mathematics

MDPI

Article

A New Neural Network Training Algorithm Based on Artificial Bee Colony Algorithm for Nonlinear System Identification

Ebubekir Kaya

Department of Computer Engineering, Engineering Architecture Faculty, Nevsehir Haci Bektas Veli Univesity, 50300 Nevsehir, Turkey; ebubekir@nevsehir.edu.tr or ebubekirkaya@yandex.com

Abstract: Artificial neural networks (ANNs), one of the most important artificial intelligence techniques, are used extensively in modeling many types of problems. A successful training process is required to create effective models with ANN. An effective training algorithm is essential for a successful training process. In this study, a new neural network training algorithm called the hybrid artificial bee colony algorithm based on effective scout bee stage (HABCES) was proposed. The HABCES algorithm includes four fundamental changes. Arithmetic crossover was used in the solution generation mechanisms of the employed bee and onlooker bee stages. The knowledge of the global best solution was utilized by arithmetic crossover. Again, this solution generation mechanism also has an adaptive step size. Limit is an important control parameter. In the standard ABC algorithm, it is constant throughout the optimization. In the HABCES algorithm, it was determined dynamically depending on the number of generations. Unlike the standard ABC algorithm, the HABCES algorithm used a solution generation mechanism based on the global best solution in the scout bee stage. Through these features, the HABCES algorithm has a strong local and global convergence ability. Firstly, the performance of the HABCES algorithm was analyzed on the solution of global optimization problems. Then, applications on the training of the ANN were carried out. ANN was trained using the HABCES algorithm for the identification of nonlinear static and dynamic systems. The performance of the HABCES algorithm was compared with the standard ABC, aABC and ABCES algorithms. The results showed that the performance of the HABCES algorithm was better in terms of solution quality and convergence speed. A performance increase of up to 69.57% was achieved by using the HABCES algorithm in the identification of static systems. This rate is 46.82% for the identification of dynamic systems.

Keywords: artificial bee colony algorithm; artificial neural network; global optimization; nonlinear system identification

MSC: 68T07

Citation: Kaya, E. A New Neural Network Training Algorithm Based on Artificial Bee Colony Algorithm for Nonlinear System Identification. *Mathematics* **2022**, *10*, 3487. https://doi.org/10.3390/math10193487

Academic Editors: Nicholas Christakis, George Kossioris and Mayur Patel

Received: 22 August 2022
Accepted: 17 September 2022
Published: 24 September 2022

1. Introduction

ANN is one of the most important artificial intelligence techniques. It has been used to solve many real-world problems [1]. Its popularity is a result of its strong structure. Security, science, engineering, medical science, finance, education, energy, and manufacturing are some of the areas where ANN is used. One of the important capabilities of ANN is the learning feature. It can provide modeling of problems using existing data and can give information about data that it has never seen. The training process of ANN is very important. Successful modeling of a problem is directly related to the successful training process of ANN. One of the artificial intelligence approaches used extensively in ANN training is meta-heuristic algorithms [2–4].

Meta-heuristic algorithms have been used successfully in solving many real-world problems [5–8]. The success of meta-heuristic algorithms has increased the interest in this field. Currently, there are more than 200 meta-heuristic algorithms in the literature. The

equilibrium optimizer (EO) [9], the marine predators algorithm (MPA) [10], the slime mould algorithm (SMA) [11], the reptile search algorithm (RSA) [12], the dandelion optimizer [13], the runge kutta optimizer (RUN) [14], and weighted mean of vectors (INFO) [15] are some of the current meta-heuristic algorithms recommended in 2020 and beyond. Artificial bee colony (ABC) [16,17], cuckoo search (CS) [18], particle swarm optimization (PSO) [19], differential evolution algorithm (DE) [20] and flower pollination algorithm (FPA) [21] are some of the very popular meta-heuristic algorithms proposed in recent years.

One of the popular meta-heuristic algorithms used in ANN training is the ABC algorithm. The ABC algorithm is used to solve many real-world problems [22]. The strong global and local convergence ability of the ABC algorithm is one of the important reasons for its popularity. This strong structure of the ABC algorithm has also enabled it to be used extensively in ANN-based studies. Some ANN studies based on the ABC algorithm are as follows: Karaboga and Akay [23] evaluated the performance of the ABC algorithm in ANN training and compared it with different algorithms. They reported that the ABC algorithm was successful in training ANN. Karaboga et al. [24] trained feed-forward neural networks (FFNNs) using the ABC algorithm. Ozturk and Karaboga [25] proposed a hybrid training algorithm based on Levenberg–Marquardt (LM) and ABC algorithm. Karaboga and Ozturk [26] evaluated its performance on classification problems by training FFNN with the ABC algorithm. Kaya [27] compared the performance of several meta-heuristic algorithms in ANN training for nonlinear system identification. It was reported that the ABC algorithm is among the most successful training algorithms. Uzlu et al. [28] used an approach based on the ABC algorithm and ANN for the prediction of hydroelectric generation in Turkey. They stated that their proposed approach was effective. Xu et al. [29] proposed a modified ABC algorithm to train FFNN. Kaya and Baştemur Kaya [30] proposed a novel neural network training algorithm called artificial bee colony algorithm based on effective scout bee stage (ABCES) for the identification of nonlinear static systems. Ghanem and Jantan [31] introduced a hybrid training algorithm based on the ABC algorithm and monarch butterfly optimization for cyberattack classification applications. Shah et al. [32] presented a quick gbest guided artificial bee colony algorithm to train FFNN for Saudi stock market prices prediction. They reported that their proposed approach was more effective than typical computational algorithms.

Many variants have been proposed to obtain more effective solutions with the ABC algorithm. At the same time, hybrid approaches have also been suggested using different meta-heuristic algorithms and some of their features. Yildiz [33] developed a new hybrid algorithm based on the ABC algorithm and the Taguchi method for a structural design optimization of a vehicle component and a multi tool milling optimization problem. Karaboga and Kaya [34] proposed a hybrid ABC algorithm using arithmetic crossover to train ANFIS. Jadon et al. [35] suggested a novel hybrid algorithm (HABCDE) based on the ABC algorithm and DE. The performance of HABCDE was compared with different variant and meta-heuristic approaches for solving various science and engineering optimization problems. Li et al. [36] presented a hybrid feature selection algorithm (MFABC) based on a discrete ABC algorithm for Parkinson's disease diagnosis. The performance of MFABC was compared with meta-heuristic approaches such as PSO, GWO, GA, ABC, QPSO, DE, FIREFLY, BAT, FWA and BBO. It was reported that MFABC was effective. Kefayat et al. [37] suggested a hybrid optimization approach including ant colony optimization and the ABC algorithm for probabilistic optimal placement and sizing of distributed energy resources. Duan et al. [38] proposed a hybrid algorithm based on the ABC algorithm and the quantum evolutionary algorithm to solve continuous optimization problems. Awadallah et al. [39] used the hill climbing optimizer in the process of the employed bee and proposed a hybrid algorithm (HABC) based on the hill climbing optimizer and ABC algorithm for a nurse rostering problem. Apart from these, there are also different hybrid approaches [40–46].

