Intelligent Automation & Soft Computing
DOI: 10.32604/iasc.2023.030651
Article

Tech Science Press

Artificial Bee Colony with Cuckoo Search for Solving Service Composition

Fadl Dahan[1,2,*] and Abdulelah Alwabel[3]

[1]Department of Information System, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj, 11942, Saudi Arabia
[2]Department of Computer Sciences, Faculty of Computing and Information Technology Alturbah, Taiz University, Taiz, 9674, Yemen
[3]Department of Computer Sciences, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj, 11942, Saudi Arabia
*Corresponding Author: Fadl Dahan. Email: f.naji@psau.edu.sa
Received: 30 March 2022; Accepted: 30 May 2022

Abstract: In recent years, cloud computing has provided a Software As A Service (SaaS) platform where the software can be reused and applied to fulfill complicated user demands according to specific Quality of Services (QoS) constraints. The user requirements are formulated as a workflow consisting of a set of tasks. However, many services may satisfy the functionality of each task; thus, searching for the composition of the optimal service while maximizing the QoS is formulated as an NP-hard problem. This work will introduce a hybrid Artificial Bee Colony (ABC) with a Cuckoo Search (CS) algorithm to untangle service composition problem. The ABC is a well-known metaheuristic algorithm that can be applied when dealing with different NP-hard problems with an outstanding record of performance. However, the ABC suffers from a slow convergence problem. Therefore, the CS is used to overcome the ABC's limitations by allowing the abandoned bees to enhance their search and override the local optimum. The proposed hybrid algorithm has been tested on 19 datasets and then compared with two standard algorithms (ABC and CS) and three state-of-the-art swarm-based composition algorithms. In addition, extensive parameter study experiments were conducted to set up the proposed algorithm's parameters. The results indicate that the proposed algorithm outperforms the standard algorithms in the three comparison criteria (best fitness value, average fitness value, and average execution time) overall datasets in 30 different runs. Furthermore, the proposed algorithm also exhibits better performance than the state–of–the–art algorithms in the three comparison criteria over 30 different runs.

Keywords: Cloud computing; web service composition; artificial bee colony; cuckoo search

1 Introduction

Over recent times, most computations have been done on the cloud rather than by local computers. The servers are located in cloud data centers in the cloud environment, where the infrastructures, software, and platforms are introduced as an internet-based service [1]. The main purpose of cloud computing is to facilitate

integration regarding both software and hardware services to produce a final product introduced to clients [2]. Cloud computing has developed rapidly, so this advance has enticed service providers into publishing worldwide. Nowadays, the published web services in the cloud appear to be a service-based system solution, and various protocols are designed to describe, search, and invoke the service-based software across the cloud. This solution was introduced to build the application for enterprises under the term of Web Service Composition (WSC) [3].

In WSC, the process is initiated through the decomposition of the clients' requests, where this request is formed as a workflow, including several different tasks subject to quality of services (QoS) constraints. As a result, optimization algorithms play a vital role in selecting an efficient QoS and achieving customer satisfaction. Several web services can be chosen for clients' requests where these services are called candidates. However, these several web services perform a similar function to the various non-functions called QoS. The QoS can be defined as non-functional requirements of the web services, such as cost, response time, and availability. The flow of the WSC process is illustrated in Fig. 1.

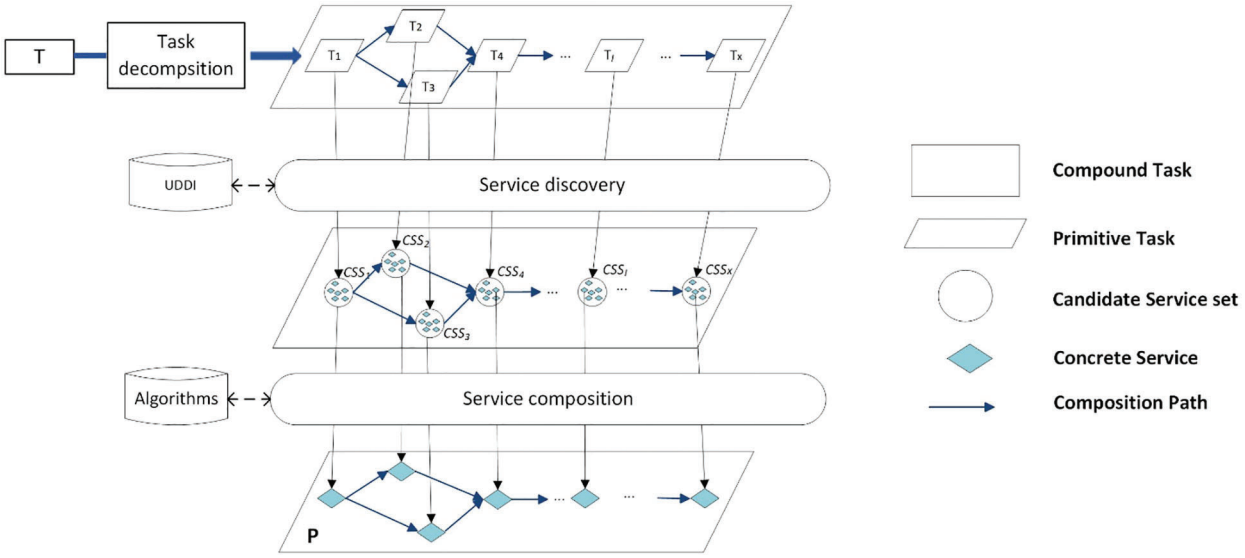

Figure 1: WSC process flow

Searching for optimal/near-optimal solutions is crucial in WSC. All possible web services/task could be selected in an exhaustive manner to form the best path. However, selecting the best paths in this way is inapplicable for large workflow. If the workflow contains X tasks and Y web services/tasks, then Y^x paths should be evaluated at an extremely huge computational cost. Therefore, searching for the optimal solution for WSC is formulated as an NP-hard problem. Recently, metaheuristic algorithms have presented an efficient and effective option when solving NP-hard problems [4]. The metaheuristic algorithms are mostly inspired by nature [5], the inspiration of their source being evolutionary (e.g., genetic algorithm), swarm (e.g., artificial bee colony optimization), and physical (e.g., simulated annealing) [6]. Metaheuristic algorithms have two contradictory searching mechanisms: Exploration (exploring the search space) and exploitation (exploiting the past knowledge of the best solutions found so far).

The artificial bee colony (ABC) is a metaheuristic algorithm that has been developed and inspired by the behavior of honey bees while foraging [7]. The ABC has the advantages of being easy to implement, few parameters needing to be controlled, competitive performance [8], and it is widely used in many

problems such as traveling salesman, multiple knapsack, and job shop scheduling. Meanwhile, it has the disadvantage of being poor in exploration because its searching mechanism is based on neighborhood search. Therefore, it has a fast convergence speed, so it is easily trapped in a premature convergence [9]. For such an algorithm, its performance can be enhanced by hybridizing with other algorithms with slow convergence speed, such as cuckoo search (CS).

Furthermore, the CS is a metaheuristic algorithm that was recently introduced by Yang et al. [10]. It simulates the behavior of some cuckoo species, which parasitize the host nests by laying their eggs in them. The possible parasitism cases are: The host distinguishes the cuckoo eggs and will then abandon the nest or dispose of these eggs. Elsewhere, the host doesn't differentiate the cuckoo eggs, and these eggs then survive and become mature. In this work, a hybrid variant of the ABC and CS was developed to solve the WSC problem. The proposed ABC variant combines the CS algorithm exploration with the ABC's exploitation. In this case, the scout bees adopt the Lévy flight before abandoning the bad solutions. The cuckoos collaborate with scout bees to avoid ABC getting stuck in a local minimum, where a set of cuckoos receives the poor bees and tries to enhance them. In the case of no improvement, the scout bees abandon these solutions.

