Computing and Informatics, Vol. 42, 2023, 896–922, doi: 10.31577/cai_2023_4_896

MOOA-CSF: A MULTI-OBJECTIVE OPTIMIZATION APPROACH FOR CLOUD SERVICES FINDING

Youcef Bezza

ICOSI Laboratory, Abbes Laghrour University
Khenchela, Algeria
e-mail: bezza.youcef@univ-khenchela.dz

Ouassila Hioual

Abbes Laghrour University, Khenchela, LIRE Laboratory of Constantine 2
Algeria
e-mail: ouassila.hioual@gmail.com

Ouided Hioual

Abbes Laghrour University, Khenchela, Algeria
e-mail: ouided.hioual@gmail.com

Derya Yiltas-Kaplan, Zeynep Gürkaş-Aydin

Department of Computer Engineering, Istanbul University-Cerrahpaşa
Avcilar, Istanbul, Türkiye
e-mail: {dyiltas, zeynepg}@iuc.edu.tr

Abstract. Cloud computing performance optimization is the process of increasing the performance of cloud services at minimum cost, based on various features. In this paper, we present a new approach called MOOA-CSF (Multi-Objective Optimization Approach for Cloud Services Finding), which uses supervised learning and multi-criteria decision techniques to optimize price and performance in cloud

computing. Our system uses an artificial neural network (ANN) to classify a set of cloud services. The inputs of the ANN are service features, and the classification results are three classes of cloud services: one that is favorable to the client, one that is favorable to the system, and one that is common between the client and system classes. The ELECTRE (ÉLimination Et Choix Traduisant la REalité) method is used to order the services of the three classes. We modified the genetic algorithm (GA) to make it adaptive to our system. Thus, the result of the GA is a hybrid cloud service that theoretically exists, but practically does not. To this end, we use similarity tests to calculate the level of similarity between the hybrid service and the other benefits in both classes. MOOA-CSF performance is evaluated using different scenarios. Simulation results prove the efficiency of our approach.

Keywords: MCDM, cloud computing, optimization, artificial neural networks, genetic algorithm, similarity measures, supervised learning

1 INTRODUCTION

In recent years, users have become increasingly accustomed to using the internet to obtain software resources. This is done in the form of web services, provided by information technology organizations, and can be accessed by end users over the internet [1]. Cloud computing is a service delivery paradigm that provides access to services and resources. It is defined by the National Institute of Standards and Technology as a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be quickly provisioned and released with minimal management effort or service provider interaction [2]. Cloud computing has three service models: Software as a Service, Platform as a Service, and Infrastructure as a Service. Furthermore, it has four deployment models: private, public, hybrid, and community [3]. The cloud service provider (CSP) supplies services to users as a rental. Due to the huge number of available virtual cloud resources, the CSP role is very complex. As such, researchers have given more attention to cloud service performance [3].

With the development of cloud computing technology, a single web service can no longer meet users' needs, since these are often complex. On the other hand, since different service providers offer web services with the same functionalities, but different in terms of their criteria, selecting the best web service that satisfies user needs is a difficult problem. This can decrease the performance of cloud services, which has a direct impact on the client's business. So, the more cloud performance is optimized, the more client confidence is increased [4].

Performance optimization of cloud computing is about making the components in the cloud meet component-level requirements and client expectations. We aim to increase the performance of cloud services with a minimum cost, depending on various constraints. Performance optimization allows us to improve the performance

of various factors [4]. To meet the needs of different clients, it is important to optimize the performance of cloud computing. In the literature, some solutions have been proposed. Among these solutions, the approaches can be cited based on: hybrid optimization [5, 6], genetic algorithm [7, 8, 9], and multi-objective optimization [3, 10, 11].

They are classified into two categories:

1. Approaches that optimize cloud performance on the client's side, in which response time and cost factors are always taken into account as users always need the best services at the lowest cost and response time; however, the cost factor is equally crucial. Since the service cost is a major factor to provide QoS, especially for commercial customers, therefore, compromising the response time for long deadline requests is desirable compared to the compromising in the cost factor [12].
2. Approaches that optimize cloud performance on the system side apply various techniques and strategies, such as caching, compression, load balancing, autoscaling, and serverless computing. This allows reduced latency, increased throughput, enhanced availability, and resources savings. To our knowledge, very few works that have taken into account user needs and system performance.

To address these challenges, this paper proposes a novel approach to optimize cloud computing performance. Compared to the above approaches, our proposition takes into consideration both the optimization of user and system preferences. Therefore, our contribution allows to solve the problem of the cloud service finding while satisfying user and system constraints. Our approach is based on neural network classification, multi-criteria decision-making systems (MCDM), optimization algorithms (genetic algorithm), and similarity measurements. First, we use an artificial neural network (ANN) to classify cloud services. The ANN inputs are service criteria. The classification results are three classes of cloud services. The first class is composed of services that are favorable for the client side (i.e. that satisfy client needs). The second class is composed of services that are favorable for the cloud provider side. In addition, the third one contains services that are common between the client and the provider. Concerning the MCDM, we have chosen to use the ELECTRE method. It is applied to sort services in each class. The purpose of this sorting is to eliminate weak services from the three classes. In a second step, we have modified the genetic algorithm (GA) to make it adaptive to our system. The genetic algorithm result is a hybrid cloud service that theoretically exists but practically does not. For this, we use similarity tests to calculate the level of similarity between the hybrid service and the other services to obtain the best service that meets the client's needs at a low price.

The rest of this paper is organized as follows: background and preliminaries are presented in Section 2, related work in Section 3, proposed model in Section 4, case study in Section 5, experimental results and evaluation in Section 6, and conclusion in Section 7.

2 BACKGROUND AND PRELIMINARIES

In this section, we will first give a brief overview of ANN, followed by an introduction to the ELECTRE II method that we consider in our proposal.

2.1 Artificial Neural Network

ANNs are an information-processing paradigm that simulates the behavior of the human brain for a specific task or function [13]. This type of network is composed of several sets of calculations called neurons, which are combined in layers and operate in parallel. The information is propagated from the input layer to the output layer.

ANNs can store empirical knowledge and make it available to users. The knowledge of the network is stored in synaptic weights, obtained through the process of adaptation or learning [14]. Activation values are transmitted from neuron to neuron based on the weights and activation functions. Each neuron adds up the activation values it receives and then changes the value according to its activation function. The activation procedure follows a look-ahead process and the difference between the predicted value and the actual value (error) is propagated backward by distributing it among the weights of each neuron according to the amount of error for which its neuron is responsible [14].

2.2 ELECTRE II

ELECTRE II [15] is a multi-criteria analysis method that solves decision problems with greater accuracy. This method was the first of the ELECTRE methods specifically designed to deal with ranking problems. The evaluation matrix is the starting point of the ELECTRE II method, in which alternatives are evaluated on different criteria. It aims to rank actions from best to worst. Based on a total pre-ordering principle, ELECTRE II assumes that all actions are comparable; incomparability is excluded, i.e., the decision-maker can always choose between action A and action B.

ELECTRE consists of two main steps. The first step is the preparation of the decision matrix (see Table 1), where g_{ij} denotes the value of variant i with respect to criterion j. The second step is the calculation of the concordance and discordance matrices [15].

		Criteria			
		C_1	C_2	...	C_n
Alternatives	a_1	g_{11}	g_{12}	...	g_{1n}
	a_2	g_{21}	g_{22}	...	g_{2n}
	...	...	...	...	...
	a_n	g_{n1}	g_{n2}	...	g_{nn}

Table 1. Decision matrix illustration

2.2.1 Calculation of the Concordance Matrix

The concordance matrix is generated by summing the weights of the elements in the concordance set. The strength of the hypothesis that alternative A_i is at least as good as alternative A_j is evaluated usisng the concordance index between the pair of alternatives A_i and A_j which is calculated using formula (1) [16]:

$$c(a,b) = \frac{\sum k_j}{k}, \quad \text{where } g_{j(a)} \geq g_{j(b)} \forall_j, \tag{1}$$

where $g_{j(a)}$ and $g_{j(b)}$ are the sets of criteria for which a is equal or preferred to b, k_j is the weight of the j^{th} criterion.

