & \text{if } p_3 < 0.5 \\ x_{best} + MOP \times ((UB - LB) \times \mu + LB), & \text{otherwise} \end{cases} \tag{18}$$

where the random number $p_3 \in [0, 1]$.

The flowchart of the AOA algorithm is shown in Fig. 4.

3.4.2 Differential evolution

The DE is an optimization algorithm that are employed to tackle a range of intricate issues [11]. In 1995, Rainer Storn

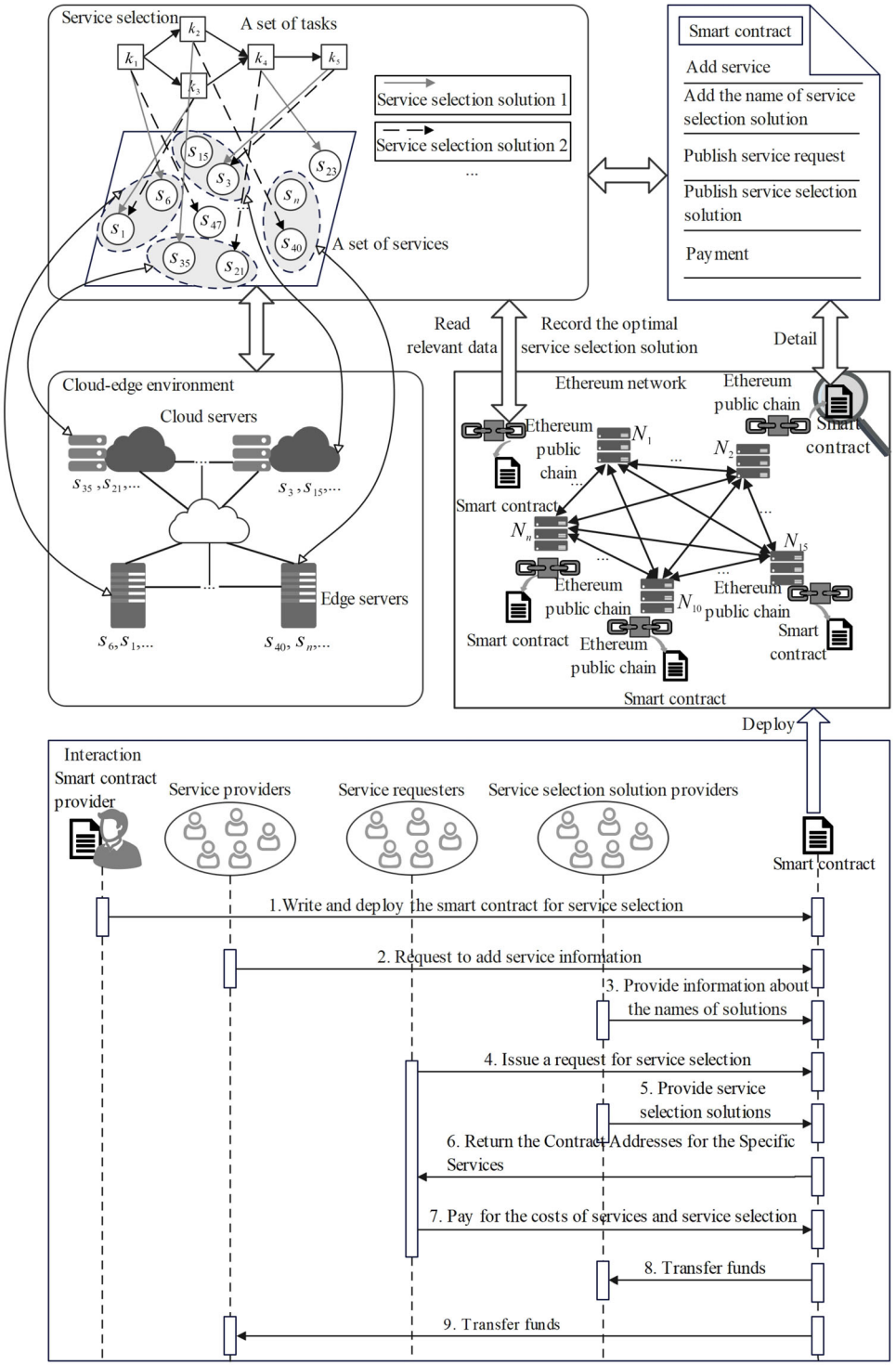

Fig. 5 Architecture of service selection based on Ethereum smart contracts in cloud-edge environment

and Kenneth Price proposed this method. The fundamental concept behind the DE is to explore the solution space by imitating the mechanisms of natural selection and mutation in order to find the most optimal solution. Each solution is represented as an individual, and the algorithm generates new individuals by carrying out mutation and crossover operations.

To obtain the mutated individual Y_i for each individual Z_i, equation (19) is utilized.

$$Y_i = Z_a + F(Z_b - Z_c) \tag{19}$$

The individuals denoted as Z_a, Z_b, and Z_c represent the a^{th}, b^{th}, and c^{th} individuals, respectively. It is important to note that i, a, b, and c are all distinct from each other. The mutation factor, denoted as F, has a range of values between 0 and 1.

To obtain the crossover individual X_i, equation (20) is used with the original individual Z_i and the mutated individual Y_i as inputs.

$$X_i(j) = \begin{cases} Y_i(j), & \text{if } rand(0,1) < W \text{ or } j = j_{random} \\ Z_i(j), & \text{otherwise} \end{cases} \tag{20}$$

The variable j is a number of the range [1, dim], where dim represents the dimension of the problem, which is in relation to the *Task* as defined in Definition 3. The value $rand(0, 1)$ refers to a random number that falls within the range of [0,1]. The symbol W represents the crossover rate, which is also within the range of [0,1]. Finally, j_{random} denotes a random integer chosen from the range [1, dim].

3.4.3 Other metaheuristic algorithms

Genetic Algorithm [16]: It is an optimization algorithm that simulates the natural evolutionary process. It gradually improves solutions from a population through operations such as selection, crossover, and mutation, and is widely used to solve complex problems. The GA possesses good adaptability and parallel performance, enabling it to find optimal solutions in solution spaces.

Moth-Flame Optimization (MFO) [28]: The MFO is a metaheuristic algorithm based on insect behavior, simulating the behavior of moths seeking light sources. By tracking and adjusting paths, moths search for the optimal solution. The MFO strikes a balance between local and global search, enabling rapid localization and convergence in the search space.

Northern Goshawk Optimization (NGO) [29]: The NGO is a bio-inspired algorithm inspired by the hunting behavior of the northern goshawk. By simulating the search strategy of the goshawk, it optimizes solutions. The NGO possesses strong search capabilities and adaptability, making it suitable for complex optimization problems.

Particle Swarm Optimization (PSO) [30]: It is an optimization algorithm that simulates the behavior of bird flocks or fish schools. Individuals adjust their positions based on individual experiences and group information to search for the optimal solution in the solution space. It is simple to implement and has a wide range of applicability.

4 Method for service selection based on blockchain smart contracts in cloud-edge environment

This section describes an overview of the method presented, introducing the approach for choosing services using blockchain smart contracts in the cloud-edge environment from three perspectives: the overall structure, the smart contract algorithm, and the execution process, along with the DEAO.