In light of the above information, it can be seen that the ABC algorithm has an important area of use in ANN training. To improve the performance of the ABC algorithm in ANN training, variants possessing improved global and local convergence capability are

needed. Therefore, in this study, the HABCES algorithm, which a powerful variant of the ABC algorithm, is proposed. For the analysis of the performance of the HABCES algorithm in ANN training, it was applied for modeling parity problems and the identification of nonlinear static and dynamic systems. This study makes important contributions to the literature and has many innovative aspects. These are summarized below:

- The study uses one of the most effective variants of the ABC algorithm proposed for ANN training. The HABCES algorithm is proven to be effective by comparing it with different variants.
- It is one of the most comprehensive studies on the identification of nonlinear static and dynamic systems.
- The study uses an important variant in which both global convergence and local convergence ability are developed. It can be used to solve many different real-world problems.
- Unlike other variants of ABC algorithms in the literature, the related algorithm is improved by using adaptive arithmetic crossover, adaptive step size, an adaptive limit parameter, and an effective scout bee solution generation mechanism.

2. Materials and Methods

2.1. Standard Artificial Bee Colony (ABC) Algorithm

The ABC algorithm models the foraging behavior of honey bees [16,17]. Unlike the behavior of real bees, the ABC algorithm makes some assumptions. The ABC algorithm includes three different types of bee, including employed bees, onlooker bees and scout bees. Half of a colony consists of employed bees. The other half includes onlooker bees. The position of a food source corresponds to a candidate solution of a problem. The nectar amount of a food source corresponds to the solution quality. The ABC algorithm has three basic control parameters. These are colony size, limit, and maximum number of generations.

In the ABC algorithm, the process starts with the random determination of the positions of the food sources using (1). x_j^{min} is the lower bound and x_j^{max} is the upper bound. x_i corresponds to the ith solution. To produce new solutions in the employed bee onlooker bee phases, the solution generation mechanisms given in (2) are used. Here, $\varnothing_{ij}$ is a random number in the range $[-1, 1]$ and k is an integer number in the range [1, number of employed bees].

$$x_{ij} = x_j^{min} + rand(0,1)\left(x_j^{max} - x_j^{min}\right) \tag{1}$$

$$v_{ij} = x_{ij} + \varnothing_{ij}\left(x_{ij} - x_{kj}\right) \tag{2}$$

$$P_i = \frac{fitness_i}{\sum_{j=1}^{SN} fitness_j} \tag{3}$$

A comparison is made as to whether the nectar amount of the candidate source is better than the previous one. If the nectar amount of the candidate source is better, the information of the previous source is deleted from the memory and the information of the candidate source is written. In the standard ABC algorithm, a selection process is carried out using the roulette wheel method. The selection probability of sources is computed by (3). Here, $fitness_i$ represents the quality of the ith source. SN is the number of employed bees.

The ABC algorithm has a failure counter. If the value of the failure counter reaches the limit value, a new solution is randomly generated by the scout bees using (1). This process is repeated until the maximum number of generations is reached.

2.2. Hybrid Artificial Bee Colony Algorithm Based on Effective Scout Bee Stage (HABCES)

The hybrid artificial bee colony algorithm based on effective scout bee stage (HABCES) improves both the global and local search capability of the standard ABC algorithm. The aABC algorithm [34] and ABCES algorithm [30] are two important variants of the ABC

algorithm. The HABCES algorithm was created by combining the innovative features of these variants and adding new additional features.

To increase the local search capability of the ABC algorithm, two fundamental innovations were made in the structure of the aABC algorithm. Firstly, arithmetic crossover was used in the solution generation mechanisms of the employed and onlooker bee phases. An arithmetic crossover was applied between the instantaneous solution and the global best solution. The second is the adaptive adjustment of the step size according to the failure counter. To strengthen the local search capability of the HABCES algorithm, the solution search mechanism of the aABC algorithm specified in (4) and (5) was used. γ is the crossover rate and α is the adaptivity coefficient. One of the major drawbacks of the aABC algorithm is the manual setting of these parameters. The optimum values of the γ and α parameters may change according to the problem type. For the best results, it is necessary to try all alternatives. This disadvantage is eliminated with the HABCES algorithm. In each iteration, the values of γ and α are adjusted adaptively according to the number of iterations as noted in (6) and (7). *iter* is the current number of iterations and *maxCycle* is the maximum number of iterations. As can be understood from (6), the effect of the global best solution on the new solution increases as the number of iterations increases. Namely, better solutions are sought around the global best solution. At first, the effect of the global best solution is minimal. With this approach, new solutions are prevented from being similar to the global best solution. (7) shows the adaptive adjustment of the step size (α). α changes according to the number of iterations and approaches from 0 to 1. Its value is initially 0. It reaches 1 when *iter* and *maxCycle* are equal. This approach allows searching for a new solution in remote region of the existing solution in the initial iterations. As the number of iterations increases, the search region narrows.

$$v_{ij} = x_{ij}\gamma + x_{g_j}(1-\gamma) + B_i \varnothing_{ij}\left(x_{ij} - x_{kj}\right) \tag{4}$$

$$B_i = \left(\frac{1}{1+trial_i}\right)^{\alpha} \tag{5}$$

$$\gamma = rand * \frac{maxCycle - iter}{maxCycle} \tag{6}$$

$$\alpha = \frac{iter}{maxCycle} \tag{7}$$

$$Limit = 1 + w \times D \times FoodNumber \times \left(\frac{maxCycle - iter}{maxCycle}\right) \tag{8}$$

$$v_{ij} = \begin{cases} x_{ij}\varepsilon + x_{g_j}(1-\epsilon), & r1 < r2 \\ x_{ij} + x_{ij}\,\delta, & other \end{cases} \tag{9}$$

$$r2 = rand + \left(\frac{maxCycle - iter}{maxCycle}\right) \tag{10}$$

Limit is one of the important control parameters of the ABC algorithm. When the *limit* value reaches the failure counter, a random new solution is generated in the scout bee stage. Random generation of the new solution means abandoning previously achieved gains. This is a situation that negatively affects the global convergence ability of the algorithm. In the ABC algorithm, the value of the *limit* parameter is fixed. Large or small *limit* values can turn into a disadvantage depending on the development of the solution. The two main issues mentioned here are solved in the ABCES algorithm. As stated in (8), the *limit* value is dynamically adjusted depending on the current iteration number. Here, w is a random number in the range [0,1]. D is the number of parameters of the problem to be optimized. The ABCES algorithm utilizes a solution generation mechanism that uses the current solution and the global best solution instead of randomly determining the solutions in the scout bee phase. In (9) and (10), $r1$ and ϵ are random numbers in the range [0,1]. δ is

the step size and is accepted as 0.01. In this way, it is ensured that the gains obtained are transferred to the new generations. At the same time, the global convergence capability of the algorithm has been increased. The two stated strengths of the ABCES algorithm are integrated into the HABCES algorithm to strengthen its global search capability.

In light of this information, the basic architecture of the HABCES algorithm can be summarized as follows:

- Arithmetic crossover is used in the solution generation mechanism of the employed and onlooker bee stages and an adaptive local search is adopted.
- Arithmetic crossover and adaptivity parameters are adjusted dynamically.
- Limit, one of the important control parameters, is dynamically adjusted depending on the number of iterations. Its value changes with each iteration.
- In the scout bee stage, instead of randomly generating new solutions, a strong solution generation mechanism including the global best solution and the current solution is used.