2.2.2 Calculation of the Discordance Matrix

The discordance index D (a, b) is calculated by formula (2) or (3):

$$D(a,b) = 0, \quad \text{if } \forall j, g_{j(a)} \geq g_{j(b)} \tag{2}$$

else

$$D(a,b) = \frac{1}{\sigma} \text{MAX} j[g_{j(b)} - g_{j(a)}], \tag{3}$$

$$\sigma = max \left| g_{j(b)} - g_{j(a)} \right|. \tag{4}$$

After calculating the concordance and discordance indices for each pair of alternatives, two types of outranking relationships are constructed by comparing these indices with two pairs of threshold values: $(C+, D+)$ and $(C-, D-)$. The pair $(C+, D+)$ is defined as the concordance and discordance thresholds for the strong outranking relationship, and the pair $(C-, D-)$ is defined as the thresholds for the weak outranking relationship, where $C+ > C-$ and $D+ > D-$. Then, outranking relationships are constructed according to the following two rules [15]:

- If $C(a,b) \geq C+$, $D(a,b) \leq D+$ and $C(a,b) \geq C(b,a)$, then alternative a is considered to strongly outperform alternative b. Likewise,
- If $C(a,b) \geq C-$, $D(a,b) \leq D-$ and $C(a,b) \geq C(b,a)$, then alternative a is considered to weakly outperform alternative b.

The values of $C-$, $C+$, $D-$, $D+$ are given by the decision makers [15].

3 RELATED WORK

The performance optimization of cloud computing, based on different features, has become increasingly important in today's world. For this reason, there are many studies developed in the literature. In this section, we will focus on some of them and suggest a comparative table (see Table 2) to introduce an analysis of these studies based on different parameters (features).

In [17], Guo et al. proposed a queuing model and developed a synthetic optimization method to optimize the performance of services. They analyzed and conducted the equation of each parameter of the services in the data center. Then, by analyzing the queuing system's performance parameters, they proposed the synthetic optimization mode, function, and strategy. Finally, they set up the simulation based on the synthetic optimization mode. By comparing and analyzing the simulation results to classical optimization methods, the authors showed that the proposed model can optimize the average wait time, average queue length, and number of clients.

The authors in [18] proposed a prediction-based dynamic multi-objective evolutionary algorithm, named NN-DNSGA-II. They incorporated an ANN with the NSGA-II. The optimization objectives taken into account included minimizing makespan, cost, energy, and imbalance, while maximizing reliability and utilization. The authors demonstrated that in Dynamic Multi-objective Optimization Problems (DMOPs) with unknown true Pareto-optimal fronts, the NN-DNSGA-II algorithm showed remarkable superiority over other alternatives. It outperformed them in various metrics, such as the number of non-dominated solutions, Schott's spacing, and the Hypervolume indicator in most cases.

The authors of [19] expanded the functionality of an existing parallel software framework called WoBinGO, which was initially designed for GA-based optimization, to be suitable for deployment in a cloud environment. Additionally, the researchers introduced an intelligent decision support engine that utilizes artificial neural networks (ANN) and metaheuristics. This engine enables users to evaluate the framework's performance on the underlying infrastructure concerning optimization duration and resource consumption cost. By conducting this assessment, users can make informed decisions based on their preferences, whether they prioritize faster result delivery or lower infrastructure expenses.

Authors of [20] recognized the role of the innovative Grasshopper Optimization Algorithm (GOA). They have strongly highlighted the significance of such an algorithm for optimizing resource allocation in a cloud computing environment. The proposed algorithm was simulated with MATLAB using eight datasets. Furthermore, the authors conducted a comparative analysis between the Grasshopper Optimization Algorithm (GOA) and the genetic algorithm (GA) and SElf-adaptive Inertia weight and Random Acceleration (SEIRA) algorithms. This comparison aimed to accurately assess the performance of GOA. The findings demonstrated the effectiveness of the proposed GOA in efficiently solving the resource allocation problem in the cloud.

In [21], Salem et al. created a new algorithm (MOABC) derived from a combination of ABC and Multi-Objective Optimization. Hence, the study introduced optimized replica placement strategies to determine the most suitable locations based on minimum distance and cost-effective paths. Additionally, for direct bees, the approach focused on identifying the shortest routes in terms of distance and lower cost. The proposed algorithm gave fast access to data and selected the best replica placement nearest to users. Additionally, the placement optimization pro-

vided more least-cost paths, better response times, and replication costs within the budget.

Authors of [22] used a hybrid metaheuristic algorithm, namely, the Whale Optimization Algorithm (WOA) with Simulated Annealing (SA), to optimize the energy consumption of sensors in IoT-based WSNs. To simulate the IoT network, the authors used the Xively IoT platform. Several performance metrics, such as load, residual energy, number of alive nodes, cost function, and temperature, were used to choose the optimal CHs in the IoT network. The proposed work in [22] was subjected to a comparison with various state-of-the-art approaches, and it demonstrated favorable results.

In [23], the authors introduced a new and innovative approach for performance optimization using a multi-agent system, which is based on both the Internet of Things (IoT) and the deep learning paradigm. They took advantage of the state-of-the-art probabilistic, recurrent neural network, and long short-term memory models to predict, intelligently, the upcoming behavior and optimization needs of the system. They deployed the proposed performance optimization approach and showed significant performance gain in comparison with existing approaches.

In [5], a hybrid optimization model has been developed which allows for an efficient task allocation to the virtual machines (VMs) in cloud computing. The task priorities are managed by using the hierarchy process. The authors have used BAT (Bandwidth-aware divisible task) and BAR models to consider the task properties and VM characteristics for task scheduling. They also used MOML (the minimum overload and minimum lease) preemption policy which was successfully employed to reduce the load on the VMs. The performance of the proposed model was then compared with existing algorithms such as BAT and ACO (ant colony optimization) algorithms. Consequently, the authors have been able to prove that their model is efficient in terms of resource, bandwidth, and memory utilizations.

In [24], the authors proposed a decision support engine that recommends optimal framework parameters to achieve minimal total execution time and total cumulative uptime for a specified optimization problem. The engine solves a bi-criteria optimization problem and uses surrogate models of the IaaS behavior under various large-scale optimization loads as a fitness evaluator in MOGA (multi-objective genetic algorithms). According to the authors, the obtained results were promising, especially in the case of computationally heavy fitness evaluation functions.

In [25], the authors introduced a novel approach named Multifaceted Optimization Scheduling Framework (MFOSF), which integrates scheduling and resource cost chronology models. According to the authors, this framework effectively illustrates the relationship between the user's budget and the producer's cost during the planning process.

In [26], Zhou et al. proposed a cloud service optimization method based on an artificial ant colony algorithm and bee colony algorithm (DAABA). To enhance the applicability of the farming season, the authors incorporated both the dynamic coefficient strategy and the reliability feedback update strategy into the optimization model. These additions were made to strengthen the overall performance and

adaptability of the model. Furthermore, the optimal fusion evaluation strategy was used to save optimization time by reducing useless iterations, while the iterative adjustment threshold strategy was adopted to improve the accuracy of cloud service finding by increasing the size of the bee colony.

Ragmani et al. [27] proposed a hybrid Fuzzy Ant Colony Optimization (FACO) algorithm for VM scheduling to guarantee high efficiency in a cloud environment. The proposed fuzzy module evaluates historical information to calculate the pheromone value and select a suitable server while keeping an optimal computing time [27]. Their study provides one of the first investigations into how to choose the optimal parameters of ant colony optimization algorithms using the Taguchi experimental design.

In [28], the authors introduced a hybrid meta-heuristic algorithm based on Firefly Optimization Algorithm and GA to optimize task scheduling in the cloud computing platform for multiple tasks. The developed system provides a distribution by reallocating the loads to the related VMs, taking into account the objective function of the VM. The system, also, increases resource utilization and communication cost during task scheduling and efficiently decreases the processing time of the process compared to different techniques such as GA, Firefly Algorithm, and Modified Firefly Optimization Algorithm.

In their paper [29], the authors proposed a novel hybrid load balancing model based on optimizing a modified particle swarm algorithm. This model incorporates enhanced metaheuristic firefly algorithms, which significantly improve the overall performance of cloud computing systems. The proposed approach primarily focuses on predictive workload allocation, emphasizing resource scalability and implementing a load balancing model that maximizes the utilization of uniformly load-distributed virtual machines (VMs) [29].