4.1 Architecture for service selection based on blockchain smart contracts in cloud-edge environment

We propose an architecture for service selection based on Ethereum smart contracts in the cloud-edge environment, as shown in Fig. 5. This architecture leverages the Ethereum platform, with Ethereum 2.0 adopting a PoS consensus mechanism. The entire architecture consists of seven modules, including smart contract provider, Ethereum network, smart contract, cloud-edge environment, service provider, service requester, and service selection solution provider. Below, we introduce the functions of these modules one by one.

1. Module for smart contract provider. The provider of the smart contract creates a contract specifically designed for selecting services in the cloud-edge environment and then proceeds to deploy it on the Ethereum network. If a service requester utilizes the service that was chosen through this particular smart contract, the provider of the said contract will attain a profit.
2. Module for Ethereum network. The Ethereum network functions as a decentralized system, with Ethereum nodes spread out across numerous servers. These nodes are connected through a peer-to-peer network and work together to uphold the complete Ethereum public blockchain. When users wish to utilize the features of a smart contract, they must interact with the smart contract via an Ethereum node's interface in order to obtain the desired outcome. In the Ethereum network, nodes vie for the privilege of recording transactions. Whenever a smart contract is triggered, the node authorized to record generates a new block and logs the

outcome of the execution on the blockchain. This block is subsequently disseminated to other nodes, which execute the smart contract and authenticate the data. Once verified, the block is added to their respective copies of the blockchain. These data possess properties of public transparency and immutability, ensuring that participants, including service requesters, service providers, and solution providers for service selection, cannot refute the events that have taken place. This mechanism guarantees the reliability of the transactions.

3. Module for smart contract. The smart contract is applied to service selection in a cloud-edge environment, with participants including service requesters, service providers, and solution providers of service selection. The smart contract assists participants in completing the process of service selection. All pertinent transactions are documented on the Ethereum blockchain and can be accessed through it.
4. Module for cloud-edge environment. There are multiple cloud servers and edge servers, which are interconnected and capable of data transmission. Services are distributed across the network of cloud and edge servers. The connection information between edge servers and cloud servers is known to the service selection solution providers.
5. Module for service provider. Service providers can offer multiple services. When a service provider's service is selected by the best solution and the service requester adopts that solution, the service provider receives revenue.
6. Module for service requester. The service requesters aim to fulfill a set of tasks by obtaining a group of services that meet both functional requirements and optimal QoS. The requesters submit requests, receive the optimal services, and pay the corresponding fees.
7. Module for service selection solution provider. In service selection, multiple service selection solutions may be obtained. The necessary data for this process is accessed via the smart contract on the blockchain. Once the selection process is finished, the best service selection solution is made public through the smart contract on the blockchain. Multiple service selection solution providers compete for the best solution. This mechanism incentivizes service selection solution providers to adopt more efficient service selection algorithms, ultimately enabling them to choose higher-quality services for service requesters.

Next, we introduce the interaction between the smart contract provider, service providers, service requesters, service selection solution providers, and the smart contract.

1. Smart contract provider writes and deploys the smart contract for service selection in a cloud-edge environment.
2. Service providers submit information about their services by triggering a function in the smart contract.
3. Service selection solution providers submit the names of their service selection solutions through the smart contract, allowing service requesters to query the maximum number of possible service selection solutions.
4. Service requesters retrieve the number of service selection solutions and publish service requests through the smart contract.
5. Based on the requests from service requesters, service selection solution providers determine service selection solutions and submit them through the smart contract.
6. Service selection solution providers view the service request and extract information on corresponding candidate services through the smart contract, determine the service selection solution, and submit the solution via the smart contract.
7. When the number of service selection solutions meets the requirements or exceeds the time limit, the service requester that is currently issuing the request obtains the contract address of the best services through the smart contract for the use of the specific services.
8. If service requesters decide to use the selected services, they pay for the service usage and selection fees via the smart contract.
9. The provider offering the optimal service selection solution receives the transfer from the smart contract.
10. The providers of the selected services in the best solution receive the transfer from the smart contract.

4.2 Algorithm and process for service selection based on blockchain smart contracts in cloud-edge environment

Using the aforementioned architecture as a basis, we design a smart contract that aids users in their pursuit of service selection within the cloud-edge environment. The smart contract is elucidated through Algorithm 1.

Algorithm 1 Service selection based on smart contracts in cloud-edge environment

Input: CS: a collection of candidate service selection solutions
S: a collection of services
A: a service request
R: a collection of service selection solutions
Output: the optimal service selection solution $OptrR$

1: **for** each $sp_i \in SP$ **do**
2: $S \leftarrow S \cup AddServ(S_i^{AddrS}, S_i^{AddrC}, S_i^B, S_i^{Fs}, S_i^Q).\ S_i$
3: $ID \leftarrow ID \cup AddServ(S_i^{AddrS}, S_i^{AddrC}, S_i^B, S_i^{Fs}, S_i^Q).ID_i^{sp}$
4: **end for**
5: **for** each $sssp_i \in SSSP$ **do**
6: $CS \leftarrow CS \cup SolutionName(n_i)$
7: **end for**
8: $Task, M, C, N_O, t_{limit} \leftarrow Demand(A^{Task}, A^M, A^C, A^{N_O}, A^{t_{limit}})$
9: **for** $i = 1 : Task$ **do**
10: $K \leftarrow K \cup FuncDemand(k_i)$
11: **end for**
12: **for** each $sssp_i^U \in SSSP^U$ **do**
13: **for** $j \leftarrow 1$ **do**
14: $O \leftarrow O \cup Solution(j, R_j^r, R_j^{del})$
15: $j \leftarrow j + 1$
16: **if** $j == Task + 1$ **then**
17: break
18: **end if**
19: **end for**
20: **end for**
21: **if** $|O| \geq N_O$ || $t_{actual} > t_{limit}$ **then**
22: $OptrR \leftarrow OptrR \cup OptimalSolution(i)$ //The i^{th} solution is the best solution
23: $U \Longrightarrow Cost_{total}$, $Cost_{total}$ is calculated based on equation (14)
24: **for** $i \leftarrow 1$ **do**
25: $AddrS \Longleftarrow Cost$
26: $i \leftarrow i + 1$
27: **if** $i == Task + 1$ **then**
28: break
29: **end if**
30: **end for**
31: $AddrSSS \Longleftarrow Cost_{SSS}$
32: **end if**

The service provider *sp* adds services and their corresponding IDs to sets *S* and *ID* by calling the smart contract function *AddServ* (lines 1-4). The service selection solution provider *sssp* adds the name of the service selection solution *n* by calling function *SolutionName* (lines 5-7). Therefore, the service requester *U* can view the existing number of solutions. *U*, by calling *Demand*, adds the number of tasks *Task*, the structure of task composition *M*, the highest acceptable cost *C*, the minimum number of solutions N_O, and the maximum waiting time for solutions t_{limit} (line 8), and then adds all tasks to the set *K* by calling *FuncDemand* (lines 9-11). The service selection solution provider *sssp* reviews *A* and candidate services. After obtaining a solution, it submits *R* through *Solution*, which is then processed and added to the collection *O* (lines 12-20), described by Algorithm 2. Until $|O|$ is no less than N_O, or t_{actual} is greater than t_{limit}, *OptimalSolution* is automatically called to record the optimal selection *OptrR* (lines 21-