The rest of this work is structured as follows: Section 2 focuses on related works. Sections 3 and 4 provide an overview of the standard ABC and CS algorithms. In Section 5, the proposed algorithms will be described. Section 6 shows the experiments 'results. The conclusion is presented in Section 7.

2 Related Works

Many swarm-based algorithms are proposed to deal with WSC problems, achieving good performances. In the literature, many review studies introduced extensive reviews of swarm-based and other optimization algorithms applied to WSC, such as [11,12]. In this section, a review of recent web service composition ABC and CS research will be presented, and the research's contribution in improving the metaheuristic searching mechanisms (exploration and exploitation) of the ABC variants will be highlighted.

Seghir [13] proposed a fuzzy artificial bee colony where the ranking method and fuzzy distance are integrated to keep the solution diversity of an artificial bee colony. The proposed algorithm proposed a trapezoidal fuzzy to deal with QoS constraints' ambiguity and didn't improve on any of the metaheuristic searching mechanisms of ABC (exploration and exploitation). Zhang et al. [14] introduced a neighborhood search to enhance the search abilities of artificial bee colonies and used opposition learning to maintain initialization diversity. Regarding metaheuristic searching mechanisms, the proposed work enhanced the ABC exploration by applying opposition learning to enhance the ABC initial population and the ABC exploitation using the dynamic adjust search range. Arunachalam et al. [15] added the integrated probability and acceptance rule-based approaches to optimize artificial bee colony search. The proposed enhancement supports exploration and exploitation balancing using acceptance rules and three probabilistic searches. The acceptance rule is used to support exploitation in the employee and onlooker phases, while three probabilistic searches are used to support exploration in the employee phase. Meanwhile, Dahan et al. [4] proposed ABC enhancement based neighborhood searches where the farthest nodes are preferred in the early stages (support exploration) while the near nodes are preferred in the later stages (support exploitation). The aforementioned research has been improved on [16] whereby a second step was added based on the swapping method to exploit the search space knowledge of the best bees. Chandra et al. [17] then introduced a modified instance of the ABC with the new search procedure applied to employed bees, and differential evolution instance applied to onlooker bees. The search procedure supports the exploration of the ABC, while exploitation is enhanced by differential evolution. Seghir et al. [18] proposed a novel interval-based method to enhance the ABC's performance. A neighborhood selection method was subsequently proposed to enhance ABC exploitation. Arunachalam

et al. [19] introduced ABC enhancement based on cosine similarity and combinatorial strategical method. This cosine similarity is used to support ABC exploitation combinatorial strategical search for exploration.

Zhou et al. [9] utilized the Pareto dominance to optimize the ABC using the CS Lévy flight for optimizing the employed bee search. The proposed work aimed to balance exploration and exploitation by designing a comprehensive learning strategy for onlooker search. Furthermore, Zhou et al. [20] introduced another hybrid algorithm of the ABC and CS. In the proposed algorithm, the CS Lévy flight is used by onlookers to improve exploitation, and the Lévy flight perturbation rate and step size are adjusted using a parameter adaptive strategy. Karthikeyan et al. [21] proposed a hybrid artificial bee colony to optimize web services search. The genetic algorithm utilized in this work to generate new solutions was set by composing the web services randomly.

Subbulakshmi et al. [22] optimized the service quality using a genetic algorithm, and cuckoo search is used to search for an optimal combination of web services. Ghobaei-Arani et al. [1] presented a cuckoo search algorithm while considering a distributed network. The quality with the credibility of service was considered in [23]. In addition, local optimization and the cuckoo search were integrated to search for optimal solutions. An adaptive cuckoo search was proposed in [24]. A new feature was added to optimize the exploration in the proposed algorithm, such as parameter adaption and linear population reduction. Thangaraj et al. [25] used the cuckoo search algorithm to search for an optimal services combination. A multi cuckoo [26] was proposed to solve the composition problem. Kouchi et al. [27] used the CS to solve the WSC problem over the cloud computing environment.

Recently, many swarm-based algorithms have been introduced to untangle the service composition problem. Allali et al. [28] presented a framework to address the WSC problem based on the ant colony optimization and the mobile agents. A hybrid algorithm ant colony optimization and the genetic algorithm is introduced in [29]. Li et al. [30] improved the convergence speed of the artificial bee colony algorithm using the genetic algorithm. Teng et al. [31] proposed enhancing the whale optimization algorithm based on the logarithmic convergence factor and aggregation potential energy. The improved eagle algorithm is introduced in [32]. Dogani et al. [33] introduced a hybrid particle swarm optimization and genetic algorithm where the genetic algorithm is used to enhance the exploration and exploitation of particle swarm optimization.

Due to swarm-based algorithms' stochastic nature, these algorithms cannot guarantee the best path for WSC problems. Moreover, the optimization No-Lunch-Free theorem (NLF) [34] emphasizes the disability of fining an optimizer good enough to settle all optimization problems. As a result, the current swarm-based algorithms for the WSC problem can suffer from degraded performance. This idea motivates the proposed work to introduce and investigate the efficiency of swarm-based hybridization.

3 Artificial Bee Colony

The ABC belongs to the population-based metaheuristic algorithms proposed by Karaboga [7,8]. It mimics the behavior of honey bees during their foraging. The ABC algorithm initializes the bees with random solutions using Eq. (1). Afterward, it measures solution quality according to the fitness function, which varies based on problems to memorize only the one best solution. Then, it dispatches three different types of bees (employee, onlooker, and scouts) that aim to search for global optimum using neighborhood search. The ABC algorithm is easy to use and has a few control parameters to tune (population size (N), maximum iterations (T), and searching limitations (limits)).

$$x_{ij} = x_{min,j} + rand(0,\ 1)(x_{max,j-}x_{min,j}) \tag{1}$$

where x_{ij} (i = 1, 2,…,N is the source number, j = 1, 2,…,D is the optimized parameters) is the new j^{th} food source for i^{th} bees. $x_{max,j,}$ $x_{min,j}$ are the maximum and minimum of the j^{th} food source.

At first, the ABC dispatches a group of bees (employee bees) to explore the search space and share the information with another group of bees (onlooker bees). Employee bees search for new solutions in accordance with the current solution neighborhood search, as shown in Eq. (2). Afterward, the fitness of each employee bee is computed, and ABC compares the current best solution with the best solution found so far and memorizes only the one best solution.

$$v_{ij} = x_{ij} + \varphi(x_{ij} - x_{kj}) \tag{2}$$

where x_{ij} is the current solution, x_{kj} (k = 1, 2,…,N is the food source) is a solution based on the neighborhood search, and $\varphi \in [-1,\ 1]$.

Secondly, the onlooker bees wait for the employee bees to finish searching for new solutions to select one to follow. They likewise search for new solutions relating to the current solution neighborhood search, as shown in Eq. (2) The selection mechanism is based on the probability of the solution quality of the employee bees, as shown in Eq. (3) Afterward, the fitness of each onlooker bee is computed, and the ABC compares the current best solution with the best solution found so far and memorizes only the one best solution.

$$p_i = \frac{fit(x_i)}{\sum_{l=1}^{N} fir(x_l)} \tag{3}$$

where *fit* represents the solution quality of i^{th} food source, N is the source number.