The authors of [7] proposed an approach to improve the capability of data centers. It allocates requests among VMs in an inefficient manner by using their current status in cloud computing with a GA [7]. The authors claim that their algorithm uses a modified GA-based approach, in which the best VM is selected by analyzing candidates which have more fitness compared to others. The proposed approach significantly reduced the response time of servers and provided an effective load balancing among VMs [7].

In [30], an efficient optimization method for task scheduling was presented. It is based on a hybrid Multi-Verse Optimizer with a GA (MVO-GA). MVO-GA was proposed to enhance the performance of tasks transfer via the cloud network, based on cloud resources' workload. The proposed method works on multiple properties of cloud resources, namely: speed, capacity, task size, number of tasks, number of VMs, and throughput. The proposed method successfully optimized the task scheduling of a large number of tasks [30]. Also it optimized the large cloud tasks' transfer time, reflecting its effectiveness.

After analyzing the articles cited in Table 2, we can classify them into two main classes. The first class optimizes cloud performance on the client's side [17, 19, 20, 21, 22, 23, 5, 10, 8]. The second class contains articles that optimize cloud perfor-

Research Paper	Environment	Used Parametres					Client/System Side	
		R	Av	C	Th	Rt	Client	System
[17]	Cloud Computing	−	−	−	−	+	+	−
[18]	Cloud Computing	+	−	−	+	+	−	+
[19]	Cloud Computing	−	−	−	−	−	+	−
[20]	Cloud Computing	−	−	−	+	−	+	−
[21]	Cloud Computing	−	−	+	+	−	+	−
[22]	IoT Network	−	−	−	+	−	+	−
[23]	IoT	−	−	−	−	−	+	−
[5]	Cloud Computing	−	−	−	−	+	+	−
[24]	Cloud Computing	−	−	−	−	+	+	−
[25]	Cloud Computing	−	−	−	+	−	−	+
[26]	Cloud Computing	+	−	−	−	−	+	−
[27]	Cloud Computing	−	−	−	−	−	−	+
[28]	Cloud Computing	−	−	−	−	−	−	+
[29]	Cloud Computing	−	−	−	−	−	−	+
[7]	Cloud Computing	−	−	−	−	+	−	+
[30]	Cloud Computing	−	−	−	−	−	−	+
Our Approach	Cloud Computing	+	+	+	+	+	+	+

Table 2. A comparative summarization of some previous studies on performances optimization

mance on the system side [7, 18, 25, 27, 28, 29, 30]. In our research, we introduce a model that optimizes cloud performance for both clients and systems, positioning our proposal at the intersection of these two classes (as shown in Figure 1).

In the model, R, Av, C, Th, and Rt represent Reliability, Availability, Cost, Throughput, and Response Time, respectively.

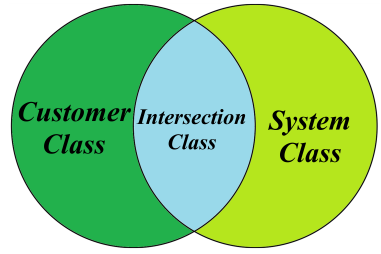

Figure 1. The contextual situation of our research work

4 PROPOSED APPROACH

The successful development of cloud computing has attracted more and more people and companies to use it. On the one hand, the use of cloud computing reduces

costs; on the other hand, it improves efficiency. As users are largely concerned by the quality of services, optimizing the performance of Cloud computing has become essential for its successful application [17]. Furthermore, the number of cloud providers is increasing rapidly. Therefore, the challenge of choosing the cloud provider that best meets a client's needs and optimizes both cost and performance has become a big challenge. In this context, we propose an approach that helps clients to choose the best provider that meets their needs and optimizes both cost and performance.

In this study, we consider a service with five criteria. Two criteria are specifically related to the client side, two criteria are focused on the cloud provider side, and there is one criterion that is common to both the client and the provider. The common criterion in our study is subject to opposing objectives: the client seeks to minimize it, while the provider aims to maximize it. This is precisely why we opted for multi-criteria decision-making systems. In many decision-making problems, diverse perspectives are encountered, often leading to contradictions and differing viewpoints.

4.1 Architecture and Functioning of Our Approach

Figure 2 depicts the overall architecture of our system, which comprises five interconnected components.

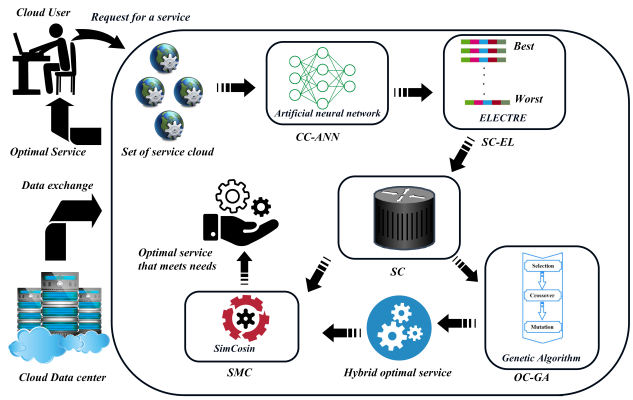

Figure 2. An overview of the proposed general architecture

The first component in our system is the Classification Component (CC-ANN), which utilizes a multi-layer neural network. This component comprises three layers. The first layer is the input layer, where the inputs represent the service criteria. Consequently, we extract the parameters of each cloud service, considering the client's requirements, and represent each service with a vector. The second layer is the hidden layer, which contains the function responsible for making the classification.

Lastly, the third layer is the output layer, generating outputs into three classes: the Client Preferences Class (CPC) containing services preferred by the client, the System Preferences Class (SPC) comprising services preferred for the system, and the Common Services Class (CSC).

The second component in our architecture is referred to as the Sorting Component (SC-El). It relies on the ELECTRE method to arrange the services within each class, ranking them from the best to the worst. This sorting process involves assigning weights to each criterion based on its significance. As for the Storage Component (SC), it serves as a centralized repository within the system, responsible for storing all relevant data.

The fourth component is the Optimization Component (OC-GA), which operates based on the principles of genetic algorithm (GA). Its primary function is to generate a new generation of cloud services derived from the initial three classes mentioned earlier.

The services produced by GA are considered hybrid services, which theoretically exist but are not practically realized. To address this, we need to assess the similarity between these hybrid services and the existing ones. To achieve this, we have introduced the fifth component, the Similarity Component (SMC). The SMC assists in identifying the best service (the closest match) among the services obtained in the preceding steps. The entire process is illustrated in the sequence diagram shown in Figure 3.

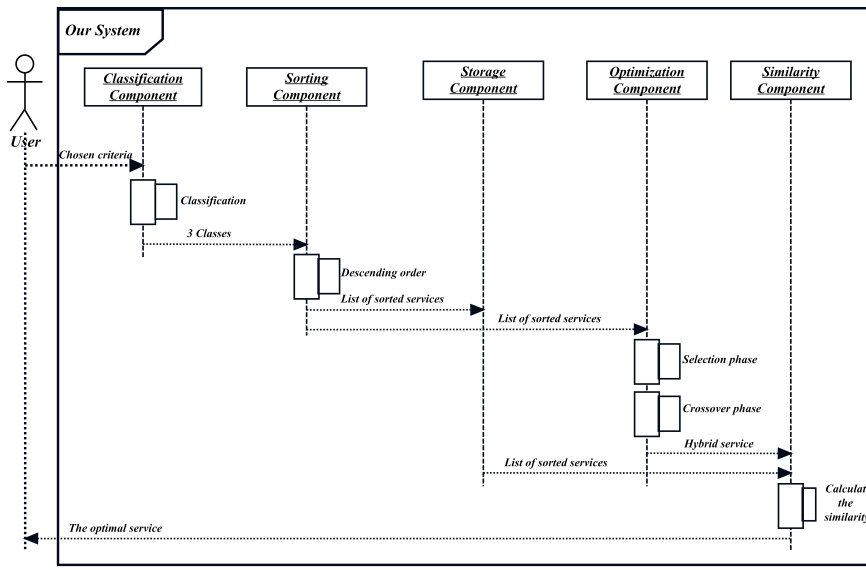

Figure 3. Sequence diagram of the MOOA-CSF functionality

4.2 Classification Step

This step is performed by the CC-ANN component, as shown in Figure 4. The ANN is composed of three layers: input, hidden and output layers. The input layer is comprised of five nodes, denoting the criteria of a service, namely Reliability, Throughput, Availability, Cost, and Response Time. These criteria values are then propagated to the hidden layer. Within the hidden layer, the activation values are passed from neuron to neuron, where each neuron aggregates the received activations and updates its value using a transfer function. This process involves an anticipation mechanism. Subsequently, the difference between the predicted value and the actual value (error) is propagated backward through the network.