Table 2 Functions

Function name	Function description
AddServ	input:s^{AddrS}; s^{AddrC}; s^{B}; s^{F_S}; s^{Q}
	procedure:$ID \leftarrow ID \cup id$, $S \leftarrow S \cup s$, $Array_{Cost} \leftarrow Array_{Cost} \cup Q.Cost$, $Array_T \leftarrow Array_T \cup Q.T$, $Array_{Av} \leftarrow Array_{Av} \cup Q.Av$ (to facilitate the reading of QoS for service selection solution providers, $Array_{Cost}$, $Array_T$, and $Array_{Av}$ are saved)
SolutionName	input:n
	procedure:$N_C \leftarrow N_C \cup n$
Demand	input:A^{Task}, A^M, A^C, A^{N_O}, $A^{t_{limit}}$
	procedure:$Task \leftarrow Task$, $M \leftarrow M$, $C \leftarrow C$, $N_O \leftarrow N_O$, $t_{limit} \leftarrow t_{limit}$
FuncDemand	input:A^k
	procedure:$K \leftarrow K \cup k$
Solution	refer to Algorithm 2
OptimalSolution	input:i
	procedure:$OptrR \leftarrow OptrR \cup addrC_1 \cup addrC_2 \cup \ldots \cup addrC_{Task}$

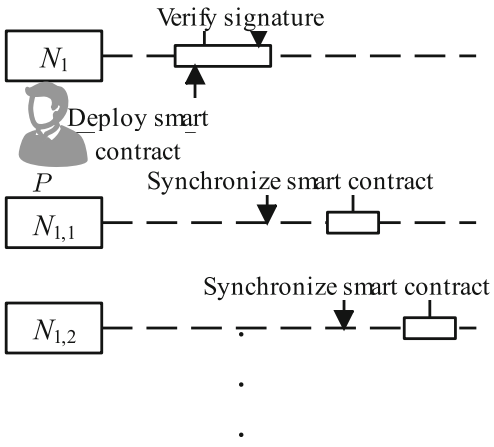

Fig. 6 Execution process of smart contract provider

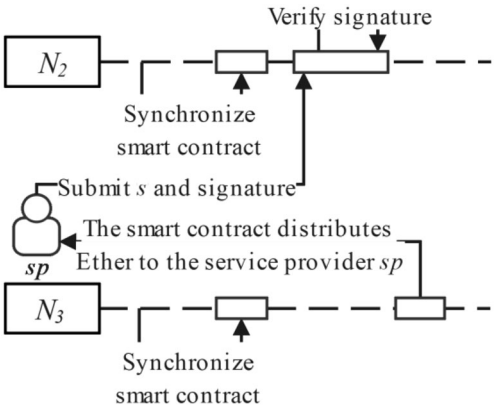

Fig. 7 Execution process of service provider

22). After requester U reviews $OptrR$, if he chooses to use the selected services, he transfers $Cost_{total}$ to the smart contract (line 23). Subsequently, the smart contract transfers the cost $Cost$ to the respective addresses $AddrS$ of the selected service providers (lines 24-30). Additionally, based on the negotiated cost $Cost_{SSS}$, the smart contract transfers funds to the provider $AddrSSS$ of the selected solution (line 31).

The functions involved in Algorithm 1 are shown in Table 2.

Algorithm 2 describes the function *Solution*. Based on the input values j and r, the unique ID id of the service is identified (line 1). The total cost $Cost_S$ of the selected services is calculated according to formulas (2), (3), and (4), and the total response time t_{RT} is computed based on del and formulas (5), (6), and (7). The availability a of the services is determined using formulas (8), (9), and (10) (lines 2–4). If the task is the last one, and the total cost $Cost_S$ of the selected services is less than the maximum acceptable cost C for the requester, the solution meets the requester's requirements; otherwise, it does not (lines 5–9).

Algorithm 2 Solution

Input: j: identification of the task
r: service selected for the j^{th} task
del: time for data transmission between the server hosting the service and the server hosting the preceding service

```
id ← ID(j, r)
calculate Cost_S based on formulas (2), (3), and (4)
compute t_RT based on formulas (5), (6), and (7)
determine a based on formulas (8), (9), and (10)
if j == Task&&Cost_S ≤ C then
    H ←true
else
    H ←false
end if
record solution o
```

We respectively introduce the processes for smart contract provider, service provider, service requester, and solution provider of service selection.

1. The execution process for the smart contract provider is illustrated in Fig. 6. The smart contract written by provider P needs to be compiled into blockchain-executable bytecode. Once the smart contract provider submits the transaction to deploy the contract, along with a digital signature, the transaction is placed in the pending transaction pool. Assuming node N_1 emerges victorious in the competition for the right to record transactions and is willing to process the transaction. After verifying the legality of the signature and the transaction, if everything is in order, it adds a new block containing the transaction for deploying the smart contract to its blockchain. Subsequently, it transmits the block to neighboring nodes $N_{1,1}$ and $N_{1,2}$. $N_{1,1}$ and $N_{1,2}$, after successful verification, similarly store and forward it to other nodes until all nodes synchronize this contract.
2. Fig. 7 illustrates the execution process for the service provider. After nodes synchronize the smart contract, the service provider *sp* submits a service *s* by invoking function *AddServ* of the smart contract, accompanied by a signature. Assuming node N_2 wins the right to the ledger and verifies and stores the transaction in its blockchain replica. The node broadcasts this transaction to other nodes, and eventually, the transaction is saved by all nodes in their blockchain replicas. If a service requester utilizes the service of *sp*, the smart contract distributes ether to *sp*. The transfer transaction is assumed to be initially recorded in the blockchain by the N_3 node that obtained the right to the ledger. Finally, all nodes record this transaction.

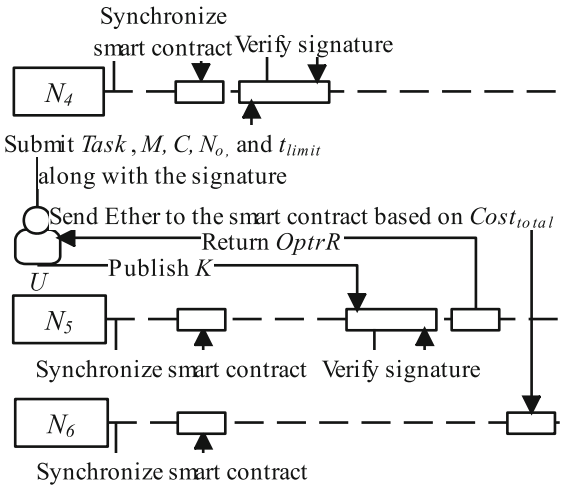

Fig. 8 Execution process of service requester

3. The execution process for the service requester is depicted in Fig. 8. After nodes synchronize the smart contract, the service requester U invokes function *Demand* of the smart contract to submit *Task*, *M*, *C*, N_O, and t_{limit}, along with a digital signature. Assuming the digital signature is valid, the transaction is presumed to be verified and packaged onto the blockchain by the N_4 node that wins the right to the ledger. Eventually, the transaction is broadcast and synchronized across the entire network. Subsequently, U, carrying a signature, publishes K by invoking *FuncDemand*. Assuming the node N_5, which obtained the right to the ledger, verifies the transaction, and packages it onto the blockchain, other nodes verify and update the data. Then, the service selection solution providers can examine all the information uploaded by

the requester. If the requester decides to use the services selected in the optimal solution, a transfer is made to the smart contract. This transaction is verified, packaged, and added to the blockchain replica by node N_6 which possesses the right to the ledger. It is then propagated to other nodes. Eventually, the transaction is broadcast and synchronized across the entire network.