3. Results

3.1. Applications on the Solution of the Global Optimization Problems

In this section, the performance of the HABCES algorithm on solving global optimization problems was evaluated. For this purpose, the 43 benchmark test functions given in Table 1 were used. The first 20 (F1–F20) test functions are suitable for high-dimensional problems. The other test functions (F21–F43) are low-dimensional optimization problems. For high-dimensional problems, colony size and maximum number of generations were taken as 50 and 500, respectively. The maximum number of generations for low-dimensional optimization problems is 100. Each application was run 30 times, and the mean fitness values were calculated.

Table 1. Benchmark functions used for solving global optimization problems.

Function	Function Name	Formulation
F1	Sphere	$f(x) = \sum_{i=1}^{n} x_i^2$
F2	Exponential	$f(x) = -exp\left(-0.5 \sum_{i=1}^{n} x_i^2\right)$
F3	High Conditioned Elliptic	$f(x) = \sum_{i=1}^{n} (10^6)^{\frac{i-1}{n-1}} x_i^2$
F4	Ackley	$f(x) = -20exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$
F5	Schwefels 1.2	$f(x) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2$
F6	Step	$f(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$
F7	Sum of Different Powers	$f(x) = \sum_{i=1}^{n} \lvert x_i \rvert^{i+1}$
F8	Griewank	$f(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
F9	Sum Squares	$f(x) = \sum_{i=1}^{n} i x_i^2$
F10	Rastrigin	$f(x) = \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$

Table 1. *Cont.*

Function	Function Name	Formulation
F11	Penalized 1	$f(x) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right] + (y_n-1)^2\right\} + \sum_{i=1}^{n} u(x_i,\, a,\, k,\, m)$ $y_i = 1 + \frac{1}{4}(x_i+1)$ $u(x_i,\, a,\, k,\, m) = \begin{cases} k(x_i-a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i-a)^m & x_i < -a \end{cases}$
F12	Penalized 2	$f(x) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right] + (x_n-1)^2\left[1+\sin^2(2\pi x_n)\right]\right\} + \sum_{i=1}^{n} u(x_i,\, a,\, k,\, m)$ $u(x_i,\, a,\, k,\, m) = \begin{cases} k(x_i-a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i-a)^m & x_i < -a \end{cases}$
F13	Rosenbrock	$f(x) = \sum_{i=1}^{n-1}\left[100\left(x_{i+1}-x_i^2\right)^2 + (x_i-1)^2\right]$
F14	Levy	$f(x) = \sin^2(\pi w_1) + \sum_{i=1}^{n-1}(w_i-1)^2\left[1+10\sin^2(\pi w_i+1)\right] + (w_d-1)^2 + \left[1+\sin^2(2\pi w_d)\right] w_i = 1 + \frac{x_i-1}{4}$
F15	Zakharov	$f(x) = \sum_{i=1}^{n} {x_i}^2 + \left(\sum_{i=1}^{n} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{n} 0.5 i x_i\right)^4$
F16	Rotated Hyper-Ellipsoid	$f(x) = \sum_{i=1}^{n}\sum_{j=1}^{i} x_j^2$
F17	Styblinski–Tang	$f(x) = \frac{1}{2}\sum_{i=1}^{n}\left(x_i^4 - 16x_i^2 + 5x_i\right)$
F18	Cigar	$f(x) = {x_1}^2 + 10^6 \sum_{i=2}^{n} {x_i}^2$
F19	Qing	$f(x) = \sum_{i=1}^{n}\left(x_i^2 - i\right)^2$
F20	Yang 1	$f(x) = \sum_{i=1}^{n} \varepsilon_i \lvert x_i\rvert^i$
F21	Bartels Conn	$f(x) = \lvert x_1^2 + x_2^2 - x_1 x_2\rvert + \lvert\sin(x_1)\rvert + \lvert\cos(x_2)\rvert$
F22	Beale	$f(x) = (1.5 - x_1 + x_1 x_2)^2 + \left(2.25 - x_1 + x_1 {x_2}^2\right)^2 + \left(2.625 - x_1 + x_1 {x_2}^3\right)^2$
F23	Bohachevsky 1	$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$
F24	Bohachevsky 2	$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3$
F25	Camel Function—Three Hump	$f(x) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1 x_2 + x_2^2$
F26	Egg Crate	$f(x) = x_1^2 + x_2^2 + 25\left(\sin^2(x_1) + \sin^2(x_2)\right)$
F27	Himmelblau	$f(x) = \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + {x_2}^2 - 7\right)^2$
F28	Leon	$f(x) = 100\left(x_2 - x_1^2\right)^2 + (1-x_1)^2$
F29	Matyas	$f(x) = 0.26\left(x_1^2 + x_2^2\right) - 0.48 x_1 x_2$
F30	Mishra 3	$f(x) = \sqrt{\left\lvert\cos\sqrt{\lvert x_1^2 + x_2^2\rvert}\right\rvert} + 0.01(x_1 + x_2)$
F31	Price 1	$f(x) = (\lvert x_1\rvert - 5)^2 + (\lvert x_2\rvert - 5)^2$
F32	Price 2	$f(x) = 1 + \sin^2 x_1 + \sin^2 x_2 - 0.1 e^{-x_1^2 - x_2^2}$
F33	Scahffer 1	$f(x) = 0.5 + \frac{\sin^2\left(x_1^2 + x_2^2\right)^2 - 0.5}{1 + 0.001\left(x_1^2 + x_2^2\right)^2}$
F34	Scahffer 2	$f(x) = 0.5 + \frac{\sin^2\left(x_1^2 - x_2^2\right)^2 - 0.5}{1 + 0.001\left(x_1^2 + x_2^2\right)^2}$
F35	Trecanni	$f(x) = x_1^4 + 4x_1^3 + 4x_1^2 + x_2^2$

Table 1. *Cont.*

Function	Function Name	Formulation
F36	Branin's RCOS 1	$f(x) = \left(x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$
F37	Easom	$f(x) = -\cos(x_1)\cos(x_2)e^{[-(x_1-\pi)^2-(x_2-\pi)^2]}$
F38	Six Hump Camel Function	$f(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$
F39	Cube	$f(x) = 100(x_2 - x_1^3)^2 + (1 - x_1)^2$
F40	Hartmann 3	$f(x) = -\sum_{i=1}^{4} a_i e^{(-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2)}$
F41	Hartmann 6	$f(x) = -\sum_{i=1}^{4} a_i e^{(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2)}$
F42	Colville	$f(x) = 100(x_1 - x_2^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) + 19.8(x_2 - 1)(x_4 - 1)$
F43	Miele Cantrell	$f(x) = (e^{-x_1} - x_2)^4 + 100(x_2 - x_3)^6 + (\tan(x_3 - x_4))^4 + x_1^8$