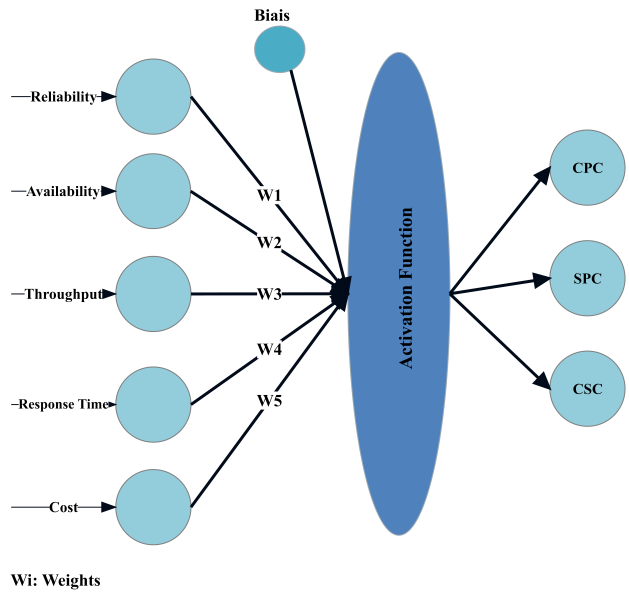

Figure 4. The layers of the CC-ANN component

4.3 Activation Function

The activation function transfers the input values to an output signal. In this paper, we have chosen the hyperbolic tangent function (Tanh). The Tanh function is similar to the sigmoid function, but it is symmetrical around the origin. This results in different signs of outputs from the previous layers being fed into the input of the next layer, as defined by formula (5).

The Tanh function is continuous and differentiable, with values ranging between -1 and 1. Compared to the sigmoid function, the gradient of the Tanh function

is steeper. Tanh is more commonly used than the sigmoid function because it has gradients that are not bounded to vary in a certain direction, and it is also centered on zero [31].

$$Tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}, \tag{5}$$

where a is the value of the neuron. Figure 5 shows the process of the classification component, where d denotes the desired goal and E is the error margin.

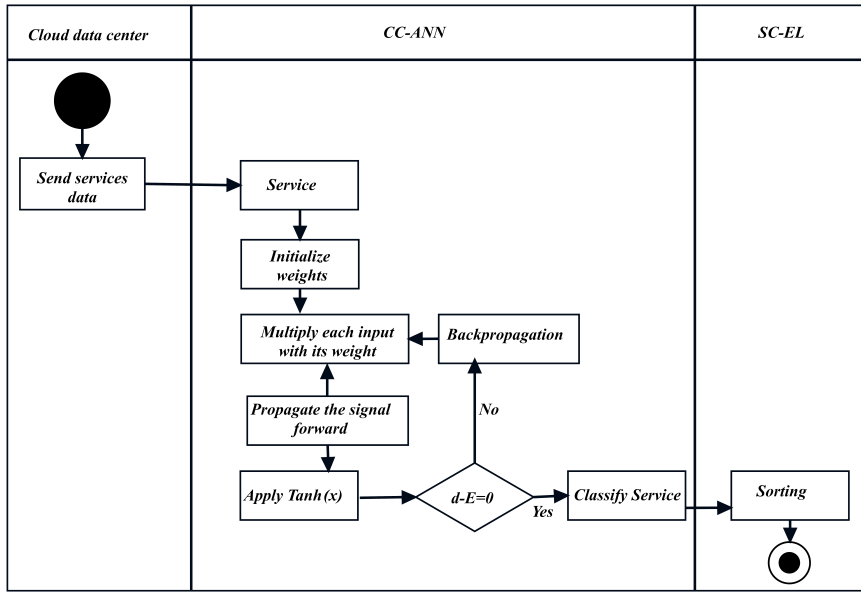

Figure 5. Activity diagram of the CC-ANN component

4.4 Sorting and Elimination Step

This component performs according to the ELECTRE II principle, and is composed of two steps; sorting and elimination. We have to prepare the decision matrix; the alternatives in our case illustrate the services. Each service has five criteria, and each class has a decision matrix that is different from the others.

In the first step, we sort the services of each class from the best to the worst by calculating the concordance, discordance, and dominant matrices. Service a is better than service b if the following strong outranking relation holds:

- If $C(a,b) \geq C+$, $D(a,b) \leq D+$ and $C(a,b) \geq C(b,a)$, then service a is considered to strongly outperform service b.

In the second step, we eliminate the services that are weakly outranking. We consider that a service a is weakly outranking with b when the following condition is true:

- If $C(a,b) \geq C-$, $D(a,b) \leq D-$ and $C(a,b) \geq C(b,a)$, then service a is considered to strongly outperform service b.

4.5 Optimization Step

This component is based on the principle of GA, a metaheuristic approach to solve multi-objective optimization problems. GA is inspired by the principles of Darwin's theory of evolution and is often used as an evolutionary computational model in various fields of study [32]. Currently, GAs are recognized as a very powerful tool in optimization, having been applied in computer science, engineering, education, and stock market data mining optimization [32].

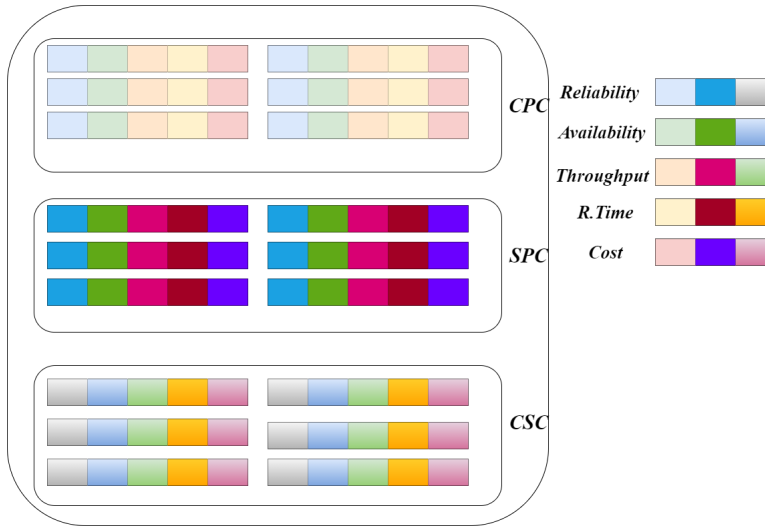

Figure 6. Initial populations of CP, SP and CS classes

We have applied modifications to the GA to make it adaptive to our context. Its operation after modification is as follows:

1. The initial population in our approach is not generated randomly. Instead, we utilize the three classes obtained during the classification phase as our initial populations. Consequently, there are three separate initial populations, not just one (as shown in Figure 6).
2. The population is evaluated by assigning a fitness value to each service, so we can generate a new population.

3. The algorithm determines the termination of the search process based on specific predefined conditions. Typically, these conditions are met when the algorithm reaches a fixed number of generations or when it discovers a satisfactory solution.
4. In case the termination condition is not satisfied, the population proceeds with the selection step. During this step, one service is chosen from each class based on its fitness score, with higher fitness scores leading to a higher likelihood of selection.
5. Following the selection step, the algorithm proceeds to implement crossover on the chosen services, as illustrated in Figure 7. This stage involves creating new services for the subsequent generation through the process of crossing or recombination.

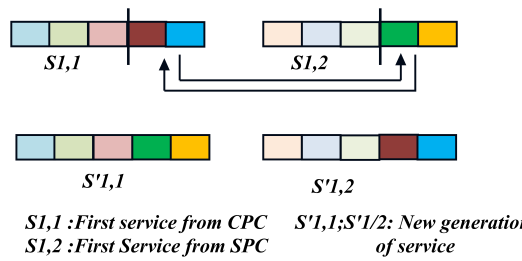

Figure 7. Illustration of the crossover operation results

6. At this stage, the new population returns to the assessment step and the process begins again. We call each cycle of this loop a *generation*.
7. When the termination condition is met, the algorithm breaks out of the loop and usually returns its final search results to the client/provider.