Finally, the scout bees monitor the solution improvement of each employee and onlooker bees, and they increase the improvement counter (limit) when these bees have no improvement. In the case of the employee and onlooker bees exceeding searching limitations (limits), the scout bees abandon the bad solutions and initialize poor bees with random solutions using Eq. (1).

The aforementioned steps will be repeated until ABC reaches the maximum iterations (T). Then, ABC will stop and return the best solution.

4 Cuckoo Search Algorithm

CS also belongs to the population-based metaheuristic algorithms proposed by Yang et al. [34]. In nature, some cuckoo species parasitize the host nests by laying their eggs in them. The possible parasitism cases are: The host distinguishes the cuckoo eggs, and it will then abandon the nest or dispose of these eggs. Elsewhere, the host doesn't differentiate the cuckoo eggs, and these eggs then survive and become mature.

The CS mimics the brood parasitism behavior of some cuckoo species. Firstly, the CS algorithm initializes the cuckoos with random solutions. Afterward, it measures the solution quality in accordance with the fitness function, which varies based on the problem of memorizing only the one best solution. Secondly, it randomly gets a cuckoo (solution) and uses Lévy flight to generate a new solution using Eq. (4). Finally, the CS algorithm abandons the worst solutions with a fraction $P_a \in [0,\ 1]$.

$$x_i^{t+1} = x_i^t + \alpha \oplus L(s,\ \lambda) \tag{4}$$

where x_i^t (i = 1, 2,…, n is the nest number, t is the egg number) is the randomly selected solution, $\alpha > 0$ represents the step size, and $L(s,\ \lambda)$ is the Lévy flight that is calculated as follows:

$$L(s,\ \lambda) = \frac{\lambda\Gamma(\lambda)\sin\left(\frac{\pi\lambda}{2}\right)}{\pi} \frac{1}{s^{1+\lambda}},\quad (s \gg s_0 > 0) \tag{5}$$

The step size α can be calculated as follows:

$$\alpha = \alpha_0(x_i^t - x_{best}) \tag{6}$$

where α_0 is a constant. x_{best} which represents the best local solution.

The aforementioned steps will be repeated until the CS reaches the stopping criteria. Then, it will return the best solution.

5 The Proposed Algorithm

In ABC, the three types of bees (employee, onlooker, and scout) are responsible for searching for the best solution and likewise balancing exploration and exploitation mechanisms. The employee and onlooker bees search for the best solution based on the neighborhood search, representing the ABC algorithm's exploitation mechanism. Scout bees abandon the bad solutions and reinitialize the poor bees; this process represents the exploration of the ABC algorithm. In fact, the ABC has a strong exploitation mechanism but a poor exploration mechanism, therefore easily getting stuck in a local minimum (premature convergence) [9]. In contrast, the CS uses Lévy flight to generate new solutions and perform the global search [35]. Then, a fraction of the new solutions will be abandoned and re-built randomly far-field from the current solution. Therefore, the CS is stronger in exploration, where it can't get easily stuck in a local minimum (Slow convergence). This work introduces a hybrid algorithm that combines the CS algorithm's exploration with ABC's exploitation. The scout bees adopt the Lévy flight before abandoning the bad solutions in the proposed algorithm. The cuckoos collaborate with scout bees to avoid ABC getting stuck in a local minimum, where a set of cuckoos receives the poor bees and tries to enhance them. In the case of no improvement, the scout bees abandon these solutions.

Normally, the proposed algorithm (ABC_CS for short that shown in Fig. 2) follows the standard ABC algorithm with employee and onlooker procedures and differs in terms of scout procedure. It starts with initializing each bee with a random solution. Then, each solution is evaluated using the fitness function (see next subsections). Next, it dispatches employee bees to search for better solutions and shares the new solutions' information with other onlooker bees using Eq. (2) Afterward, the fitness of each employee bee is computed, and the ABC compares the current best solution with the best solution found so far and memorizes only the one best solution. Each onlooker bee selects an employed bee to follow; then, it searches for a new solution using Eq. (2) Afterward, the fitness of each onlooker bee is computed, and the ABC compares the current best solution with the best solution found so far and memorizes only the one best solution.

In ABC_CS, the scout bees monitor the exhausted solutions, and rather than abandon them, they collaborate with cuckoos with Mantegna's algorithm to give these bees a chance to visit different solutions. The abounded solutions mean that the corresponding bees get stuck in the local minimum and the enhancement will help these bees avoid this problem while collaborating with the cuckoos. The proposed enhancement of the scout procedure contains the following steps:

Step1: The cuckoos receive the poor bees from the scout bees and use the Lévy flight in Mantegna's algorithm to give these bees a chance to visit different solutions. In Mantegna's algorithm, the step lengths can be drawn using Lévy flight as:

$$s = \frac{u}{|v|^{\frac{1}{\beta}}} \tag{7}$$

where v, u are calculated as shown in Eq. (8) and Eqs. (7) and (9), respectively.

$$v \sim N(0,\ \sigma_v^2),\ \text{where}\ \sigma_v = 1 \tag{8}$$

$$u \sim N(0,\ \sigma_u^2) \tag{9}$$

where σ_u is calculated as follows:

$$\sigma_u = \left\{ \frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\Gamma[(1+\beta)/2]\beta 2^{(\pi\beta/2)}} \right\}^{\frac{1}{\beta}} \tag{10}$$

where Γ represents the Gamma function.