The performance of the HABCES algorithm was compared with the standard ABC, aABC and ABCES algorithms. The results obtained for the first 20 test functions are given in Table 2. In addition to the results, Table 2 also includes the search range, dimensions, and optimal values. The best results on all test functions were obtained with the proposed HABCES algorithm. A value of 5.7×10^{-2} was reached using the HABCES algorithm for F1. For the ABCES and aABC algorithms, the fitness values were at 10^{-6} and 10^{-5} levels. Only the HABCES algorithm reached the optimal value for F2. After the HABCES algorithm, ABCES had the best fitness value, with a standard deviation value of 3.0×10^{-7}. Although very high error values were obtained with the ABC, ABCES and aABC algorithms for F3, very successful results were found when utilizing the HABCES algorithm. The best fitness value for F4 was obtained by using HABCES algorithm as 3.0×10^{-5}. Due to the difficulty of the problem for F5, high fitness values were generally achieved in all algorithms. The best result was obtained using the HABCES algorithm. The optimal value of F6 is 0, and the optimal value was achieved by the HABCES and ABCES algorithms. Interestingly, the least successful result was obtained using the aABC algorithm for this function. The best result for F7 was found to be 9.1×10^{-18} and it was obtained using the HABCES algorithm. The fitness value of the HABCES algorithm for F8 was at the 10^{-3} level. For other algorithms, it was greater than 10^{-3}. The HABCES algorithm was very successful for F9. Its fitness value was 2.4×10^{-10}. The closest result for F9 was obtained using the ABCES algorithm. The fitness value of the ABCES algorithm was 1.2×10^{-5}. While the fitness value of the HABCES algorithm was at the 10^{-2} level for F10, it was greater than 1 for the other algorithms. The best result after the HABCES algorithm was obtained when using the ABC algorithm. The best fitness value found for F11 was 5.9×10^{-11} and was obtained when utilizing the HABCES algorithm. A value of 2.2×10^{-10} was obtained by using the HABCES algorithm for F12. The fitness values of the other algorithms were greater than 10^{-5}. The fitness values found for F13 was between 22 and 32. The best result was achieved when using the HABCES algorithm. The least successful result was obtained with ABC algorithm.

Table 2. Comparison of the results of the ABC, aABC, ABCES and HABCES algorithms for solving high-dimensional global optimization problems.

Function	Search Range	D	Optimal Value	ABC		ABCES		aABC		HABCES (Proposed)	
				Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
F1	$[-5, 5]$	30	0	3.0×10^{-4}	1.1×10^{-3}	2.5×10^{-6}	2.4×10^{-6}	2.2×10^{-5}	6.7×10^{-5}	$\mathbf{5.7 \times 10^{-12}}$	$\mathbf{8.2 \times 10^{-12}}$
F2	$[-1, 1]$	30	-1	-0.999972	8.4×10^{-5}	-1	3.0×10^{-7}	-0.999995	1.0×10^{-5}	$\mathbf{-1}$	**0**
F3	$[-10, 10]$	30	0	4.8×10^{-1}	1.15215	2.4×10^{-2}	7.2×10^{-2}	2.7×10^{-1}	6.9×10^{-1}	$\mathbf{9.0 \times 10^{-9}}$	$\mathbf{2.0 \times 10^{-8}}$
F4	$[-32, 32]$	30	0	5.3×10^{-1}	6.0×10^{-1}	1.0×10^{-1}	7.5×10^{-2}	3.7×10^{-1}	1.59738	$\mathbf{3.0 \times 10^{-5}}$	$\mathbf{1.4 \times 10^{-5}}$
F5	$[-10, 10]$	30	0	1.8×10^{2}	3.6×10^{1}	7.7×10^{1}	3.1×10^{1}	1.1×10^{2}	2.8×10^{1}	$\mathbf{4.7 \times 10^{1}}$	$\mathbf{2.1 \times 10^{1}}$
F6	$[-100, 100]$	30	0	1.3×10^{-1}	4.3×10^{-1}	**0**	**0**	2.7×10^{-1}	5.7×10^{-1}	**0**	**0**
F7	$[-1, 1]$	30	0	9.4×10^{-9}	2.9×10^{-8}	1.5×10^{-12}	2.8×10^{-12}	6.9×10^{-12}	2.6×10^{-11}	$\mathbf{9.1 \times 10^{-18}}$	$\mathbf{8.0 \times 10^{-18}}$
F8	$[-600, 600]$	30	0	1.7×10^{-1}	2.8×10^{-1}	1.7×10^{-2}	2.2×10^{-2}	5.3×10^{-2}	6.7×10^{-2}	$\mathbf{2.5 \times 10^{-3}}$	$\mathbf{6.8 \times 10^{-3}}$
F9	$[-10, 10]$	30	0	2.8×10^{-1}	1.48421	1.2×10^{-5}	9.3×10^{-6}	2.1×10^{-3}	6.8×10^{-3}	$\mathbf{2.4 \times 10^{-10}}$	$\mathbf{7.2 \times 10^{-10}}$
F10	$[-5, 5]$	30	0	4.46395	1.67945	5.64602	1.7277	6.35917	2.26352	$\mathbf{6.9 \times 10^{-2}}$	$\mathbf{2.3 \times 10^{-1}}$
F11	$[-50, 50]$	30	0	1.8×10^{-4}	4.0×10^{-4}	1.9×10^{-5}	1.6×10^{-5}	4.3×10^{-5}	9.6×10^{-5}	$\mathbf{5.9 \times 10^{-11}}$	$\mathbf{2.0 \times 10^{-10}}$
F12	$[-50, 50]$	30	0	4.9×10^{-3}	2.0×10^{-2}	1.4×10^{-4}	1.0×10^{-4}	8.3×10^{-4}	2.5×10^{-3}	$\mathbf{2.2 \times 10^{-10}}$	$\mathbf{2.9 \times 10^{-10}}$
F13	$[-5, 5]$	30	0	3.2×10^{1}	2.8×10^{1}	2.7×10^{1}	2.5×10^{1}	2.4×10^{1}	$\mathbf{2.0 \times 10^{1}}$	$\mathbf{2.2 \times 10^{1}}$	2.1×10^{1}
F14	$[-10, 10]$	30	0	7.1×10^{-4}	2.1×10^{-3}	3.8×10^{-5}	3.9×10^{-5}	7.3×10^{-5}	1.3×10^{-4}	$\mathbf{1.1 \times 10^{-10}}$	$\mathbf{2.3 \times 10^{-10}}$
F15	$[-5, 10]$	30	0	3.1×10^{2}	3.6×10^{1}	9.3×10^{1}	3.4×10^{1}	2.3×10^{2}	3.9×10^{1}	$\mathbf{8.0 \times 10^{1}}$	$\mathbf{2.5 \times 10^{1}}$
F16	$[-100, 100]$	30	0	1.33847	7.19865	1.4×10^{-5}	1.8×10^{-5}	2.0×10^{-3}	4.7×10^{-3}	$\mathbf{1.2 \times 10^{-10}}$	$\mathbf{2.4 \times 10^{-10}}$
F17	$[-5, 5]$	30	-1174.98	-1142.88	33.1328	-1171.84	6.70843	-1174.18	2.03502	$\mathbf{-1174.98}$	**0**
F18	$[-10, 10]$	30	0	2.9×10^{1}	6.2×10^{1}	8.7×10^{-2}	2.7×10^{-1}	3.0×10^{1}	1.1×10^{2}	$\mathbf{3.1 \times 10^{-7}}$	$\mathbf{3.0 \times 10^{-7}}$
F19	$[-100, 100]$	30	0	3.92817	6.78767	6.3×10^{-1}	6.3×10^{-1}	1.9×10^{-1}	3.7×10^{-1}	$\mathbf{2.7 \times 10^{-4}}$	$\mathbf{6.8 \times 10^{-4}}$
F20	$[-5, 5]$	30	0	5.7×10^{-2}	3.5×10^{-2}	1.5×10^{-3}	1.1×10^{-3}	4.1×10^{-2}	1.6×10^{-2}	$\mathbf{2.3 \times 10^{-4}}$	$\mathbf{2.6 \times 10^{-4}}$

The most successful algorithm for F14 was the HABCES algorithm. Generally, large fitness values were achieved for F15. The best result was 8.0×10^1 and was obtained when using the HABCES algorithm. The most successful algorithm after HABCES was ABCES. Its fitness value was 9.3×10^1. For F16, the difference between the fitness values of the HABCES algorithm and the ABC algorithm was very large. The HABCES algorithm was better than the other algorithms tested here. The optimal value of F17 was -1174.98. Only the HABCES algorithm reached this value. The best result after the HABCES algorithm was obtained with aABC algorithm. The best fitness value for F18 was found to be 3.1×10^{-7}, and was obtained when using the HABCES algorithm. For F19 and F20, the HABCES algorithm was more effective than the other algorithms.