4. Fig. 9 illustrates the execution process of the service selection solution provider. After nodes synchronize with the smart contract, the service selection solution provider *sssp* carries a signature and calls the *SolutionName* of the smart contract to publish the name *n* of the solution. Assuming the transaction is verified without errors, this solution is presumed to be packaged onto the blockchain by N_7, which has obtained the right to be recorded, and it is then verified and synchronized across the entire network. After the service requester publishes a request, *sssp* extracts relevant information for service selection and determines the optimal solution *R*. *sssp* then submits *R* by calling the *Solution* function of the smart contract. Assuming node N_8 possesses the right to record, N_8 verifies and stores the transaction in the blockchain replica, while other nodes validate and update data. If the service requester adopts this solution, *sssp* receives a transfer from the smart contract. The transaction is presumed to be packaged onto the blockchain by node N_9, assumed to have acquired the right to record, and is then verified and synchronized across the entire network.

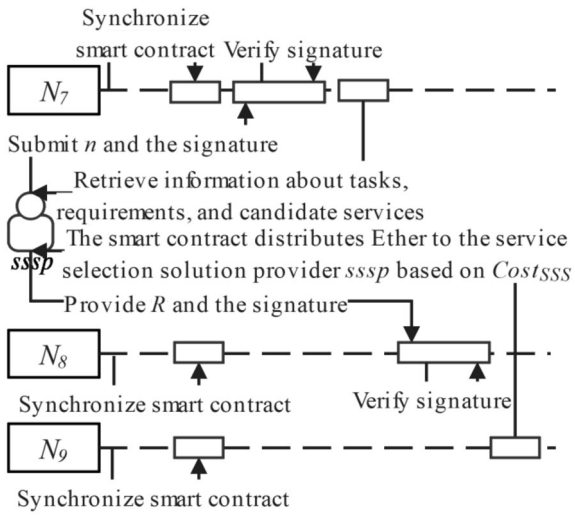

Fig. 9 Execution process of service selection solution provider

4.3 Differential evolution arithmetic optimization algorithm

The AOA possesses a faster convergence speed and effective global search capability. The algorithm demonstrates good computational performance and is adept at achieving high-quality solutions. However, the algorithm suffers from the drawback of premature convergence, which may be attributed to its limited exploration capability [31, 32]. The operations of mutation and crossover in the DE can improve the problem of premature convergence, whose effectiveness has been proven in other studies. Yan et al. applied the DE to enhance the MFO, addressing the early convergence and local optima issues of multi-objective models [5]. Debnath et al. hybridized the Dragonfly Algorithm with the DE, increasing the probability of obtaining globally optimal solutions [33]. Tang et al. suggested integrating the PSO with the DE to evolve the individual optimal positions of particles, avoiding stagnation issues [34]. Additionally, there has been no study introducing the crossover and mutation operations of the DE into the AOA. Therefore, we propose the DEAO algorithm, which combines the crossover and mutation operations of the DE with the AOA. Algorithm 3 explains this approach.

Each individual's position in Algorithm 3 represents a service selection solution. Altering the dimensions of the position vector allows individuals to explore solutions in different dimensional spaces. The number of tasks *Task* in the service selection problem is represented by the dimensions of the position vector in Algorithm 3. The service selection solution $\{r_1, r_2, \ldots, r_{Task}\}$ submitted by the service

Table 3 Configuration of Ethereum nodes

Configuration	Node 1	Node 2	Node 3–5	Node 6–8
Processor	AMD Ryzen 7 5800 H 3.20GHz	Inter core i5-10400F 2.90GHz	Intel Core i7-7700HQ 2.80GHz	Intel Core i9-9900K 3.60GHz
Internal memory	16 GB	16 GB	8 GB	32 GB
Operating system	Windows 10 Home	Windows 11 Professional	Windows 10 Enterprise	Windows 10 Enterprise
IP address	192.168.1.106	192.168.1.112	192.168.1.113–115	192.168.1.101–103
Geth	v1.10.25	v1.10.25	v1.11.5	v1.11.5

selection solution provider is represented as x_{best}. If the candidate service IDs start from 1, then *LB* is 1. The number of candidate services corresponding to each task is denoted as *UB*. The utility value of the selected services in the solution, calculated according to formula (13), serve as the fitness for each individual. A higher fitness indicates a better position for the individual in the solution space.

In each iteration of the algorithm, individuals update their positions through arithmetic operations (lines 11-20). Following this, for each individual, three individuals are randomly selected excluding the individual itself. Then, mutation operation is performed on the selected individual according to formula (19) to obtain mutated individual (line 22). Subsequently, crossed individual is obtained according to formula (20) (line 23). Then, individuals are updated to the better position between the original position and the position after crossover (lines 25-27). Finally, the best fitness position obtained after a certain number of iterations is considered the optimal solution (line 28).

Algorithm 3 Solution

Input: $Task$: number of tasks
M: structure composition of tasks
$S_A = \{s_1, s_2, ..., s_k\}$: candidate services
Output: optimal solution $OptrR$

1: Initialize UB, LB, M_Iter, and the number of individuals N_d
2: Initialize Min, Max, ϵ, μ, and α
3: Randomly initialize the positions of N_d individuals
4: **for** $iteration \leftarrow 1$ **do**
5: **if** $iteration$==1 **then**
6: Calculate the fitness of all individuals based on formula (13)
7: Select x_{best} as the current best solution
8: **end if**
9: Update MOA according to equation (15)
10: Update MOP based on equation (17)
11: **for** i=1:N_d **do**
12: $p_1 \leftarrow \text{Rand}(0, 1)$
13: **if** $p_1 > MOA$ **then**
14: $p_2 \leftarrow \text{Rand}(0, 1)$
15: Individual Z_i is updated according to equation (16)
16: **else**
17: $p_3 \leftarrow \text{Rand}(0, 1)$
18: Individual Z_i is updated based on equation (18)
19: **end if**
20: **end for**
21: **for** i=1:N_d **do**
22: Individual Z_i undergoes mutation according to formula (19), generating mutated individual Y_i
23: Y_i undergoes crossover according to formula (20), resulting in the crossed individual X_i
24: **end for**
25: Merge the individuals and the crossed individuals
26: Sort the merged fitness values
27: Take the top N_d positions in terms of fitness ranking from the merged positions as the new positions for the individuals
28: Select x_{best} as the current best solution
29: $iteration \leftarrow iteration$+1
30: **end for**
31: **return** x_{best}