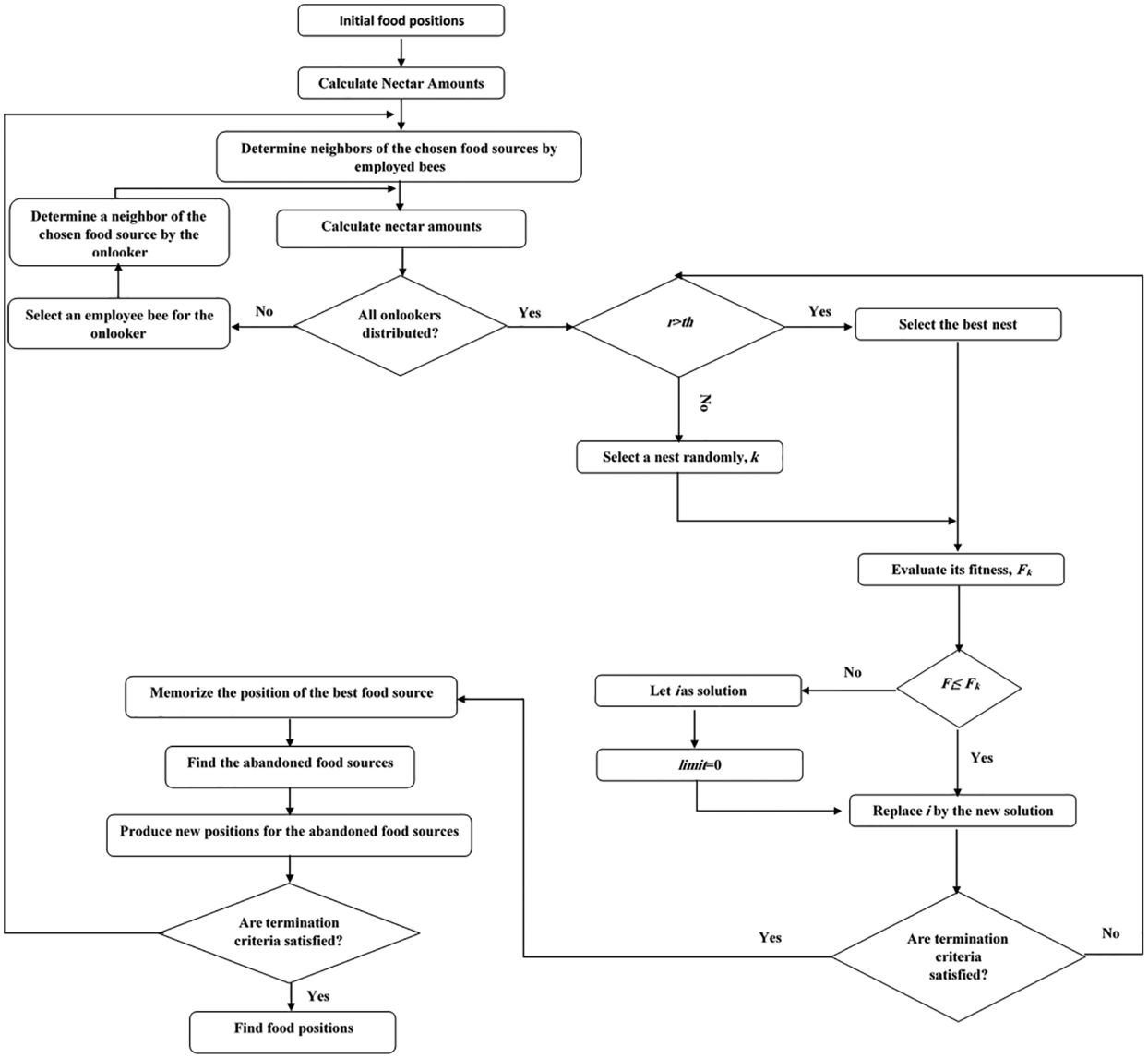

Figure 2: The proposed method (ABC_CS) flowchart

Step2: The best solution is kept in the CS algorithm while drawing the new best solution, as shown in Eq. (6). However, this process doesn't support the ABC_CS claim, where it aims to diversify the search

solutions. Therefore, the step size calculation is modified, and the ABC_CS gives a chance to build a solution relating to the participation of current solutions and the best solution or randomly selected ones. The step size formula is shown in Eq. (6) updates to:

$$\alpha = \begin{cases} \alpha_0(x_i^t - x_{best}) & if\ (r \geq th) \\ \alpha_0(x_i^t - x_k) & if\ (r < th) \end{cases} \tag{11}$$

where r is a random variable between 0 and 1. k = 1, 2,…, N is a randomly selected source of food. th is a threshold constant.

Step3: The above-mentioned steps will be repeated until the CS reaches the stopping criteria.

Step4: In the case of no enhancement being obtained by cuckoos, the scout bees abandon the bad solutions and reinitialize the poor bees with a random solution using Eq. (1).

5.1 Solution Representation

The solution representation is a challenge when designing population-based metaheuristic algorithms. In this work, each bee/cuckoo represents the solution using a one-dimensional vector that includes X (represents the number of tasks) elements. Each column in the vector contains a value of "1" to "Y" (represents the number of candidate services for each task), indicating that the corresponding services are selected to fulfill the task functionality. An example of solution representation is shown below:

$$x_i = (W_{(1,1)},\ W_{(2,5)},\ W_{(3,Y)},\ \ldots,\ W_{(X,7)}) \tag{12}$$

where i represents the bee/cuckoo. $W_{(3,Y)}$ represents the Y web services selected for the 3rd task.

5.2 Fitness Function

The fitness function in the service composition should be designed to consider the problem constraints. These constraints include the QoS properties. In addition, the nature of these properties is based on their objectives, where they either need to be maximized or minimized. Therefore, the service composition problem is represented as a multi-objective optimization problem. To cover this aspect, we adopted four different QoS properties in this work, which are the Cost (C), the Response Time (RT), Reliability (R), and Throughput (T). The objective value is for cost and response time to be minimized and the objective value of reliability and throughput to be minimized. In this work, the fitness function that covers the objective problem constraints is:

$$F_i = \left(\prod_{j=1}^{X} T_{jb} + \prod_{j=1}^{X} R_{jb} - \sum_{j=1}^{X} C_{jb} - \sum_{j=1}^{X} RT_{jb}\right) \tag{13}$$

where F_i represents the solution fitness of i^{th} bee/cuckoo. X represents the task number.

6 Experimental Result and Discussion

6.1 Experimental Setup

The proposed algorithm (ABC_CS for short) was implemented using Java. Nineteen datasets with tasks and web services/task settings, as used in [16], were utilized to evaluate the efficiency of the proposed hybrid algorithm. Tab. 1 presents the datasets where the first column is an index, the second presents the dataset name, the third presents the number of tasks, and the fourth presents the number of web services/task.

Table 1: List of datasets used in this work

	Dataset	No. tasks	No. WSs/task
1	dataset1	10	100
2	dataset2	10	200
3	dataset3	10	300
4	dataset4	10	400
5	dataset5	10	500
6	dataset6	10	600
7	dataset7	10	700
8	dataset8	10	800
9	dataset9	10	900
10	dataset10	10	1000
11	dataset11	20	100
12	dataset12	30	100
13	dataset13	40	100
14	dataset14	50	100
15	dataset15	60	100
16	dataset16	70	100
17	dataset17	80	100
18	dataset18	90	100
19	dataset19	100	100

These datasets can be categorized, based on the size of tasks and web services, into three different types (small, medium, and large). Small size datasets contain: Dataset1, dataset2, dataset3, dataset4, dataset5, dataset6, dataset7, dataset8, dataset9, dataset10, and dataset11. Medium size: Datasets contain dataset12, dataset13, dataset14, dataset15, and dataset16. Large size datasets contain: Dataset17, dataset18, and dataset19. The QoS constraints (C, A, RT, and R) values are generated arbitrarily from 1 to 1000 for each web service.

The experiments are implemented using a processor Intel(R) Core(TM) i5–3470 CPU @ 3.20 GHz, 3201 MHz, 4 Core(s), 4 Logical Processor(s) and 8.00 GB RAM. The control parameters were set to the values of 100, 400, and 10 for the ABC population size, maximum iteration, and CS population size, respectively. The parameters' values are chosen after extensive experiments, shown in the next subsection.

The ABC_CS is compared with two standard algorithms (ABC and CS) and three state-of-the-art swarm-based algorithms (OABC [14], MOHABC [9], SABC [16]) based on the following criteria:

- Best Fitness Value (BFV): Each algorithm was implemented for 30 different runs, where the best fitness value was obtained over all the runs.
- Average Fitness Value (AFV): Each algorithm was implemented for 30 different runs where the average fitness value was obtained.
- Average Execution Time (AET): It is obtained for each algorithm for the 30 different runs in MilliSeconds (MS).

6.2 Parameters Study

To test the main parameters' impact on the proposed algorithm, extensive experiments have been conducted with different values of the ABC's population size, maximum iteration, and the CS's population size. Three datasets were used (one for each dataset type) to examine the different combinations of these three parameters, which are dataset1 (represented by the small size datasets), dataset12 (represented by the medium-size datasets), and dataset19 (represented by the large size datasets). The ABC population (N) is allowed to test four different values: 50, 70, 100, or 150, and the maximum iteration (T) is allowed to test five different values: 100, 200, 300, 400, or 500, and the CS population (n) is allowed to test six different values: 5, 10, 15, 20, 25, or 30.