The results obtained for functions between F21 and F43 are presented in Table 3. The optimum value was achieved with all algorithms for F21, F33, F37 and F38. The best result for F22 was found to be 4.4×10^{-6}, and was obtained when using the HABCES algorithm. The optimum value was achieved when utilizing the ABCES, aABC and the HABCES algorithms for F23. The fitness value of the ABC algorithm was 7.1×10^{-15}. The best result for F24 was found with the HABCES algorithm. The HABCES algorithm achieved the optimal value here. After the HABCES algorithm, the best result was obtained when using the aABC algorithm. Effective fitness values were obtained when using the HABCES and aABC algorithms for F25. Although the HABCES algorithm had the best fitness value for F26, the results obtained when using the other algorithms were also successful. The HABCES algorithm was better than the other algorithms for F27. The best fitness value for F28 was obtained when using the aABC algorithm. After the aABC algorithm, the best result was found by utilizing the HABCES algorithm. The best results for F29 were obtained via the HABCES and aABC algorithms. The fitness values achieved were 7.5×10^{-6} and 3.2×10^{-5}, respectively. The optimal value of F30 is -0.18465. The closest result to the optimal value was obtained when using the HABCES algorithm. The best result for F31 was found to be 1.7×10^{-18}, and was obtained when using the HABCES algorithm. The least successful algorithm was ABCES. The optimal value of F32 is 0.9. The best result for F32 was obtained by using the aABC algorithm. The fitness value of the HABCES algorithm was 0.906667. The fitness values obtained with the aABC and HABCES algorithms for F34 were both 0. Other algorithms were less successful. Generally effective fitness values were obtained for F35. The most successful was the HABCES algorithm. The best fitness value for F36 was found to be 0.397887 by utilizing the aABC and the HABCES algorithms. The fitness value of the HABCES algorithm for F39 was 1.2×10^{-3}, and was the best result. For F40, the fitness value of all algorithms is the same. The best value for F41 was obtained when using the ABC algorithm. For F42, the best fitness value was found to be 4.1×10^{-1}, and was obtained with the HABCES algorithm. The performance of the HABCES algorithm for F43 was better than that of other algorithms. The fitness value obtained when using the HABCES algorithm was 1.8×10^{-7}. The best result after the HABCES algorithm was found with the ABCES algorithm.

Table 3. Comparison of the results of the ABC, aABC, ABCES and the HABCES algorithms when solving low-dimensional global optimization problems.

Function	Search Range	D	Optimal Value	ABC		ABCES		aABC		HABCES (Proposed)	
				Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
F21	$[-500, 500]$	2	1	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**
F22	$[-4, 4]$	2	0	1.5×10^{-3}	1.5×10^{-3}	3.5×10^{-4}	7.1×10^{-4}	1.2×10^{-4}	2.2×10^{-4}	$\mathbf{4.4 \times 10^{-6}}$	$\mathbf{1.4 \times 10^{-5}}$
F23	$[-100, 100]$	2	0	7.1×10^{-15}	3.5×10^{-14}	**0**	**0**	**0**	**0**	**0**	**0**
F24	$[-100, 100]$	2	0	1.1×10^{-9}	4.8×10^{-9}	2.3×10^{-10}	8.5×10^{-10}	4.8×10^{-14}	2.6×10^{-13}	**0**	**0**
F25	$[-5, 5]$	2	0	1.7×10^{-9}	5.8×10^{-9}	1.4×10^{-10}	5.4×10^{-10}	1.1×10^{-17}	8.1×10^{-18}	$\mathbf{2.9 \times 10^{-18}}$	$\mathbf{3.1 \times 10^{-18}}$
F26	$[-5, 5]$	2	0	1.9×10^{-17}	6.5×10^{-17}	4.9×10^{-18}	4.7×10^{-18}	7.5×10^{-18}	8.4×10^{-18}	$\mathbf{1.5 \times 10^{-18}}$	$\mathbf{1.4 \times 10^{-18}}$
F27	$[-5, 5]$	2	0	1.5×10^{-7}	3.5×10^{-7}	3.3×10^{-6}	1.3×10^{-5}	3.7×10^{-11}	2.0×10^{-10}	$\mathbf{3.1 \times 10^{-18}}$	$\mathbf{2.9 \times 10^{-18}}$
F28	$[-1, 1]$	2	0	1.1×10^{-2}	2.6×10^{-2}	2.7×10^{-2}	4.1×10^{-2}	$\mathbf{1.1 \times 10^{-3}}$	$\mathbf{1.9 \times 10^{-3}}$	5.9×10^{-3}	1.6×10^{-2}
F29	$[-10, 10]$	2	0	1.8×10^{-3}	3.8×10^{-3}	1.5×10^{-4}	2.7×10^{-4}	3.2×10^{-5}	4.7×10^{-5}	$\mathbf{7.5 \times 10^{-6}}$	$\mathbf{1.1 \times 10^{-5}}$
F30	$[-10, 10]$	2	-0.18465	-0.152569	0.027513	-0.175058	0.0269035	-0.148548	0.0326784	$\mathbf{-0.186899}$	0.0396051
F31	$[-10, 10]$	2	0	8.8×10^{-14}	2.4×10^{-13}	8.6×10^{-11}	1.8×10^{-10}	1.7×10^{-17}	2.1×10^{-17}	$\mathbf{1.7 \times 10^{-18}}$	$\mathbf{1.7 \times 10^{-18}}$
F32	$[-10, 10]$	2	0.9	0.90009	4.5×10^{-4}	0.9	5.6×10^{-7}	**0.9**	$\mathbf{4.4 \times 10^{-16}}$	0.906667	2.5×10^{-2}
F33	$[-100, 100]$	2	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
F34	$[-100, 100]$	2	0	5.6×10^{-14}	2.3×10^{-13}	7.4×10^{-18}	2.8×10^{-17}	**0**	**0**	**0**	**0**
F35	$[-2, 2]$	2	0	2.1×10^{-18}	2.8×10^{-18}	4.9×10^{-19}	8.1×10^{-19}	5.1×10^{-19}	6.2×10^{-19}	$\mathbf{2.5 \times 10^{-20}}$	$\mathbf{4.7 \times 10^{-20}}$
F36	$[-5, 15]$	2	0.397887	0.397887	6.0×10^{-7}	0.397887	3.40×10^{-7}	**0.397887**	$\mathbf{1.67 \times 10^{-16}}$	**0.397887**	$\mathbf{1.67 \times 10^{-16}}$
F37	$[-10, 10]$	2	-1	$\mathbf{-1}$	**0**	$\mathbf{-1}$	**0**	$\mathbf{-1}$	**0**	$\mathbf{-1}$	**0**
F38	$[-5, 5]$	2	-1.03163	$\mathbf{-1.03163}$	**0**	$\mathbf{-1.03163}$	**0**	$\mathbf{-1.03163}$	**0**	$\mathbf{-1.03163}$	**0**
F39	$[-10, 10]$	2	0	2.7×10^{-2}	3.4×10^{-2}	5.5×10^{-3}	9.1×10^{-3}	1.7×10^{-3}	3.5×10^{-3}	$\mathbf{1.2 \times 10^{-3}}$	$\mathbf{2.4 \times 10^{-3}}$
F40	$[0, 1]$	3	-3.86278	$\mathbf{-3.86278}$	$\mathbf{4.4 \times 10^{-16}}$	$\mathbf{-3.86278}$	$\mathbf{4.4 \times 10^{-16}}$	$\mathbf{-3.86278}$	$\mathbf{4.4 \times 10^{-16}}$	$\mathbf{-3.86278}$	$\mathbf{4.4 \times 10^{-16}}$
F41	$[0, 1]$	6	-3.32237	$\mathbf{-3.32232}$	$\mathbf{7.0 \times 10^{-5}}$	-3.32231	1.1×10^{-4}	-3.3031	4.3×10^{-2}	-3.29058	5.3×10^{-2}
F42	$[-10, 10]$	4	0	2.41054	2.5185	7.1×10^{-1}	6.8×10^{-1}	8.4×10^{-1}	8.1×10^{-1}	$\mathbf{4.1 \times 10^{-1}}$	$\mathbf{5.3 \times 10^{-1}}$
F43	$[-10, 10]$	4	0	1.1×10^{-3}	3.9×10^{-3}	2.3×10^{-5}	5.8×10^{-5}	7.9×10^{-4}	4.1×10^{-3}	$\mathbf{1.8 \times 10^{-7}}$	$\mathbf{4.6 \times 10^{-7}}$