4.6 Similarity Step

The calculation of the degree of similarity between two services is ensured by the SMC, which is based on the similarity SimCosin(X, Y). We suppose that we have two services X and Y represented by two vectors, each vector containing five criteria [33]:

$$X = [R_x, Av_x, C_x, Th_x, Rt_x];$$

$$Y = [R_y, Av_y, C_y, Th_y, Rt_y]$$

Additionally, the function $SimCosin(X, Y)$ must satisfy the following properties [33]:

Property 1: $0 \leq \cos(X, Y) \leq 1$.

Property 2: $\cos(X, Y) = \cos(Y, X)$.

Property 3: $\cos(X, Y) = 1$; if $X = Y$.

5 CASE STUDY

To validate our system, a simulation environment is established. The simulation context proceeds as follows: we suppose that we have several cloud services providers and each one provides a service. A client searches for optimal services that meet their needs among these services, and each service has (m) criteria.

The simulation environment is a PC with the following configuration: Nvidia GeForce GTX 1060 GDDR5, Intel Core i7-7700HQ CPU 2.80 GHz and RAM 16 GB. The programming environment is Eclipse IDE 2020-09. The test data is based on the QWS2 dataset [34] where the number of services is 4000. The neural network was trained using 2600 services. Each service is composed of five criteria.

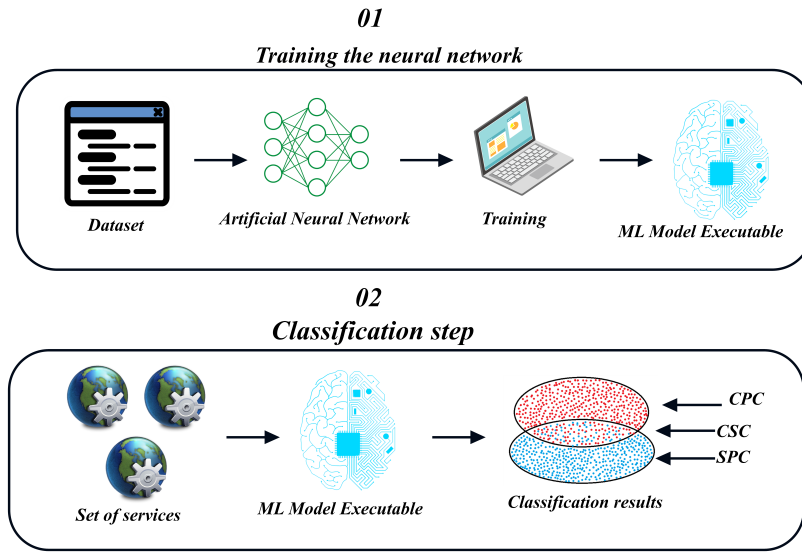

Figure 8. Classification process of the CC-ANN component

As a first phase, we develop a neural network to classify these services. Thus, we assign each service to the appropriate class.

As shown in Figure 8, in the first step we train the neural network using a dataset. We initialize our weights randomly then feed-forward the values from one layer to the next. If the output is not equal to the desired value, we back-propagate from the output neuron to the input neuron. We update the weights and feed-forward the values. We repeat this process until we find the desired values and obtain a model of a neural network.

In the Classification step (Figure 8), we use the model of the neural network to classify services into three categories. Each class contains a set of services that have the same range of values. The first class (CPC) contains 916 services, the second (SPC) contains 640 services, and the last one (CSC) contains 444. After that, we

make a copy list of these services and send it to the storage component. Then, we transfer the result to the sorting component.

As a second phase, the sorting component uses the ELECTRE method to sort and then eliminate services to find the best one in each class. The decision problem is characterized by five parameters or criteria. All criteria are advantage criteria, i.e., performance is better when the score is high.

The weights of the criteria are presented in Table 3.

Criterion	Availability	Cost	Response Time	Throughput	Reliability
Weight	2	3	3	2	1

Table 3. Initialisation of the five criteria weights

Due to the large number of services in the dataset, we will take as a sample the services S6, S8, S9, S907, S908 and S910. Therefore, the service performance matrix of the first class, as illustrated in Table 4, is used to calculate the concordance matrix $C(a, b)$, which is illustrated in Table 5. This is calculated using the formula (1) that was quoted in the previous section.

Service ID	Reliability	Cost	Response Time	Throughput	Availability
Service 6	61	68	1 046	2 178	68
Service 8	73	63	1 250	2 144	75
Service 9	81	124	1 244	2 060	85
Service 907	64	62	1 175	1 655	43
Service 908	73	63	1 188	1 963	50
Service 910	52	110	1 107	1 999	60

Table 4. Performance matrix of the illustrative example

Service ID	Service 6	Service 8	Service 9	Service 907	Service 908	Service 910
Service 6	**1**	0.454	0.181	0.545	0.545	0.454
Service 8	0.545	**1**	0.454	**1**	**1**	0.727
Service 9	0.818	0.545	**1**	**1**	**1**	**1**
Service 907	0.454	**0**	**0**	**1**	**0**	0.454
Service 908	0.454	0.454	**0**	**1**	**1**	0.454
Service 910	0.545	0.272	**0**	0.545	0.545	**1**

Table 5. The concordance matrix relating to the illustrative example

We calculate the discordance matrix $D(a, b)$ of our illustrative example using formula (3), results are shown in Table 6. To obtain σ, we calculate, for each attribute, the differences between all its values in the dataset. Moreover, the attribute corresponding to the maximum value of these deviations is maintained. We then calculate σ, which is equal to the maximum value of the maintained attribute minus its

minimum value (see formula (4)). According to the QWS2 dataset, the maintained attribute is the Cost attribute consequently $\sigma = 140$.

Service ID	Service 6	Service 8	Service 9	Service 907	Service 908	Service 910
Service 6	**0**	0.085	0.4	0.021	0.081	0.3
Service 8	0.035	**0**	0.435	**0**	**0**	0.335
Service 9	0.0008	0.004	**0**	**0**	**0**	**0**
Service 907	0.178	0.228	0.442	**0**	0.064	0.342
Service 908	0.128	0.178	0.435	**0**	**0**	0.335
Service 910	0.064	0.15	0.207	0.085	0.15	**0**

Table 6. The discordance matrix relating to the illustrative example

After calculating the concordance and discordance matrices, the dominant matrix is constructed from the two concordance and discordance indices. Thus, the following strong and weak outranking indices are obtained: $C+ = 0.850$, $C- = 0.750$, $D+ = 0.200$, and $D- = 0.300$. We obtain the dominant matrix as illustrated in Table 7.

Service ID	Service 6	Service 8	Service 9	Service 907	Service 908	Service 910
Service 6	Strong	–	–	–	–	–
Service 8	–	Strong	–	Strong	Strong	–
Service 9	Weak	–	Strong	Strong	Strong	Strong
Service 907	–	–	–	Strong	–	–
Service 908	–	–	–	Strong	Strong	–
Service 910	–	–	–	–	–	Strong

Table 7. The dominant matrix relating to the illustrative example

Figure 9 shows the final rank between services, to get this graph we follow this two properties:

- If service a outranks service b, an arrow starting at vertex a and ending at vertex b.
- If no outranking relation exists between the two services a and b, then no arrow can be drawn between the two vertices.