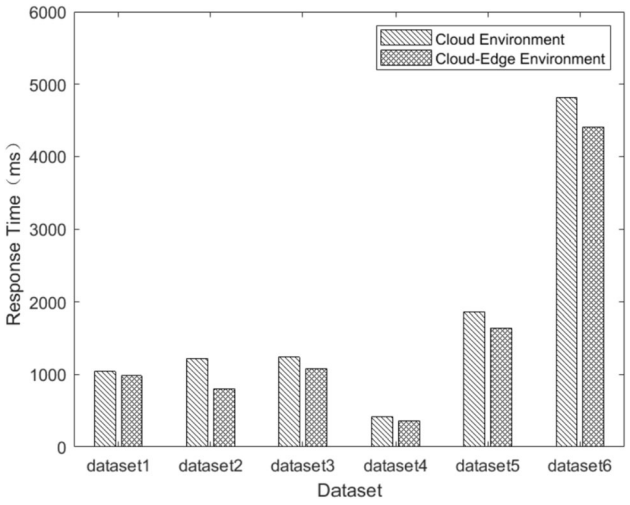

Fig. 10 Response time in different environments

5 Experimental results and discussion

We employ two different methods to address the service selection problem: one is the traditional approach, and the other is our proposed blockchain-based approach. The traditional approach utilizes the programming language Golang [35], with the selected integrated development environment being Goland [36]. We choose Ethereum as the foundation to validate the effectiveness of our proposed blockchain-based method. We implement our smart contract in the Remix integrated development environment [37] using the Solidity language [38] and deploy it on the local Ethereum blockchain. To interact with this smart contract, we use the Geth client [39]. We configure 8 Ethereum nodes in our local blockchain, with specific configuration details outlined in Table 3. Additionally, the algorithms for service selection employed in both approaches are executed using Golang.

5.1 Experimental settings

We assume there is a service requester needing to complete a set of tasks. The requester has two options: submitting the request traditionally or issuing the request through a smart contract. For each task, there are multiple candidate services available for selection. Once the optimal service selection solution is determined, the service requester can adopt that solution.

To assess the suggested approach, this research paper utilizes two collections of data. The first set, comprising datasets 1–3, consists of three datasets with 15, 75, and 120 candidate services for each task, and each dataset contains 10 tasks. The second set includes datasets 4–6, with three datasets having 5, 15, and 30 tasks for each, and each task has 75 candidate services. The QoS of services in each dataset takes into consideration cost *Cost*, response time *RT*, and availability *Av*, where *RT* and *Av* are derived from the Quality of Web Service (QWS) dataset [40]. The QWS comes from the paper by Al-Masri and Mahmoud in 2007, with its 2.0 version having QoS data for 2507 real web services. The response time ranges from 8 to 4637 ms, and the availability ranges from 12 to 100%. Additionally, randomly generate *Cost* within the range [10, 130]. Based on the response time and availability of the QWS, along with randomly generated cost, we create virtual services. These services are randomly deployed across 2 cloud servers and 4 edge servers. The data transfer times are 1-3ms between interconnected edge servers, 20-50ms between cloud servers, and 10–50 ms between cloud and edge servers.

5.2 Performance evaluation

In terms of computational complexity, Table 4 documents the gas [41] costs when users invoke functions in the smart contract. In Ethereum, gas is a unit of measurement used to quantify the computational resources and resource consumption required for operations on the network. Transaction cost refers to the amount of gas needed to execute a transaction, and execution cost refers to the amount of gas required to run a smart contract on the Ethereum network. In Table 4, we observe that the functions *AddServ* and *Solution* incur higher gas costs in both transaction and execution. This is because the *AddServ* function involves storing a larger number of variables related to the service information, and each of the QoS attributes needs to be stored separately to facilitate retrieval by service selection solution providers. Additionally, the *Solution* function requires calculating the total cost, total response time, and total availability of the selected services. Compared to other functions, these two functions have a higher complexity, leading to higher gas costs.

In the aspect of information tampering prevention, we conduct experiments at different time points, t_1, and t_2, comparing the performance of service selection using the traditional method and the blockchain-based method. The experiment uses dataset 1 and employs the proposed DEAO for service selection. The results of the experiment are shown in Table 5. In the traditional method, the DEAO offers the optimal solution at moments t_1 and t_2. It is noteworthy that, although the maximum utility values of the services selected by the DEAO at t_2 are greater than those at t_1, the actual utility values of the selected services at t_2 are less than those at t_1. This is because, between t_1 and t_2, information regarding the prices, response time, and availability of certain services has been tampered with. Consequently, the first, fifth, and seventh services selected in the optimal solution at t_2 are tampered services. Service

Table 4 Gas cost of users calling different functions

Gas consumption	AddServ	SolutionName	Demand	FuncDemand	Solution
Transaction cost/gas	334248	50374	28962	45435	206911
Execution cost/gas	311788	28878	7478	23763	185439

Table 5 Service selection of traditional and blockchain-based methods at times t_1 and t_2

Method	Time	Optimal solution	Maximum utility value obtained	Actual utility value
Traditional	t_1	[6, 9, 10, 3, 8, 7, 7, 10, 1, 11]	0.84	0.84
	t_2	$[0^{[1]}, 9, 0, 3, 5^{[1]}, 7, 3^{[1]}, 10, 1, 11]$	0.86	0.76
Blockchain-based	t_1	[6, 9, 10, 3, 8, 7, 7, 10, 1, 11]	0.84	0.84
	t_2	[6, 9, 10, 3, 8, 7, 7, 10, 1, 11]	0.84	0.84

[1]When using the traditional method, services with tampered *Cost*, *RT*, and *Av* between t_1 and t_2: the candidate services with ID 0 in task 1, ID 5 in task 5, and ID 3 in task 7

requesters are not utilizing the optimal services when using the services. However, our blockchain-based approach leverages the tamper-resistant characteristics of blockchain, yielding significantly different results. When using the blockchain-based method, the DEAO also provides optimal solutions at moments t_1 and t_2. It is noteworthy that the maximum utility value at t_1 is equal to the maximum utility value at t_2. More importantly, the actual utility values align perfectly with the results provided by the DEAO. This indicates that no information tampering occurred between moments t_1 and t_2. Therefore, the service requester uses the optimal services when utilizing the selected services.

In the context of environments, we conducted experiments in two scenarios, cloud environment, and cloud-edge environment, comparing the response time of selected services in 6 datasets. The experiment employed the proposed DEAO for service selection. Figure 10 illustrates the response time of selected services in both cloud and cloud-edge environments across the 6 datasets. It is evident that, in each dataset, the response time of selected services in the cloud-edge environment is consistently lower than those in the cloud environment. This is because edge computing shifts computational capabilities from the cloud to edge servers, thereby reducing service response time.

In the aspect of service selection algorithms, we compare the proposed DEAO with the widely used GA, MFO, PSO, and the newer AOA, NGO.

Firstly, we evaluate the performance of the algorithms, focusing on diversity, convergence, and comprehensive index. We use metrics such as Spacing [42], Generational Distance (GD) [43], and Inverse Generational Distance (IGD) [44] to objectively assess the algorithms' performance in different aspects.

Spacing is a metric used to quantify the diversity of solutions by examining the distribution of solutions generated by an algorithm in the solution space. A smaller Spacing value indicates a better distribution of the solution set. The formula for calculating Spacing is as follows:

$$\text{Spacing} = \sqrt{\sum_{i=1}^{m} \frac{(d_{1,i} - \bar{d_1})^2}{m-1}} \tag{21}$$

where m represents the number of solutions within the set addressed by the algorithm, $d_{1,i}$ represents the Manhattan distance between the i^{th} solution and its nearest neighbor, and $\bar{d_1}$ is the mean of all $d_{1,i}$ distances.