The fitness values obtained on 43 test functions showed that the performance of the HABCES algorithm was the best. However, solution quality is not the only criterion. The convergence graphs of the algorithms also give an idea of their performance. In Figures 1 and 2, the convergence of the ABC, aABC, ABCES and the HABCES algorithms is compared for the first 20 test functions. For F1, F3, F4, F7, F9, F14, F16, F17, F18 and F19, the HABCES algorithm showed a fast convergence and reached the effective solution in a short time. For F2, all algorithms approached the optimal value in a short time. For F5, F6, F8, F10, F11, F12 and F20, the initial convergence speeds of the algorithms are similar. With increasing number of generations, the convergence speed of the HABCES algorithm became more effective than the others. For F13, the convergence speeds of the algorithms are close to one another. Due to the structure of F15, the convergence graphs of the algorithms showed a different appearance. The convergence graphics for the functions between F21 and F32 are compared in Figure 3. The HABCES algorithm converged rapidly for F22, F23, F24, F27, F29, F30 and F31. For F21, the convergence speeds of the algorithms are close to one another. For F25 and F26, the HABCES algorithm converged rapidly in a short time. Then, the speed of convergence decreased. For F28, the convergence speed of the HABCES algorithm was fast at first, and then decreased. The convergence graphics for the functions between F33 and F43 are compared in Figure 4. The HABCES algorithm showed fast convergence for F33, F34 and F35. In general, the convergence speeds were close for F36, F37, F38, F39 and F40. For F41, the HABCES algorithm reached a good solution in a short time. For F42, the convergence speed of the HABCES algorithm was better than for the others. For F43, it was seen that the convergence of the ABCES and HABCES algorithms was more effective. When the convergence graphs and solution qualities were evaluated together, it was concluded that the performance of the HABCES algorithm was better than the others.