For each dataset, independent experiments were conducted with varying values of N, T, and n simultaneously to show the parameters' effect on the ABC_CS performance. Therefore, the total combinations of 120 parameter values were considered on each dataset. The ABC_CS was run for three independent times on each dataset for every combination, where the AFV and AET in MS were obtained.

Figs. 3–5 show how the ABC_CS performance was affected by the value selection of N while varying the value of other parameters T and n on small, medium, and large datasets. The parameters' effect on these experiments was studied and show that:

– Small size dataset: The performance of different combinations is so close because the problem is not complicated, and all combinations are able to achieve better results.
– Medium and large size datasets: The performance of the different combinations is variable, so we will highlight the main finding in the following.

Figs. 4–5 shows that the best result was obtained when N = 70, T = 400, and n = 10 for the medium-sized dataset. The best result was obtained for the large size dataset when N = 70, T = 500, and n = 5. Moreover, the table shows that while the value of N, T, and n increases, the AET increases simultaneously. Therefore, the value of N, T, and n were set to 100, 400, and 5 for the next experiments to obtain a sufficient tradeoff between fitness value and execution time.

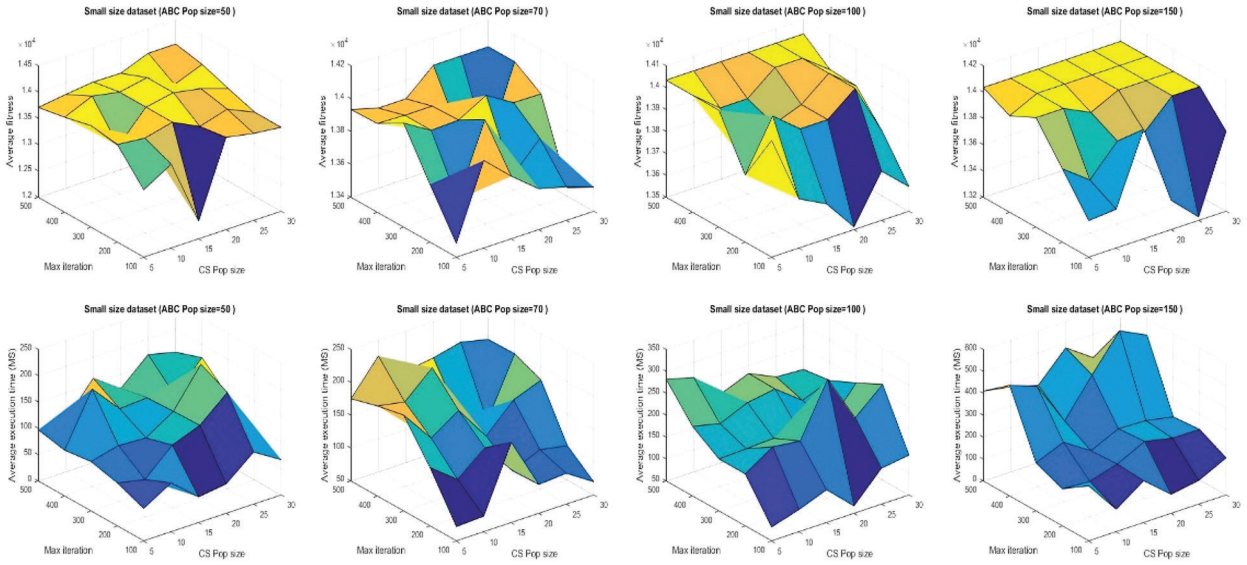

Figure 3: AFV and AET for dataset 1 when the ABC population size parameter is changed

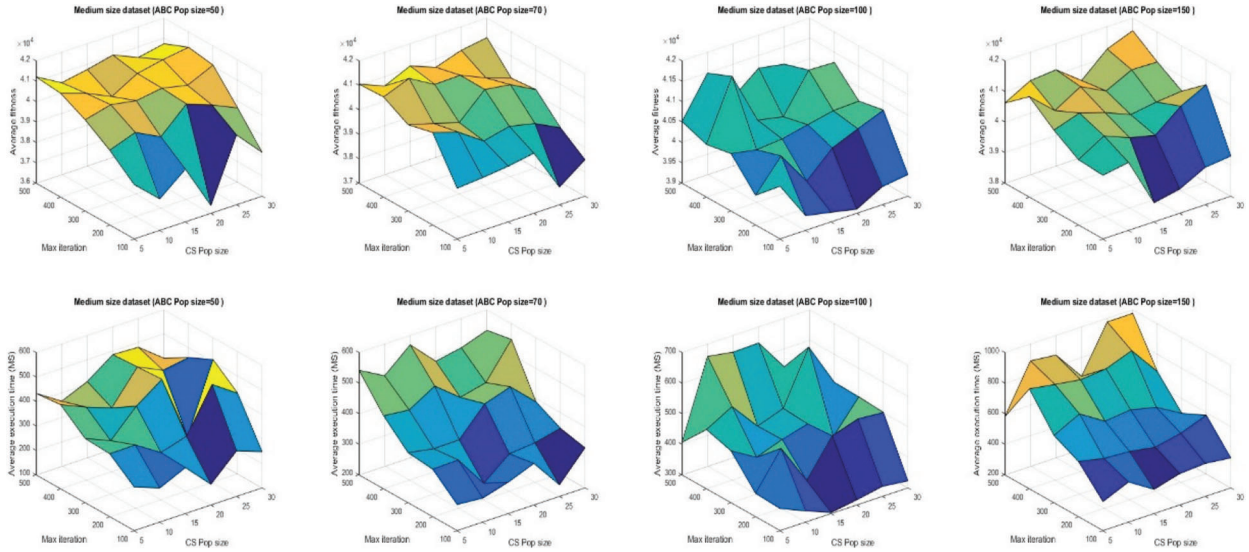

Figure 4: AFV and AET for dataset 12 when the ABC population size parameter is changed