The GD is a metric used to measure the dissimilarity between the solution set generated by an algorithm and the true Pareto front. Generally, the lower the GD value, the better the convergence performance of the algorithm. The following is the formula for calculating the GD:

$$\text{GD} = \frac{\sqrt{\sum_{i=1}^{m} d_{2,i}^2}}{m} \tag{22}$$

where $d_{2,i}$ represents the Euclidean distance between the i^{th} generated solution and the nearest solution on the true Pareto front.

The IGD assesses the performance of an algorithm by measuring the average distance from each solution on the true Pareto front to the generated solutions. It simultaneously gauges the convergence and diversity of the solution set, with the lower IGD value indicating superior convergence and diversity. The following is the formula for calculating the IGD:

$$\text{IGD} = \frac{\sqrt{\sum_{i=1}^{|P|} d_{3,i}^2}}{|P|} \tag{23}$$

where P represents the true Pareto front, and $d_{3,i}$ is the Euclidean distance between the i^{th} solution on the true Pareto front and the nearest generated solution.

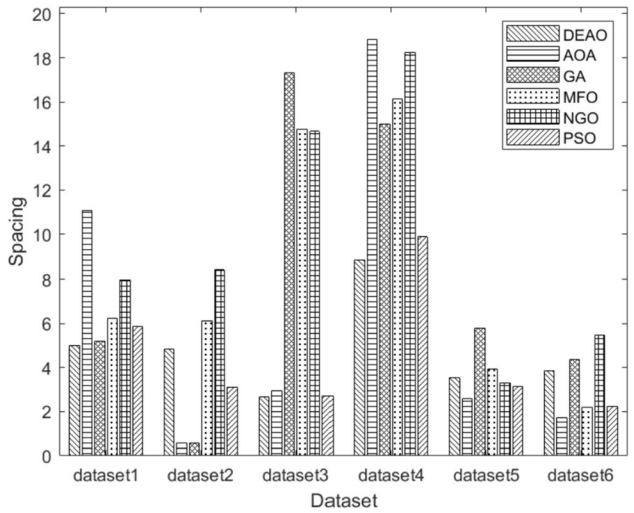

Fig. 11 Statistical results of the spacing of algorithms in different datasets

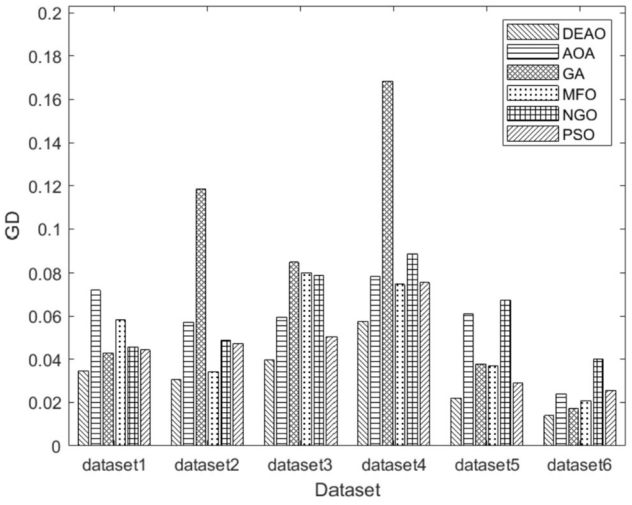

Fig. 12 Statistical results of the GD of algorithms in different datasets

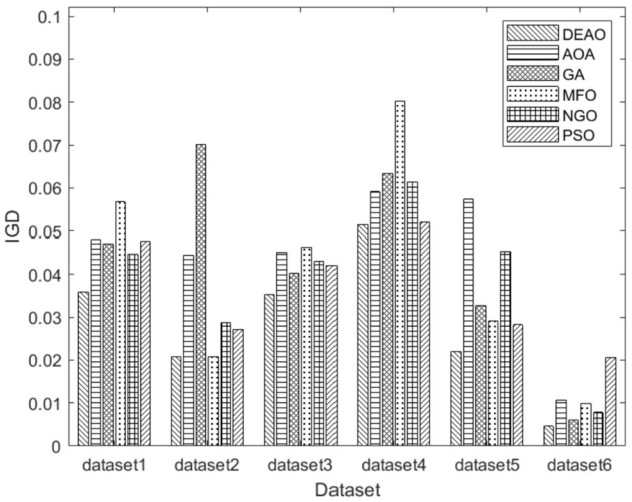

Fig. 13 Statistical results of the IGD of algorithms in different datasets

We take the Pareto solution sets obtained by the mentioned six algorithms on six datasets as the generated solutions. The results of the evaluations are as follows:

The SP, GD, and IGD of the algorithm are shown in Figs. 11, 12, and 13, respectively, on different datasets. From the graph 11, it can be observed that the DEAO has a relatively lower Spacing compared to the other five algorithms, indicating its superior solution distribution. However, the performance of the PSO's Spacing is lower than that of the DEAO in multiple datasets, and when the dataset is large, the AOA's Spacing is prone to be lower than that of the DEAO. Therefore, in terms of solution distribution, the PSO outperforms the DEAO, and the AOA surpasses the DEAO for larger datasets. This may be because the mutation and crossover operations of the DEAO sacrifice some distribution for the sake of enhancing its convergence. Furthermore, Figs. 12 and 13 indicate that the GD and IGD of the DEAO are lower than those of the AOA, GA, MFO, NGO, and PSO across all datasets. Therefore, the DEAO exhibits better convergence and overall performance. This suggests that the improvements made to the AOA have effectively enhanced the quality of solutions.

Finally, Fig. 14 illustrates the relationship between the utility values of selected services and the number of iterations for different algorithms across the six datasets. The utility values of services selected by the DEAO are consistently greater than those of the AOA, GA, MFO, NGO, and PSO. Specifically, across the six distinct datasets, the DEAO outperforms the AOA. This advantage stems from the introduction of mutation and crossover operations in the DEAO, enhancing the algorithm's search capabilities and effectively preventing premature convergence to local optima, thereby aiding in the discovery of superior solutions.

6 Conclusion

This paper proposes a service selection method based on blockchain smart contracts in the cloud-edge environment to address the challenge of information tampering faced by traditional service selection in the current cloud-edge environment. This method leverages blockchain technology to ensure a high level of transparency and tamper resistance to information of services. Specifically, the method makes it extremely difficult and costly for participants or attackers to tamper with transaction data, making it nearly impossible to achieve. This ensures the reliability of service selection in the cloud-edge environment. This provides participants with a trustworthy environment for service selection. The paper provides a detailed introduction to the algorithm of smart contracts, elucidating the

execution process for different users. To address the issue of local optimization caused by premature convergence, this method incorporates the mutation and crossover operations of the Differential Evolution into the Arithmetic Optimization Algorithm, creating a more exploratory Differential Evolution Arithmetic Optimization Algorithm. In addition, this paper assesses the computational complexity of the designed smart contract. Different methods are used in experiments to calculate the utility values of services, and the results show that, compared to traditional methods, the proposed approach effectively prevents information tampering with services. The convergence and overall performance of the proposed service selection algorithm surpass those of the Arithmetic Optimization Algorithm, Genetic Algorithm, Moth-Flame Optimization, Northern Goshawk Optimization, and Particle Swarm Optimization.