After that, we sort services to eliminate those with weak outclass relations. A service with a weak outclass relation to another service means that the former is included in the latter. The remaining services, which have not been eliminated, are the best in each class. We use these services as an initial population. This latter is used to generate new populations based on the good services. We repeat the same process for the second and third classes, then transferring the resulting services to the optimization component. The OC will consider each class as an initial population. At first, the OC crosses a service from the first class with the services from the second class. Then, we evaluate the new service. If the value is less than

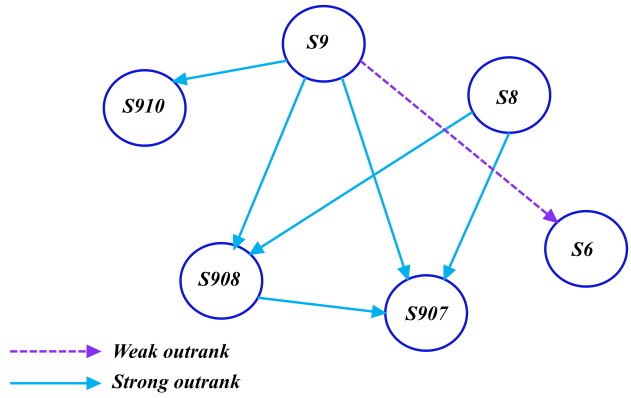

Figure 9. The outrank relation between services relating to the illustrative example

the fitness function, we eliminate the service. After crossing all services of the first class with all services of the second class, we will obtain a new population. Then we cross a service from the second class with the services of the third class, obtaining a second population. After that, we cross the services of the new populations to get the final one.

The final population contains hybrid services; the latter are abstract. To design the optimal service, we must calculate the similarity between services already stored in our storage component and those of the final population.

The similarity component (Figure 10) calculates the similarity index between the best services of each class and services of the final population. In our case, we use the Sim-Cos function. If the similarity index is in the interval [0.8:1], we keep the service. Furthermore, we keep all services that have a high index. Then, we send the list of these services to the client.

6 EXPERIMENTAL RESULTS AND EVALUATION

In this section, we evaluate our system based on the number of services in each class. We assume that there are more than 100 services in each class. In the experiments (1, 2, and 3), the values of response time and throughput do not change because they include technical aspects (servers, computer network, etc.) as well as those related to the interface ergonomics between the user and the system.

6.1 Experiment 1

The goal of this experimentation is to show the average of services in the client-preferred-class (CPC) before and after optimization. As shown in Figure 11, the values of response time, throughput, availability, reliability, and cost before optimization are, respectively, 1.12, 1.6, 83 %, 89 % and $ 110. After optimization,

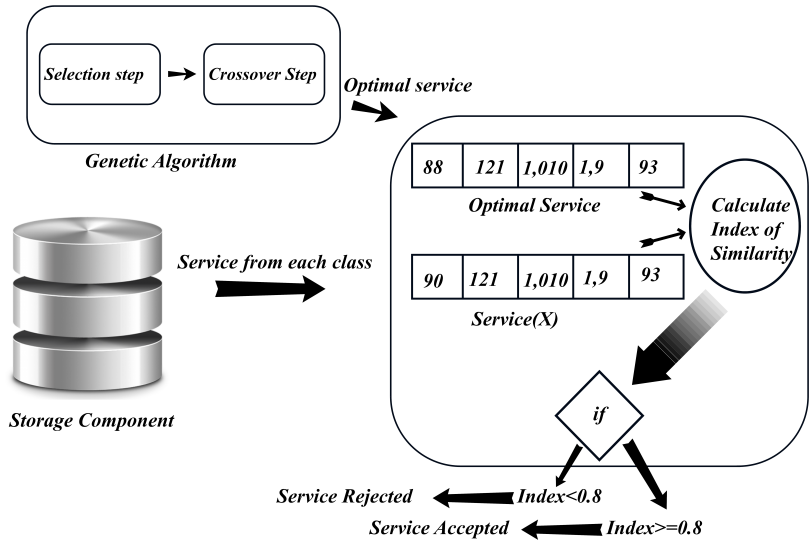

Figure 10. The similarity component functioning

availability increases from 83 % to 92 %, reliability from 89 % to 96 % and cost decreases from $ 110 to $ 93. Through this experimentation, we note that the values of three criteria have been optimized, due to the number of crossed services.

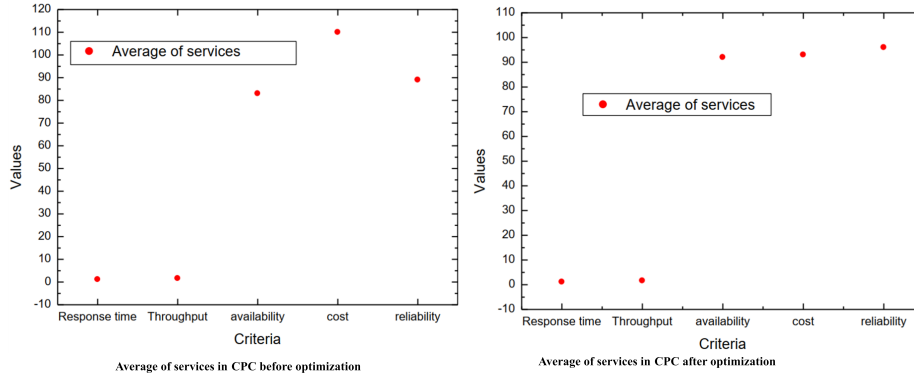

Figure 11. Average of services in CPC before and after optimization

6.2 Experiment 2

The goal of this experimentation is to show the average of services in the SPC before and after optimization. As shown in Figure 12, the values of the response time,

throughput, availability, reliability and cost before optimization were respectively 2.1, 2.74, 74 %, 82 % and $ 135. After optimization, the availability increases from 74 % to 89 %, reliability from 82 % to 93 % and the cost decreases from $ 135 to $ 100. In this experiment, we found that the number of crossing services led to an optimization in the values of three criteria.

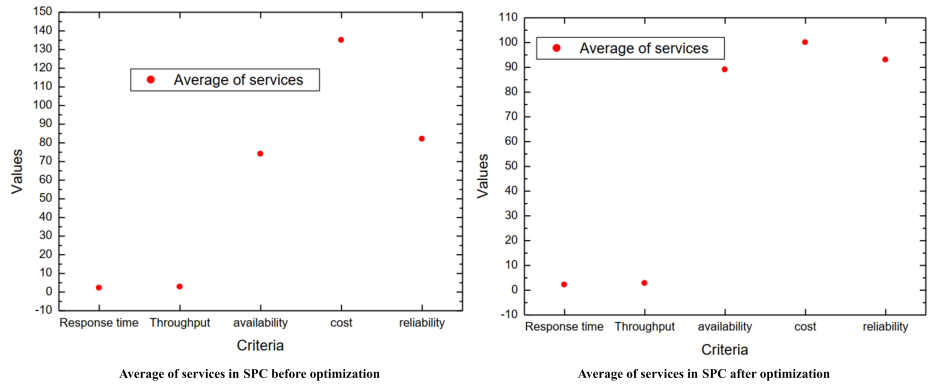

Figure 12. Average of services in SPC before and after optimization

6.3 Experiment 3

The goal of this experimentation is to show the average of services in the Common-Services-Class (CSC) before and after optimization. As shown in Figure 13, the values of the response time, throughput, availability, reliability and cost, before the optimization, are respectively 1.52, 2.35, 85 %, 81 % and $ 120. After the optimization, the availability increases from 85 % to 96 %, the reliability increases from 81 % to 94 % and the cost decreases from $ 120 to $ 93. Due to the number of intersecting services, we observe through this experiment that the values of the three criteria have been optimized.

6.4 Experiment 4

The goal of this experiment is to evaluate our system according to the number of services. To reach this goal, we vary the number of services from less than 100 to up to 1000, then observe the values of the criteria. The experimental results are shown in Figure 14. When the number of services is more than 100, the availability has increased from 75.35 % to 82 %, the reliability has increased from 80.52 % to 85 %, the cost has decreased from $ 141.58 to $ 130, and the response time and throughput are unchanged. When the number of services is more than 500 and less than 1000, the availability has increased from 75.35 % to 90 %, the reliability has increased from 80.52 % to 92 %, the cost has decreased from $ 141.58 to $ 115.4, and the response

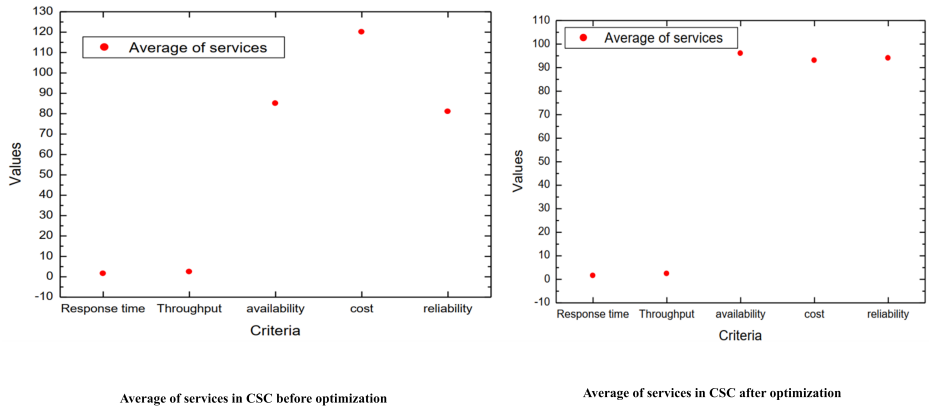

Figure 13. Average of services in CSC before and after optimization

time and throughput are unchanged. Finally, when the number of services is more than 1000, the availability has increased from 75.35 % to 99 %, the reliability has increased from 80.52 % to 98 %, the cost has decreased from $ 141.58 to $ 95, and response time and throughput are unchanged. From these results, we can conclude that the optimization rate increases accordingly with the increase of the number of services.