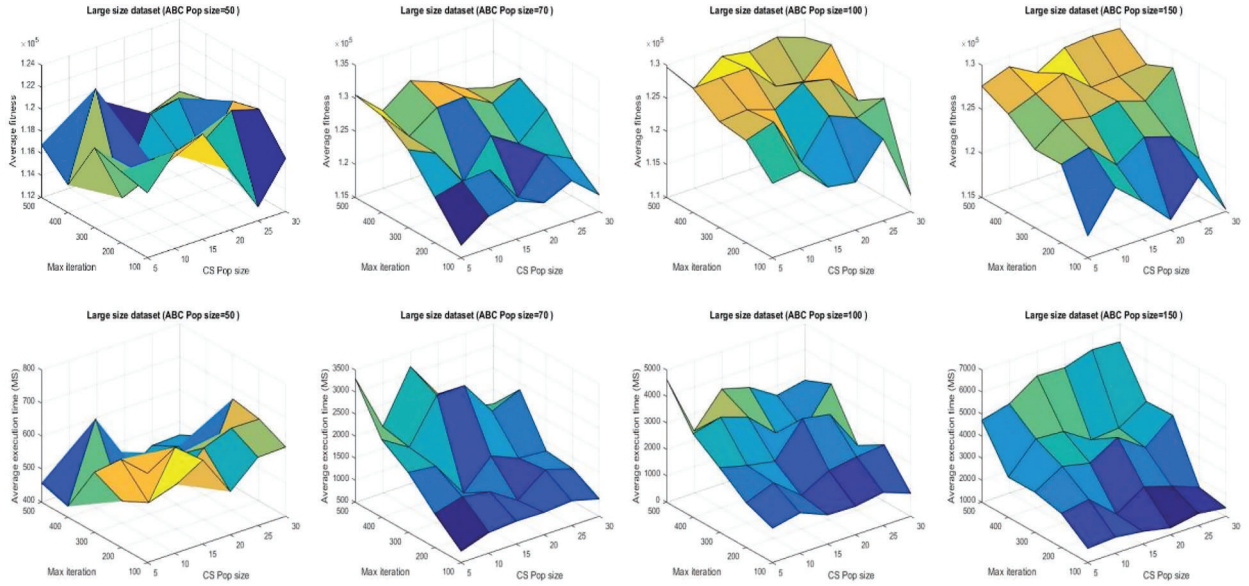

Figure 5: AFV and AET for dataset 19 when the ABC population size parameter is change**d**

Tab. 2 shows the rest of the control parameters.

6.3 Result and Discussion

To benchmark the ABC_CS performance, two comparison methodologies were adopted. Firstly, the comparisons were conducted between the ABC_CS and set against two standard algorithms (ABC and CS), and these comparisons' results are shown in Tab. 3. Secondly, the comparisons were conducted between the ABC_CS and three state–of–the–art algorithms from the literature (OABC [14], MOHABC [9], and SABC [16]), and these comparisons' results are shown in Tab. 4.

Table 2: Control parameters' value

Parameter	*Value*
Limits	100
th	0.5
P_a	0.5

Table 3: BFV, AFV, and AET values for ABC_CS compared to ABC, and CS

Dataset	Best fitness value			Average fitness value			Average execution time		
	ABC	CS	ABC_CS	ABC	CS	ABC_CS	ABC	CS	ABC_CS
dataset1	14034.43	14034.43	14034.43	13582.96	13082.6	**14012.10**	**82**	85	87
dataset2	13785.03	13785.03	**14307.40**	13267.33	13196.26	**14142.7**	89	**85**	101
dataset3	14264.25	14445.11	**14790.41**	13787.93	13848.65	**14648.16**	101	**95**	110
dataset4	14331.34	14621.24	**14844.86**	13600.17	13817.50	**14518.81**	**93**	101	120
dataset5	14457.69	14745.66	**15336.22**	14010.36	14320.61	**14991.79**	**96**	120	126
dataset6	15062.73	15080.35	**16099.90**	14713.19	14522.23	**15832.94**	**95**	103	130
dataset7	14891.64	14981.46	**15574.31**	14223.51	14342.24	**15385.48**	107	**106**	136
dataset8	14720.78	14862.54	**15771.44**	14279.82	14725.79	**15464.69**	**100**	109	137
dataset9	15005.45	15342.21	**16384.25**	14591.42	14452.99	**15964.31**	**86**	98	142
dataset10	15019.29	15339.22	**16359.06**	14457.35	14674.57	**15888.11**	**95**	102	143
dataset11	26433.71	26334.54	**27447.15**	25459.35	25592. 59	**26987.15**	137	**126**	150
dataset12	39229.49	39139.52	**41303.38**	38274.40	38287.50	**40584.81**	211	210	**162**
dataset13	50728.70	50835.56	**53339.69**	48614.73	48726.14	**52178.45**	286	254	**224**
dataset14	59303.92	58533.25	**64557.34**	57654.94	57533.49	**63096.24**	378	370	**366**
dataset15	71373.31	73543.25	**76847.28**	69116.94	69326.53	**74982.56**	480	465	**441**
dataset16	82515.75	83205.85	**90320.09**	80284.58	81265.85	**88003.73**	591	562	**552**
dataset17	95730.93	96029.35	**104215.63**	93189.27	93029.28	**101667.51**	715	703	**658**
dataset18	104142.91	112369.76	**115029.29**	100369.57	101639.77	**110363.55**	**791**	801	797
dataset19	115235.25	115295.35	**127963.64**	112289.22	112978.23	**124389.67**	**927**	935	929

6.3.1 Comparison of the ABC_CS and Standard Algorithms

In this work, a set of comparisons were carried out to benchmark the ABC_CS performance compared to the ABC and CS's. The values of the algorithms' control parameters were set as in the ABC_CS, which gives clarity and accuracy in evaluating the algorithms. Each algorithm was run for 30 times, and the BFV, AVF, and AET were noted for all the runs. The performance of the ABC_CS, ABC, and CS over the BFV, AFV, and AET are outlined in Tab. 3 (the best results are highlighted in bold).

The table shows that the ABC_CS outperforms the ABC and CS in terms of the BFV and AFV over all datasets. From the results, it is evident that combining the exploitation behavior of the ABC and exploration

of behavior of the CS has improved the ABC_CS performance. This good performance of ABC_CS results from avoiding the ABC getting stuck in a local minimum using the proposed enhancement. Furthermore, the results show that finding a good tradeoff between exploitation and exploration is very important when designing population-based metaheuristic algorithms. In terms of AVT, the ABC needs less execution time in nine datasets than both algorithms, and the CS needs less execution time in four datasets because of the fast convergence of ABC compared to the slow convergence speed of CS. In comparison, the ABC_CS needs less execution time on six datasets. These results support the above-mentioned claim that the ABC can get easily stuck in a local minimum while the CS does not. In addition, the proposed enhancement has the advantages of the ABC and CS, so it can achieve better results in a short time.

From the table, we can notice that the CS outperformed the ABC in thirteen datasets, while the ABC outperformed the CS in 6 datasets. These results show that the CS is more convenient for the WSC problem because of the global search mechanism that it has.

6.3.2 Comparison of the ABC_CS and State-of-the-art Algorithms

The performance of the ABC_CS compared to three state-of-the-art algorithms over the three evaluation criteria (best fitness values, average fitness values, and execution time values) are outlined in Tab. 4.

To make a fair comparison, the values of the algorithms' control parameters were set in the algorithms' preferences mentioned in their works.

The table shows that the ABC_CS obtains the BFV in 16 datasets, while SABC in three datasets. For the AFV, the results show that the ABC_CS outperforms other competitors in 14 datasets while SABC does so in five datasets. For the AVT, the results vary among the competitors. However, if we compare AET regarding the best results obtained for AFV between the ABC_CS and SABC, we can observe the following:

- The ABC_CS obtains the best results in terms of the AFV for 14 faster than SABC.
- The SABC obtains the best results in terms of the AFV for five datasets (dataset1, dataset12, dataset13, dataset14, and dataset15), but it needs more execution time compared to the ABC_CS. Furthermore, Tab. 5 depicts the normalization of the results of these five datasets in terms of AFV and AET. From the table, the enhancement percentage in AFV does not exceed 1% overall the datasets, while the different percentages in the AET are 1.9%, 14.6%, 15.7%, 8.1%, and 7.2% for dataset1, dataset12, dataset13, dataset14, and dataset15 respectively. These results show that the SABC is slower than the ABC_CS because of the extra procedures that have been added for the SABC to search the neighboring nodes. In fact, we can see that the ABC_CS is still the best if we consider the ratio of the results enhancement compared to time.