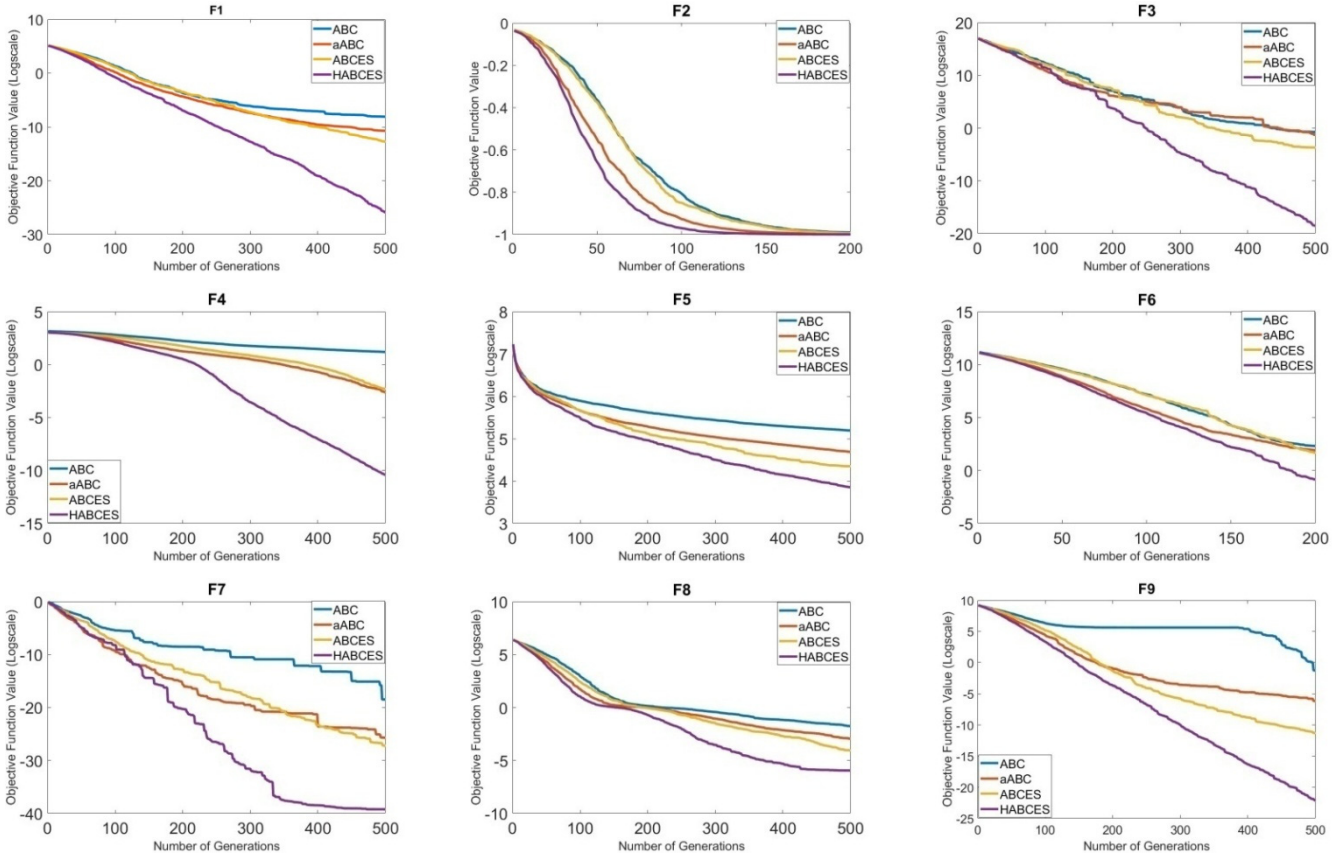


Figure 1. *Cont.*

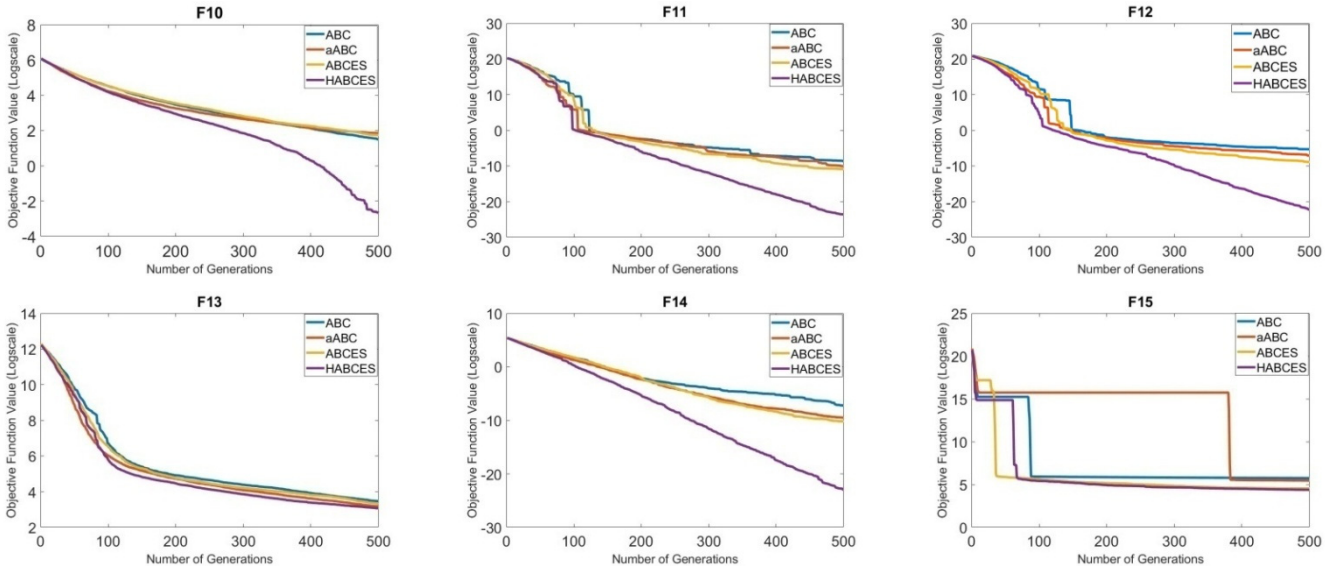


Figure 1. Comparison of the convergences of the ABC, aABC, ABCES and HABCES algorithms when solving high-dimensional global optimization problems (F1–F15).

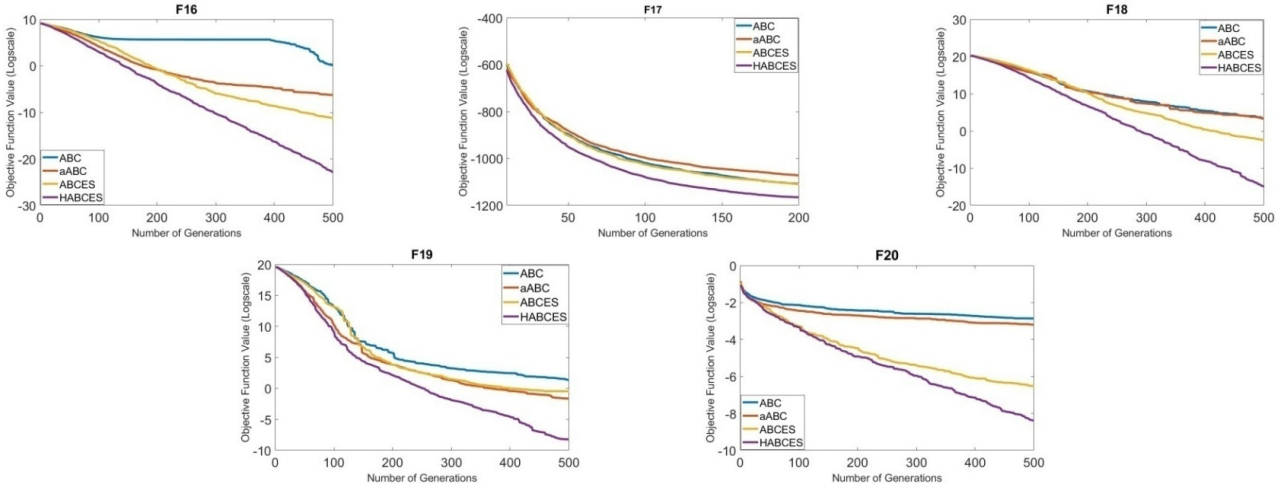


Figure 2. Comparison of the convergences of the ABC, aABC, ABCES and HABCES algorithms when solving high-dimensional global optimization problems (F16–F20).