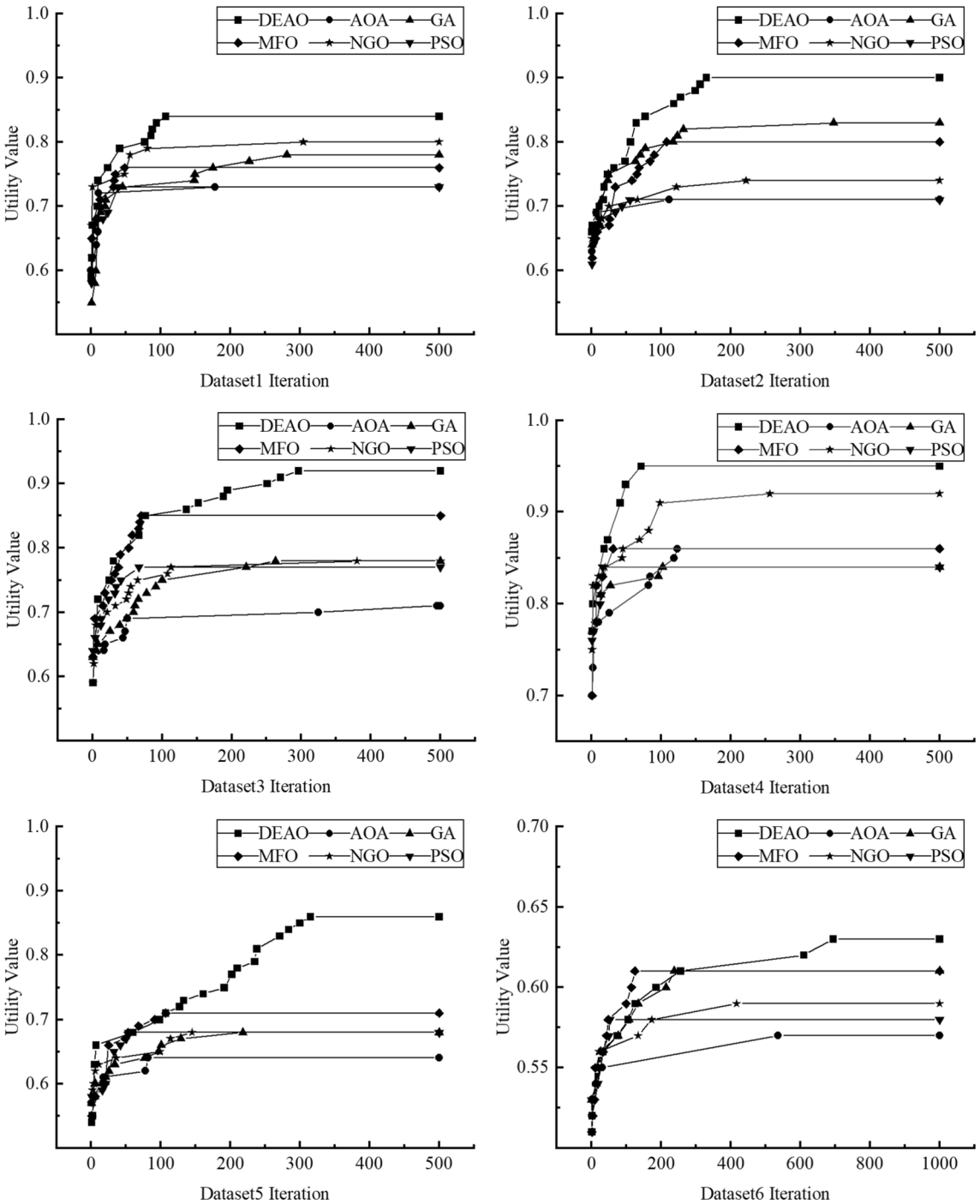

Fig. 14 Utility value and iterations

Moreover, the utility values obtained by the proposed algorithm are higher than those achieved by these algorithms.

In the future, the author will consider the practical implementation in real-world scenarios within the Internet of Things or smart city applications. The focus will be on applying the proposed method in the Internet of Things or smart city to validate its feasibility and practical effectiveness.

Acknowledgements This work was supported by the "Project of promoting scientific research cooperation and high-level talent with Canada, Australia, New Zealand, and Latin America" (Grant Number liujinmei [2023] No. 21) of the China Scholarship Council; and the China University Industry University Research Innovation Fund—new generation information technology innovation project (Grant Number 2022IT048) of the university science research and development center of the Ministry of education. Ming Zhu is the corresponding author.

Author contributions YN contributed to the design of the methodology and the initial drafting of the manuscript. JL acquired funding and contributed to the revision and editing of the manuscript. MZ conceived the idea, formulated objectives, and contributed to the revision and editing of the manuscript. CL prepared materials, collected data, and performed data analysis.

Funding This work was supported by the "Project of promoting scientific research cooperation and high-level talent with Canada, Australia, New Zealand, and Latin America" (Grant Number liujinmei [2023] No. 21) of the China Scholarship Council; and the China University Industry University Research Innovation Fund—new generation information technology innovation project (Grant Number 2022IT048) of the university science research and development center of the Ministry of education.

Data availibility The data of this study can be obtained from the corresponding author upon a reasonable request.

Declarations

Conflict of interest The authors have not disclosed any Conflict of interest.