6.3.3 Result Significance Test

In this section, Tab. 6 presents the statistical test results with a p-value in order to be sure whether the results obtained by the ABC_CS compared to other algorithms differ or not using a Wilcoxon signed-rank test with a 0.05 significance level on AFV. In the table, "-", "+", "=" denotes that the competitors' performance is worse than, better than, or equal to the version of IBA, respectively. For the small size datasets, the difference between the ABC_CS compared to the ABC, CS, and OABC is statistically significant regarding small size datasets. However, the AFV of the ABC_CS significantly differs from the MOHABC, and SABC with 10, and 9 datasets, respectively. For medium-size datasets, the difference between the ABC_CS compared to ABC, CS, and OABC is statistically significant regarding medium size datasets. However, the AFV of the ABC_CS significantly differ from the MOHABC with three datasets.

Table 4: BFV, AFV, and AET values for ABC_CS compared to OABC, MOHABC, and SABC

Dataset	Best fitness value				Average fitness value				Average execution time			
	OABC	MOHABC	SABC	ABC_CS	OABC	MOHABC	SABC	ABC_CS	OABC	MOHABC	SABC	ABC_CS
dataset1	**14034.43**	**14034.43**	**14034.43**	**14034.43**	13955.76	13996.54	**14030.98**	14012.10	95	97	94	**87**
dataset2	14243.52	**14307.40**	**14307.40**	**14307.40**	13950.15	14042.64	14089.48	**14142.7**	220	126	104	**101**
dataset3	14439.24	**14790.41**	14757.05	**14790.41**	14070	14275.48	14467.51	**14648.16**	302	122	141	**110**
dataset4	14528.76	14550.74	14683.33	**14844.86**	14135.95	14235.03	14433.93	**14518.81**	416	**114**	128	120
dataset5	15242.81	15193.6	15110.27	**15336.22**	14759.82	14712.52	14786.30	**14991.79**	489	**121**	128	126
dataset6	15952.2	15787.76	15895.63	**16099.90**	15300.72	15215.68	15428.28	**15832.94**	612	**113**	140	130
dataset7	15066.81	15099.15	15290.58	**15574.31**	14658.27	14740.46	14873.87	**15385.48**	645	**114**	143	136
dataset8	15327.59	15252.1	15427.51	**15771.44**	14914.15	14824.33	15046.35	**15464.69**	721	**124**	140	137
dataset9	15698.4	15713.09	15870.63	**16384.25**	15315.53	15260.13	15436.61	**15964.31**	801	**129**	143	142
dataset10	15546.62	15764.98	15843.81	**16359.06**	15040	15193.58	15337.82	**15888.11**	857	163	155	**143**
dataset11	27185.88	27318.1	27309.69	**27447.15**	26428.9	26869.39	26891.32	**26987.15**	156	178	213	**150**
dataset12	40925.86	41162.14	**41480.34**	41303.38	40180.29	40411.4	**40872.83**	40584.81	211	256	295	**162**
dataset13	53542.62	53296.14	**53978.77**	53339.69	51649.75	52032.55	**52732.31**	52178.45	268	383	430	**224**
dataset14	62868.06	63866.37	**64666.88**	64557.34	61595.27	61947.55	**63176.44**	63096.24	344	**425**	507	366
dataset15	75333.25	75870.73	76726.70	**76847.28**	73348.98	74535.01	**75705.42**	74982.56	**371**	563	589	441
dataset16	87238.51	88635.27	88628.77	**90320.09**	85098.37	85679.46	87184.94	**88003.73**	**466**	767	786	552
dataset17	100921.44	101562.39	102985.17	**104215.63**	98447.01	99301.95	101224.38	**101667.51**	**560**	830	953	658
dataset18	111073.18	110410.59	112471.75	**115029.29**	107648.32	107479.15	110214.98	**110363.55**	**632**	807	1027	797
dataset19	122028.87	125145.18	125650	**127963.64**	119621.15	121500.80	123595.73	**124389.67**	**641**	1012	1140	929

Table 5: The results normalization for ABC_CS and SABC

	Average of fitness values		Average of execution time	
	SABC	ABC_CS	SABC	ABC_CS
dataset1	50.03%	49.97%	51.93%	48.07%
dataset12	50.18%	49.82%	64.55%	35.45%
dataset13	50.26%	49.74%	65.75%	34.25%
dataset14	50.03%	49.97%	58.08%	41.92%
dataset15	50.24%	49.76%	57.18%	42.82%

Table 6: The wilcoxon signed-rank significance test between ABC_CS compared to other algorithms

		ABC	*CS*	*OABC*	*MOHABC*	*SABC*
Small datasets	-	11	11	11	10	9
	+	0	0	0	1	2
	=	0	0	0	0	0
Medium datasets	-	5	5	5	3	4
	+	0	0	0	2	1
	=	0	0	0	0	0
Large datasets	-	3	3	3	3	0
	+	0	0	0	0	3
	=	0	0	0	0	0

Regarding the SABC, as shown in Tab. 4, it outperforms the ABC_CS on four datasets out of five. From Tab. 6, the difference between the ABC_CS compared to the SABC is statistically significant in dataset16 (where the ABC_CS outperforms SABC). For the other datasets where the SABC outperforms the ABC_CS, the difference between the SABC compared to the ABC_CS is statistically significant over the three datasets only (dataset12, dataset13, and dataset15). For large-size datasets, the difference between the ABC_CS compared to the ABC, CS, OABC, and MOHABC is statistically significant overall in terms of large-size datasets. However, the AFV of the ABC_CS doesn't significantly differ. According to the Wilcoxon test, we can conclude that the ABC_CS provides significantly better performance compared to other algorithms with a 0.05 significance level on AFV.

Regarding the experiments above' results, the ABC_CS exhibits better performance, which proves the ability of the proposed enhancement to overcome the ABC exploration drawback, introducing an outstanding balancing mechanism between exploration and exploitation. The results show the ability of the proposed enhancement to obtain better results relating to different problem sizes.

7 Conclusion

A hybrid variant of the ABC and CS was developed in this work to solve the WSC problem. The ABC is a well-known metaheuristic algorithm that is applied in different NP-hard problems with an outstanding level of performance. However, the ABC suffers from a slow convergence problem. The proposed ABC variant combines the CS algorithm exploration with the ABC's exploitation. In this case, the scout bees adopt the

Lévy flight before abandoning the bad solutions. The cuckoos collaborate with scout bees to avoid ABC getting stuck in a local minimum, where a set of cuckoos receives the poor bees and tries to enhance them. In the case of no improvement, the scout bees abandon these solutions. Nineteen datasets were used to evaluate the performance of the proposed algorithm. The experimental results were compared with two standard algorithms and three state-of-the-art swarm-based algorithms from the literature: The OABC, MOHABC, and OABC. The results showed that the hybrid algorithm is able to find better solutions than other algorithms in most of the datasets.

Moreover, we can see that when we added the Lévy flight to enhance the ABC searching (exploration) in the ABC_CS, the performance becomes better than a standard ABC algorithm. Therefore, the ABC algorithm searching for optimal/near-optimal solutions becomes more efficient at a lower execution time. In future, this work can be extended by employing different nature-inspired algorithms and their hybridization.

Acknowledgement: The authors would like to express their gratitude to Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia for providing the support.

Funding Statement: The authors extend their appreciation to Deputyship for research & Innovation, Ministry of Education in Saudi Arabia for funding this research work through the project number (IF-PSAU-2021/01/17793)

Conflicts of Interest: The authors declare that they have no competing interests.