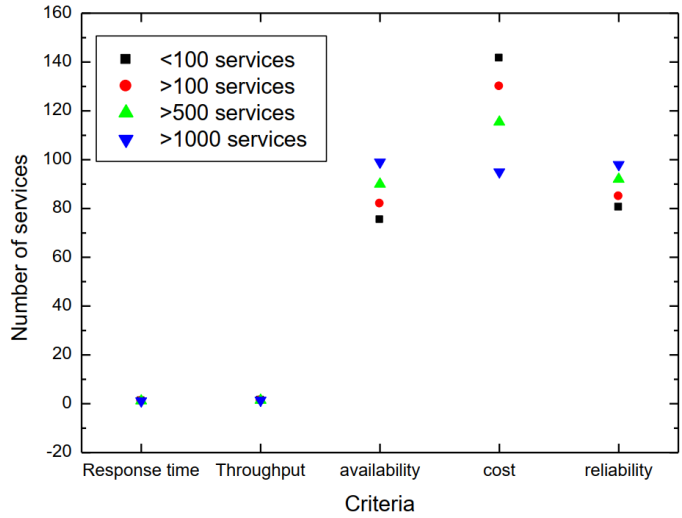

Figure 14. Impact of the services number on the optimization rate

7 CONCLUSION

Recently, with the development of cloud computing, the performance optimization of cloud services has become a very attractive research topic. This paper develops a new approach based on supervised learning and multi-criteria decision techniques to optimize cost and performance of services in a cloud-computing environment. Our approach uses an ANN to classify a set of cloud services. The inputs of this ANN are the service features, and its output is the classification results, consisting of two classes of cloud services. The first class comprises services preferred by the client, while the second class consists of services preferred by the providers. The ELECTRE method is employed to rank the services in both classes. Additionally, we made adaptations to the GA to suit our system. The GA produces a hybrid cloud service, which exists only in theory and not in practice. To resolve this issue, we used similarity tests to quantify the similarity level between the hybrid service and other advantages found in both classes. The experimental results demonstrate the effectiveness of MOOA-CSF. Nevertheless, certain limitations exist in this study, including the absence of optimization for response time and throughput. In our future research, we intend to expand our approach by incorporating other crucial criteria. Moreover, we aim to optimize response time and throughput using alternative optimization methods.

REFERENCES

[1] Ghobaei-Arani, M.—Rahmanian, A. A.—Souri, A.—Rahmani, A. M.: A Moth-Flame Optimization Algorithm for Web Service Composition in Cloud Computing: Simulation and Verification. Software: Practice and Experience, Vol. 48, 2018, No. 10, pp. 1865–1892, doi: 10.1002/spe.2598.

[2] Gayathri, V.—Selvi, S.—Kalaavathi, B.: Analysis on Cost and Performance Optimization in Cloud Scheduling. International Journal of Engineering Research and Technology (IJERT), Vol. 3, 2014, No. 11, pp. 929–934.

[3] Mishra, S. K.—Sahoo, B.—Parida, P. P.: Load Balancing in Cloud Computing: A Big Picture. Journal of King Saud University - Computer and Information Sciences, Vol. 32, 2020, No. 2, pp. 149–158, doi: 10.1016/j.jksuci.2018.01.003.

[4] Periasamy, R.: Performance Optimization in Cloud Computing Environment. 2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), 2012, pp. 1–6, doi: 10.1109/CCEM.2012.6354621.

[5] Sreenivasulu, G.—Paramasivam, I.: Hybrid Optimization Algorithm for Task Scheduling and Virtual Machine Allocation in Cloud Computing. Evolutionary Intelligence, Vol. 14, 2021, pp. 1015–1022, doi: 10.1007/s12065-020-00517-2.

[6] Abualigah, L.—Yousri, D.—Abd Elaziz, M.—Ewees, A. A.—Al-Qaness, M. A. A.—Gandomi, A. H.: Aquila Optimizer: A Novel Meta-Heuristic Optimization Algorithm. Computers and Industrial Engineering, Vol. 157, 2021, Art. No. 107250, doi: 10.1016/j.cie.2021.107250.

[7] Kaurav, N. S.—Yadav, P.: A Genetic Algorithm-Based Load Balancing Approach for Resource Optimization for Cloud Computing Environment. International Journal of Information and Computing Science, Vol. 6, 2019, No. 3, pp. 175–184.

[8] Zhou, Z.—Li, F.—Zhu, H.—Xie, H.—Abawajy, J. H.—Chowdhury, M. U.: An Improved Genetic Algorithm Using Greedy Strategy Toward Task Scheduling Optimization in Cloud Environments. Neural Computing and Applications, Vol. 32, 2020, pp. 1531–1541, doi: 10.1007/s00521-019-04119-7.

[9] Yiqiu, F.—Xia, X.—Junwei, G.: Cloud Computing Task Scheduling Algorithm Based on Improved Genetic Algorithm. 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2019, pp. 852–856, doi: 10.1109/ITNEC.2019.8728996.

[10] Shrimali, B.—Patel, H.: Multi-Objective Optimization Oriented Policy for Performance and Energy Efficient Resource Allocation in Cloud Environment. Journal of King Saud University - Computer and Information Sciences, Vol. 32, 2020, No. 7, pp. 860–869, doi: 10.1016/j.jksuci.2017.12.001.

[11] Bezza, Y.—Hioual, O.—Hioual, O.: A Multicriteria Decision Model for Optimizing Costs and Performances for a Cloud User. Distributed Sensing and Intelligent Systems: Proceedings of ICDSIS 2020, 2022, pp. 427–437, doi: 10.1007/978-3-030-64258-7_37.

[12] Ahmad, S. G.—Iqbal, T.—Munir, E. U.—Ramzan, N.: Cost Optimization in Cloud Environment Based on Task Deadline. Journal of Cloud Computing, Vol. 12, 2023, No. 1, Art. No. 9, doi: 10.1186/s13677-022-00370-x.

[13] Abdella, M.—Marwala, T.: The Use of Genetic Algorithms and Neural Networks to Approximate Missing Data in Database. IEEE 3rd International Conference on Computational Cybernetics (ICCC 2005), 2005, pp. 207–212, doi: 10.1109/ICCCYB.2005.1511574.

[14] Hicham, G. T.—Chaker, E. A.—Lotfi, E.: Comparative Study of Neural Networks Algorithms for Cloud Computing CPU Scheduling. International Journal of Electrical and Computer Engineering (IJECE), Vol. 7, 2017, No. 6, pp. 3570–3577, doi: 10.11591/ijece.v7i6.pp3570-3577.

[15] Aiello, G.—Enea, M.—Galante, G. M.: A Multi-Objective Approach to Facility Layout Problem by Genetic Search and Electre Method. Book Faim, 2005.

[16] Mary, S. A. S. A.—Suganya, G.: Multi-Criteria Decision Making Using ELECTRE. Circuits and Systems, Vol. 7, 2016, No. 6, pp. 1008–1020, doi: 10.4236/cs.2016.76085.

[17] Guo, L.—Yan, T.—Zhao, S.—Jiang, C.: Dynamic Performance Optimization for Cloud Computing Using M/M/M Queueing System. Journal of Applied Mathematics, Vol. 2014, 2014, Art. No. 756592, doi: 10.1155/2014/756592.