References

1. Kong, X., Wu, Y., Wang, H., Xia, F.: Edge computing for internet of everything: a survey. IEEE Internet Things J. **9**(23), 23472–23485 (2022)
2. Alshareef, H.N.: Current development, challenges, and future trends in cloud computing: a survey. Int. J. Adv. Comput. Sci. Appl. **14**(3), 329–337 (2023)
3. Zhu, M., Yu, F., Yan, X., Li, J., Wang, Y.: Scaling up mobile service selection in edge computing environment with cuckoo optimization algorithm. In: 2021 IEEE International Conference on Services Computing (SCC), pp. 394–400 (2021). IEEE
4. Alrawais, A., Alhothaily, A., Hu, C., Cheng, X.: Fog computing for the internet of things: security and privacy issues. IEEE Internet Comput. **21**(2), 34–42 (2017)
5. Zhu, M., Yan, X., Li, J., Liu, C., Cao, Y.: Selecting mobile services in cloud and edge environment by moth-flame optimization algorithm. Int. J. Web Serv. Res. (IJWSR) **19**(1), 1–23 (2022)
6. Xie, Y., Zhu, Y., Wang, Y., Cheng, Y., Xu, R., Sani, A.S., Yuan, D., Yang, Y.: A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment. Future Gener. Comput. Syst. **97**, 361–378 (2019)
7. Wang, X., Ren, X., Qiu, C., Xiong, Z., Yao, H., Leung, V.C.: Integrating edge intelligence and blockchain: what, why, and how. IEEE Commun. Surv. Tutor. **24**(4), 2193–2229 (2022)
8. Lyu, Q., Rao, Y., Wang, J., Qi, P.: A blockchain-based manufacturing service composition architecture for trust issues. In: Dolgui, A., Bernard, A., Lemoine, D., Cieminski, G., Romero, D. (eds.) Advances in Production Management Systems Artificial Intelligence for Sustainable and Resilient Production Systems: IFIP WG 57 International Conference, pp. 70–80. Springer, Nantes (2021)
9. Sridevi, S., Karpagam, G., et al.: Investigation on blockchain technology for web service composition: a case study. Int. J. Web Serv. Res. (IJWSR) **18**(1), 70–93 (2021)
10. Abualigah, L., Diabat, A., Mirjalili, S., Abd Elaziz, M., Gandomi, A.H.: The arithmetic optimization algorithm. Comput. Methods Appl. Mech. Eng. **376**, 113609 (2021)
11. Storn, R., Price, K.: Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**, 341–359 (1997)
12. Zhu, M., Meng, S., Li, J., Yan, S.: Mobile service selection in edge and cloud computing environment with grey wolf algorithm. Int. J. Web Grid Serv. **18**(3), 229–249 (2022)
13. Wang, Y., Zhou, N., Lang, H., Li, Y.: An optimal composite service selection model based on edge-cloud collaboration. In: Chan, W.K., Claycomb, B., Takakura, H., Yang, J.-J., Teranishi, Y., Towey, D., Segura, S., Shahriar, H., Reisman, S., Ahamed, S.I. (eds.) 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 1170–1175. IEEE, Madrid (2021)
14. Jian, C., Li, M., Kuang, X.: Edge cloud computing service composition based on modified bird swarm optimization in the internet of things. Clust. Comput. **22**, 8079–8087 (2019)
15. Wang, P., Liu, X., Chen, J., Zhan, Y., Jin, Z.: Qos-aware service composition using blockchain-based smart contracts. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, pp. 296–297 (2018)
16. Holland, J.H.: Genetic algorithms. Sci. Am. **267**(1), 66–73 (1992)
17. Kaur, M., Jadhav, A., Akter, F.: Resource selection from edge-cloud for iiot and blockchain-based applications in industry 4.0/5.0. Secur. Commun. Netw. **2022**, 1–10 (2022)
18. Papadakis-Vlachopapadopoulos, K., Dimolitsas, I., Dechouniotis, D., Tsiropoulou, E.E., Roussaki, I., Papavassiliou, S.: On blockchain-based cross-service communication and resource orchestration on edge clouds. Informatics **8**(1), 13 (2021)
19. Duan, W., Jiang, Y., Xu, X., Zhang, Z., Liu, G., et al.: An edge cloud data integrity protection scheme based on blockchain. Secur. Commun. Netw. **2022**, 5016809 (2022)
20. Khan, W.Z., Ahmed, E., Hakak, S., Yaqoob, I., Ahmed, A.: Edge computing: a survey. Future Gener. Comput. Syst. **97**, 219–235 (2019)
21. Bashir, I.: Mastering Blockchain. Packt Publishing Ltd, Birmingham (2017)
22. Zhang, P., Zhou, M.: Security and trust in blockchains: architecture, key technologies, and open issues. IEEE Trans. Comput. Soc. Syst. **7**(3), 790–801 (2020)
23. Vivekanadam, B.: Analysis of recent trend and applications in block chain technology. J. ISMAC **2**(04), 200–206 (2020)
24. Li, J., Yan, Y., Lemire, D.: Full solution indexing for top-k web service composition. IEEE Trans. Serv. Comput. **11**(3), 521–533 (2016)
25. Yu, F., Li, J., Zhu, M., Yan, X.: Using seagull optimisation algorithm to select mobile service in cloud and edge computing environment. Int. J. Web Eng. Technol. **17**(1), 88–114 (2022)

26. Li, J., Zhu, M., Yu, M., Yan, Y., Cui, L.: Service composition based on pre-joined service network in graph database. Int. J. Web Grid Serv. **16**(4), 422–440 (2020)
27. Bi, W., Ma, J., Zhu, X., Wang, W., Zhang, A.: Cloud service selection based on weighted kd tree nearest neighbor search. Appl. Soft Comput. **131**, 109780 (2022)
28. Mirjalili, S.: Moth-flame optimization algorithm: a novel nature-inspired heuristic paradigm. Knowl.-Based Syst. **89**, 228–249 (2015)
29. Dehghani, M., Hubálovskỳ, Š, Trojovskỳ, P.: Northern goshawk optimization: a new swarm-based algorithm for solving optimization problems. IEEE Access **9**, 162059–162080 (2021)
30. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95-international Conference on Neural Networks vol. 4, pp. 1942–1948. IEEE, Perth (1995)
31. Xu, Y.-P., Tan, J.-W., Zhu, D.-J., Ouyang, P., Taheri, B.: Model identification of the proton exchange membrane fuel cells by extreme learning machine and a developed version of arithmetic optimization algorithm. Energy Rep. **7**, 2332–2342 (2021)
32. Abualigah, L., Diabat, A., Sumari, P., Gandomi, A.H.: A novel evolutionary arithmetic optimization algorithm for multilevel thresholding segmentation of covid-19 ct images. Processes **9**(7), 1155 (2021)
33. Debnath, S., Baishya, S., Sen, D., Arif, W.: A hybrid memory-based dragonfly algorithm with differential evolution for engineering application. Eng. Comput. **37**(4), 2775–2802 (2021)
34. Tang, B., Xiang, K., Pang, M.: An integrated particle swarm optimization approach hybridizing a new self-adaptive particle swarm optimization with a modified differential evolution. Neural Comput. Appl. **32**, 4849–4883 (2020)
35. Pike, R.: The go programming language. Talk given at Google's Tech Talks **14** (2009)
36. JetBrains: GoLand: Go IDE by JetBrains. https://www.jetbrains.com/go (2001) Accessed 20 August 2023
37. Ethereum: Remix-Project. https://github.com/ethereum/remix-project (2007) Accessed 20 August 2023
38. Ethereum: Solidity. https://github.com/ethereum/solidity/blob/develop/docs/index.rst (2007) Accessed 20 August 2023
39. Ethereum: Geth. https://geth.ethereum.org (2013) Accessed 20 August 2023
40. Al-Masri, E., Mahmoud, Q.H.: Qos-based discovery and ranking of web services. In: 2007 16th International Conference on Computer Communications and Networks, pp. 529–534 (2007). IEEE
41. Buterin, V., *et al.*: A next-generation smart contract and decentralized application platform. white paper **3**(37), 2–1 (2014)
42. Schott, J.R.: Fault tolerant design using single and multicriteria genetic algorithm optimization. PhD thesis, Massachusetts Institute of Technology (1995)
43. Wang, Y., Gao, S., Wang, S., Zimmermann, R.: An adaptive multiobjective multitask service composition approach considering practical constraints in fog manufacturing. IEEE Trans. Ind. Inform. **18**(10), 6756–6766 (2021)
44. Coello, C.A.C., Cortés, N.C.: Solving multiobjective optimization problems using an artificial immune system. Genet. Program Evolv. Mach. **6**, 163–190 (2005)

Yingying Ning is a graduate student at the School of Computer Science and Technology at Shandong University of Technology, Shandong, China. Her main research interests are blockchain and service computing.

Jing Li is an associate professor at the School of Computer Science and Technology at Shandong University of Technology, Shandong, China. She received her doctoral degree in Computer Science at Concordia University, Montreal. Her main research interests are Service computing and Database-based Service Composition.

Ming Zhu is an assistant professor at the School of Computer Science and Technology at Shandong University of Technology, Shandong, China. He did his Ph.D. in computer science at Concordia University. Ming's research interests are related to Service computing, Edge computing, Cloud computing and Graph Neural Network.

Chuanxi Liu is a graduate student at the School of Computer Science and Technology at Shandong University of Technology, Shandong, China. His main research interests are blockchain and service computing.