References

[1] M. Ghobaei-Arani, A. A. Rahmanian, M. S. Aslanpour and S. E. Dashti, "CSA-WSC: Cuckoo search algorithm for web service composition in cloud environments," *Soft Computing*, vol. 22, no. 24, pp. 8353–8378, 2018.

[2] M. Ghobaei-Arani, S. Jabbehdari and M. A. Pourmina, "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach," *Future Generation Computer Systems*, vol. 78, pp. 191–210, 2018.

[3] T. Erl, "SOA principles of service design (the prentice hall service-oriented computing series from thomas Erl)," in *Prentice Hall PTR*, Hoboken, New Jersey, United States, 2007.

[4] F. Dahan, K. El Hindi and A. Ghoneim, "Enhanced artificial bee colony algorithm for QoS-aware web service selection problem," *Computing*, vol. 99, no. 5, pp. 507–517, 2017.

[5] I. Fister Jr, X. -S. Yang, I. Fister, J. Brest and D. Fister, "A brief review of nature-inspired algorithms for optimization," arXiv preprint arXiv:1307.4186, 2013.

[6] M. Mafarja and S. Mirjalili, "Whale optimization approaches for wrapper feature selection," *Applied Soft Computing*, vol. 62, pp. 441–453, 2018.

[7] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471, 2007.

[8] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (ABC) algorithm," *Applied Soft Computing*, vol. 8, no. 1, pp. 687–697, 2008.

[9] J. Zhou and X. Yao, "A hybrid approach combining modified artificial bee colony and cuckoo search algorithms for multi-objective cloud manufacturing service composition," *International Journal of Production Research*, vol. 55, no. 16, pp. 4765–4784, 2017.

[10] X. -S. Yang and S. Deb, "Engineering optimisation by cuckoo search," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 1, no. 4, pp. 330–343, 2010.

[11] M. Masdari, M. Nouzad and S. Ozdemir, "QoS-Driven metaheuristic service composition schemes: A comprehensive overview," *Artificial Intelligence Review*, vol. 55, pp. 1–68, 2021.

[12] L. Purohit and S. Kumar, "A study on evolutionary computing based web service selection techniques," *Artificial Intelligence Review*, vol. 54, no. 2, pp. 1117–1170, 2021.

[13] F. Seghir, "FDMOABC: Fuzzy discrete multi-objective artificial bee colony approach for solving the non-deterministic QoS-driven web service composition problem," *Expert Systems with Applications*, vol. 167, pp. 114413, 2021.

[14] S. Zhang, Y. Shao and L. Zhou, "Optimized web service composition using evolutionary computation techniques," *International Journal of Machine Learning and Computing*, vol. 11, no. 5, pp. 457–470, 2021.

[15] N. Arunachalam and A. Amuthan, "Integrated probability multi-search and solution acceptance rule-based artificial bee colony optimization scheme for web service composition," *Natural Computing*, vol. 20, no. 1, pp. 23–38, 2021.

[16] F. Dahan, H. Mathkour and M. Arafah, "Two-step artificial bee colony algorithm enhancement for QoS-aware Web service selection problem," *IEEE Access*, vol. 7, pp. 21787–21794, 2019.

[17] M. Chandra and R. Niyogi, "Web service selection using modified artificial bee colony algorithm," *IEEE Access*, vol. 7, pp. 88673–88684, 2019.

[18] F. Seghir, A. Khababa and F. Semchedine, "An interval-based multi-objective artificial bee colony algorithm for solving the web service composition under uncertain QoS," *The Journal of Supercomputing*, vol. 75, no. 9, pp. 5622–5666, 2019.

[19] N. Arunachalam and A. Amuthan, "Improved cosine similarity-based artificial bee colony optimization scheme for reactive and dynamic service composition," *Journal of King Saud University-Computer and Information Sciences*, pp. 270–281, 2018.

[20] J. Zhou and X. Yao, "Multi-objective hybrid artificial bee colony algorithm enhanced with lévy flight and self-adaption for cloud manufacturing service composition," *Applied Intelligence*, vol. 47, no. 3, pp. 721–742, 2017.

[21] P. Karthikeyan and G. Preethi, "Artificial bee colony and genetic algorithms in selecting and combining web services for enhancing QoS," *Design Engineering*, vol. 2021, pp. 6009–6021, 2021.

[22] S. Subbulakshmi, K. Ramar, A. E. Saji and G. Chandran, "Optimized web service composition using evolutionary computation techniques," in *Intelligent Data Communication Technologies and Internet of Things: Proc. of ICICI 2020*, Singapore, pp. 457–470, 2021.

[23] H. Wang, D. Yang, Q. Yu and Y. Tao, "Integrating modified cuckoo algorithm and creditability evaluation for QoS-aware service composition," *Knowledge-Based Systems*, vol. 140, pp. 64–81, 2018.

[24] F. Zhao, J. Bao and F. Ding, "A new integrating adaptive cuckoo search optimization algorithm for management service composition," in *Proc. of the Int. Conf. on Information Technology and Electrical Engineering 2018*, New York, United States, pp. 1–6, 2018.

[25] P. Thangaraj and P. Balasubramanie, "QoS based service composition for service computing using cuckoo search," *Parameters*, vol. 16, pp. 17, 2018.

[26] H. Kurdi, F. Ezzat, L. Altoaimy, S. H. Ahmed and K. Youcef-Toumi, "Multicuckoo: Multi-cloud service composition using a cuckoo-inspired algorithm for the internet of things applications," *IEEE Access*, vol. 6, pp. 56737–56749, 2018.

[27] S. Kouchi and H. Nacer, "Service selection in cloud computing environment by using cuckoo search," in *the Int. Conf. on Information, Communication & Cybersecurity*, Khourigba, Morocco, pp. 219–228, 2021.

[28] N. El Allali, M. Fariss, H. Asaidi and M. Bellouki, "A web service composition framework in a heterogeneous environment," *Journal of Ambient Intelligence and Humanized Computing*, vol. 13, pp. 1–25, 2022.

[29] Z. Wang, "Optimization of resource service composition in cloud manufacture based on improved genetic and ant colony algorithm," *Smart Innovation, Systems and Technologies*, vol. 268, pp. 183–198, 2022.

[30] T. Li, Y. Yin, B. Yang, J. Hou and K. Zhou, "A Self-learning bee colony and genetic algorithm hybrid for cloud manufacturing services," *Computing*, vol. 104, pp. 1–27, 2022.

[31] X. Teng, Y. Luo, T. Zheng and X. Zhang, "An improved whale optimization algorithm based on aggregation potential energy for qos-driven web service composition," *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–13, 2022.

[32] V. Rajendran, R. K. Ramasamy and W. -N. Mohd-Isa, "Improved eagle strategy algorithm for dynamic web service composition in the iot: A conceptual approach," *Future Internet*, vol. 14, no. 2, pp. 56, 2022.

[33] J. Dogani and F. Khunjush, "Cloud service composition using genetic algorithm and particle swarm optimization," in *In Proc. 2021 11th Int. Conf. on Computer Engineering and Knowledge (ICCKE)*, Mashhad, Iran, pp. 98–104, 2022.

[34] X. -S. Yang and S. Deb, "Cuckoo search via lévy flights," *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, Coimbatore, India, pp. 210–214, 2009.

[35] X. -S. Yang, *Nature-inspired optimization algorithms*, Academic Press, 2020.