[18] Ismayilov, G.—Topcuoglu, H. R.: Neural Network Based Multi-Objective Evolutionary Algorithm for Dynamic Workflow Scheduling in Cloud Computing. Future Generation Computer Systems, Vol. 102, 2020, pp. 307–322, doi: 10.1016/j.future.2019.08.012.

[19] Simic, V.—Stojanovic, B.—Ivanovic, M.: Optimizing the Performance of Optimization in the Cloud Environment - An Intelligent Auto-Scaling Ap-

proach. Future Generation Computer Systems, Vol. 101, 2019, pp. 909–920, doi: 10.1016/j.future.2019.07.042.

[20] Vahidi, J.—Rahmati, M.: Optimization of Resource Allocation in Cloud Computing by Grasshopper Optimization Algorithm. 2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI), 2019, pp. 839–844, doi: 10.1109/KBEI.2019.8735098.

[21] Salem, R.—Salam, M. A.—Abdelkader, H.—Mohamed, A. A.: An Artificial Bee Colony Algorithm for Data Replication Optimization in Cloud Environments. IEEE Access, Vol. 8, 2019, pp. 51841–51852, doi: 10.1109/ACCESS.2019.2957436.

[22] Iwendi, C.—Maddikunta, P. K. R.—Gadekallu, T. R.—Lakshmanna, K.—Bashir, A. K.—Piran, M. J.: A Metaheuristic Optimization Approach for Energy Efficiency in the IoT Networks. Software: Practice and Experience, Vol. 51, 2021, No. 12, pp. 2558–2571, doi: 10.1002/spe.2797.

[23] Irshad, O.—Khan, M. U. G.—Iqbal, R.—Basheer, S.—Bashir, A. K.: Performance Optimization of IoT Based Biological Systems Using Deep Learning. Computer Communications, Vol. 155, 2020, pp. 24–31, doi: 10.1016/j.comcom.2020.02.059.

[24] Ivanovic, M.—Simic, V.—Stojanovic, B.—Kaplarevic-Malisic, A.—Marovic, B.: Elastic Grid Resource Provisioning with WoBinGO: A Parallel Framework for Genetic Algorithm Based Optimization. Future Generation Computer Systems, Vol. 42, 2015, pp. 44–54, doi: 10.1016/j.future.2014.09.004.

[25] Bansal, M.—Malik, S. K.: A Multi-Faceted Optimization Scheduling Framework Based on the Particle Swarm Optimization Algorithm in Cloud Computing. Sustainable Computing: Informatics and Systems, Vol. 28, 2020, Art. No. 100429, doi: 10.1016/j.suscom.2020.100429.

[26] Zhou, K.—Wen, Y.—Wu, W.—Ni, Z.—Jin, T.—Long, X.: Cloud Service Optimization Method Based on Dynamic Artificial Ant-Bee Colony Algorithm in Agricultural Equipment Manufacturing. Mathematical Problems in Engineering, Vol. 2020, 2020, pp. 1–11, doi: 10.1155/2020/9134695.

[27] Ragmani, A.—Elomri, A.—Abghour, N.—Moussaid, K.—Rida, M.: FACO: A Hybrid Fuzzy Ant Colony Optimization Algorithm for Virtual Machine Scheduling in High-Performance Cloud Computing. Journal of Ambient Intelligence and Humanized Computing, Vol. 11, 2020, pp. 3975–3987, doi: 10.1007/s12652-019-01631-5.

[28] Vinothini, C.—Balasubramanie, P.—Priya, J.: Hybrid of Meta Heuristic Firefly and Genetic Algorithm for Optimization Approach in the Cloud Environment. Webology, Vol. 17, 2020, No. 1, pp. 297–305, doi: 10.14704/WEB/V17I1/WEB17005.

[29] Lilhore, U. K.—Simaiya, S.—Maheshwari, S.—Manhar, A.—Kumar, S.: Cloud Performance Evaluation: Hybrid Load Balancing Model Based on Modified Particle Swarm Optimization and Improved Metaheuristic Firefly Algorithms. International Journal of Advanced Science and Technology, Vol. 29, 2020, No. 5, pp. 12315–12331.

[30] Abualigah, L.—Alkhrabsheh, M.: Amended Hybrid Multi-Verse Optimizer with Genetic Algorithm for Solving Task Scheduling Problem in Cloud Computing. The Journal of Supercomputing, Vol. 78, 2022, No. 1, pp. 740–765, doi:

10.1007/s11227-021-03915-0.

[31] Sharma, S.—Sharma, S.—Athaiya, A.: Activation Functions in Neural Networks. International Journal of Engineering Applied Sciences and Technology (IJEAST), Vol. 4, 2020, No. 12, pp. 310–316.

[32] Lin, L.—Cao, L.—Wang, J.—Zhang, C.: The Applications of Genetic Algorithms in Stock Market Data Mining Optimisation. Management Information Systems 2004, 2004, pp. 273–280.

[33] Biswas, P.—Pramanik, S.—Giri, B. C.: Cosine Similarity Measure Based Multi-Attribute Decisionmaking with Trapezoidal Fuzzy Neutrosophic Numbers. Neutrosophic Sets and Systems, Vol. 8, 2015, pp. 46–56.

[34] Al-Masri, E.—Mahmoud, Q. H.: Investigating Web Services on the World Wide Web. Proceedings of the 17th International Conference on World Wide Web, 2008, pp. 795–804, doi: 10.1145/1367497.1367605.

Youcef Bezza is a Ph.D. student in computer science, in security and web technology specialty at the Abbas Laghrour Khenchela University, Algeria. He obtained his Master's degree and Licence degree from the same university. His research interest includes cloud computing, optimization, IoT and fault tolerance.

Ouassila Hioual received her B.Sc. and M.Sc. in computer science from the Mentouri University of Constantine, Algeria in 2002 and 2005. She has worked as a Lecturer at the Department of Computer Science and Mathematics Science at the Mentouri University of Constantine, Algeria from 2005 to 2008. Currently, she works as Full Professor at the Department of Mathematics and Computer Science at Abbes Laghrour University of Khenchela, Algeria. She supervised many doctoral, master and licence students. Since September 2006 until October 2011, she prepared her Ph.D. in computer science. She has published a number of articles in international conferences and journals. Her research interests include CPS, Industry 4.0, data science, IoT and cloud computing environments, energy consumption.

Ouided Hioual received her B.Sc. degree in computer science from the Mentouri University of Constantine, Algeria in 2004, and M.Sc. degree in computer science from the Abbes Laghrour University of Khenchela, Algeria in 2009. Currently, she is working as Associate Professor at the Department of Mathematics and Computer Science at Abbes Laghrour University of Khenchela, Algeria. She has supervised many Ph.D., Master and Licence students since September 2012 until now. Since October 2010 until December 2019 she prepared her Ph.D. in computer science. She has published a number of articles in international conferences and journals. Her research interests include semantic web, ontology, web services, multiagent system, cloud computing, knowledge management, reasoning in artificial intelligence, engineering of human-machine interfaces and automatic treatment of natural language.

Derya Yiltas-Kaplan received her B.Sc., M.Sc., and Ph.D. degrees in computer engineering from the Istanbul University, Istanbul, Türkiye, in 2001, 2003, and 2007, respectively. She completed her postdoctoral research with the North Carolina State University. She is currently Associate Professor with the Department of Computer Engineering, Istanbul University-Cerrahpaşa. Specializing in computer science, her research is notably focused on computer networks and data routing, making significant contributions to engineering and technology. She received a Postdoctoral Research Scholarship from The Scientific and Technological Research Council of Türkiye (TUBITAK).

Zeynep Gürkaş-Aydin completed her undergraduate studies in computer engineering at the Istanbul University Engineering Faculty in 2003. She completed her Master's degree in computer engineering at Istanbul University Institute of Science in 2005, her first Ph.D. in computer engineering at the Istanbul University Institute of Science in 2011, and her second Ph.D. in the field of informatics as part of the Ecole Doctorale program at Université Pierre-et-Marie-Curie: Paris VI in France in 2014. Currently, she serves as a faculty member in the Department of Cybersecurity at Istanbul University-Cerrahpaşa Engineering Faculty. Her research covers a wide range of topics in computer science and engineering.