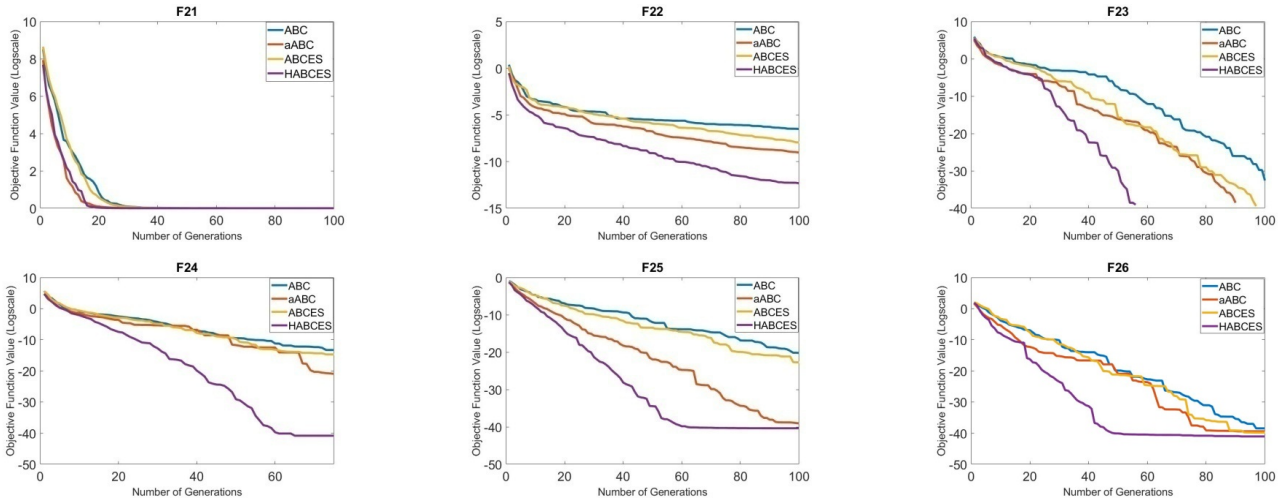


Figure 3. *Cont.*

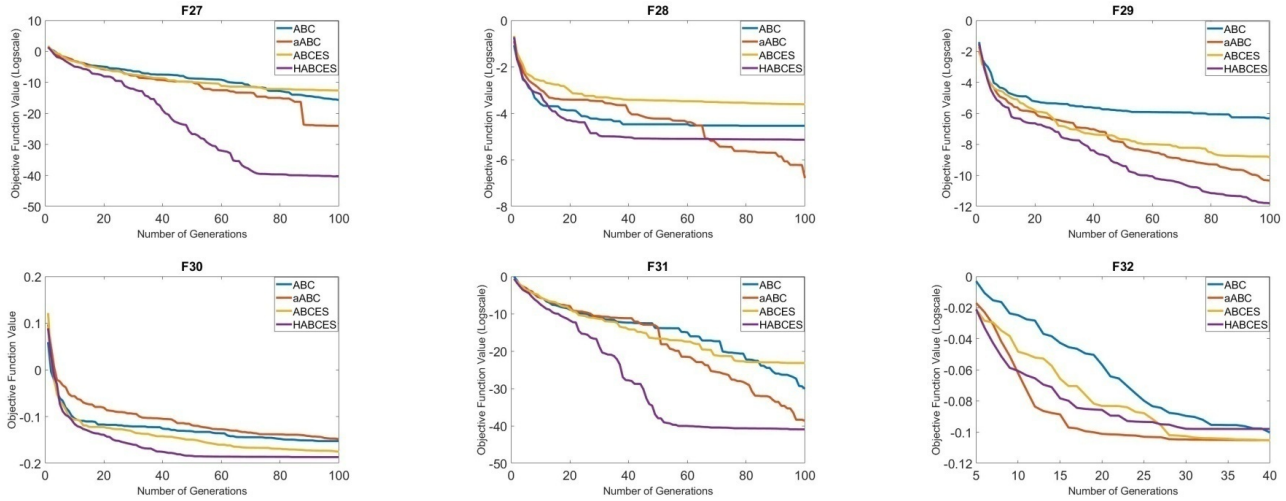


Figure 3. Comparison of the convergences of the ABC, aABC, ABCES and HABCES algorithms when solving low-dimensional global optimization problems (F21–F32).

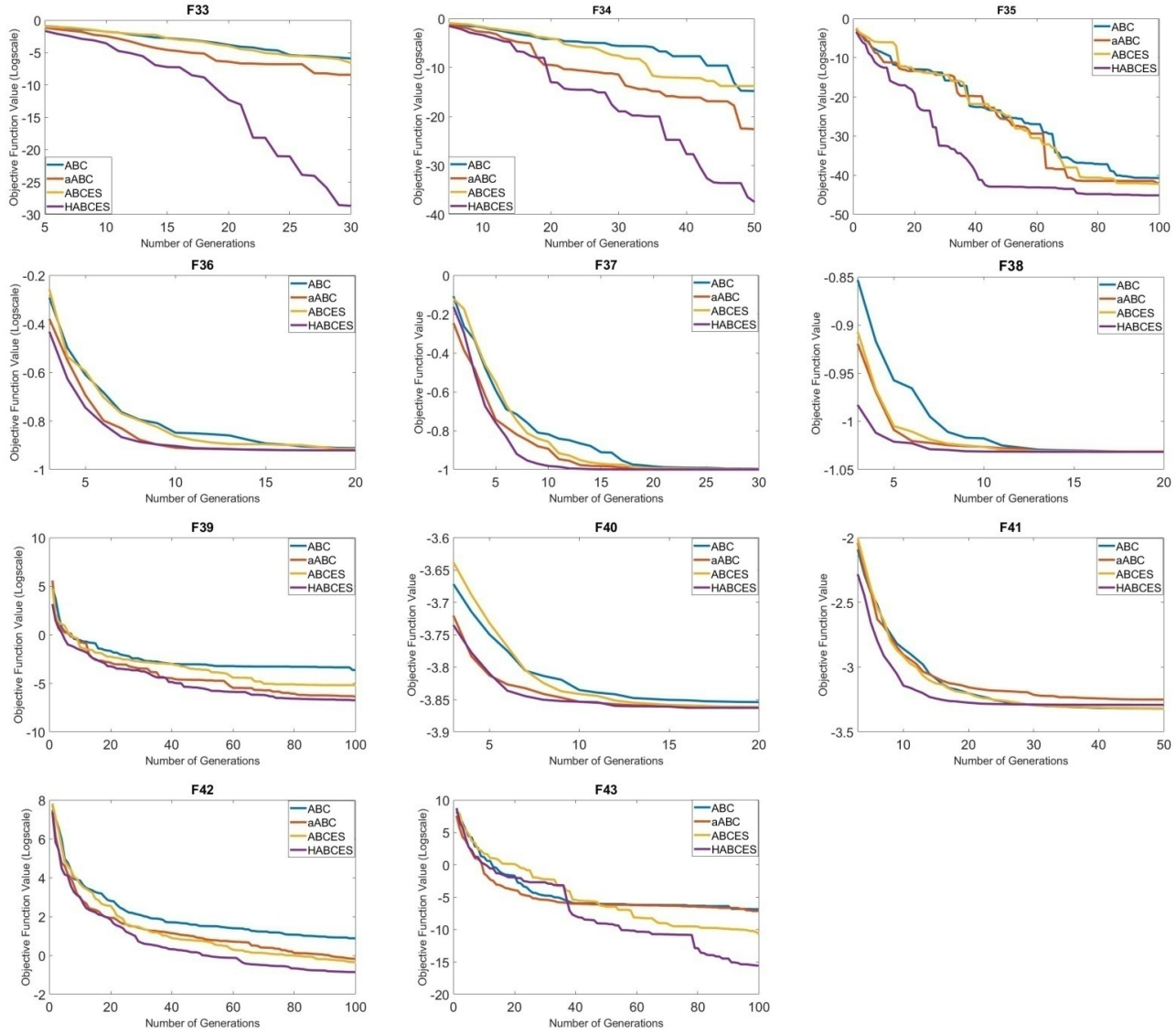


Figure 4. Comparison of the convergences of the ABC, aABC, ABCES and HABCES algorithms in solving low-dimensional global optimization problems (F33–F43).

3.2. Applications for the Training of the Neural Network

In this section, the performance of the HABCES algorithm regarding neural network training was evaluated. In this context, the algorithm was applied to three different groups of problems. These were the modeling of parity problems, the identification of nonlinear static systems, and the identification of nonlinear dynamic systems. The mean square error (MSE) was used as the error metric. Each application was run 30 times, and the mean error was calculated. The summing function was used as the transfer function. The sigmoid function was chosen as the activation function. The maximum number of generations was considered as the stopping criterion.

The results obtained when using the ABC, aABC, ABCES and the proposed HABCES algorithms for modeling of XOR, 3-bit parity and 4-bit parity problems are compared in Table 4. For each problem, 5 and 10 neurons were used in the hidden layer. For the XOR problem, the number of generations was taken to be 50. In the others, it was 100. A colony size of 20 was adopted. In the XOR problem, increasing numbers of neurons decreased the error value. The error values obtained when using the HABCES algorithm for 2-5-1 and 2-10-1 network structures were 0.00406869 and 0.000516968, respectively. After the HABCES algorithm, the best results were found with the ABCES algorithm. As in the XOR problem, the increase in the number of neurons improved the quality of the solution in the 3-bit parity problem. The best result obtained was 0.00855215, which was achieved when using the HABCES algorithm with the 3-10-1 network structure. The 4-bit parity problem was more difficult than the other problems. This situation can also be observed from the error values found. The error values found for the 4-5-1 and 4-10-1 network structures when using the HABCES algorithm were 0.118425 and 0.0913625, respectively. The performance of the other algorithms was worse. Table 4 shows that the solution quality of the HABCES algorithm was better for all problem and network structures. Figure 5 provides a comparison of the convergences of the ABC, aABC, ABCES and HABCES algorithms when solving parity problems. The fastest and most efficient convergence for each problem was achieved using the HABCES algorithm.

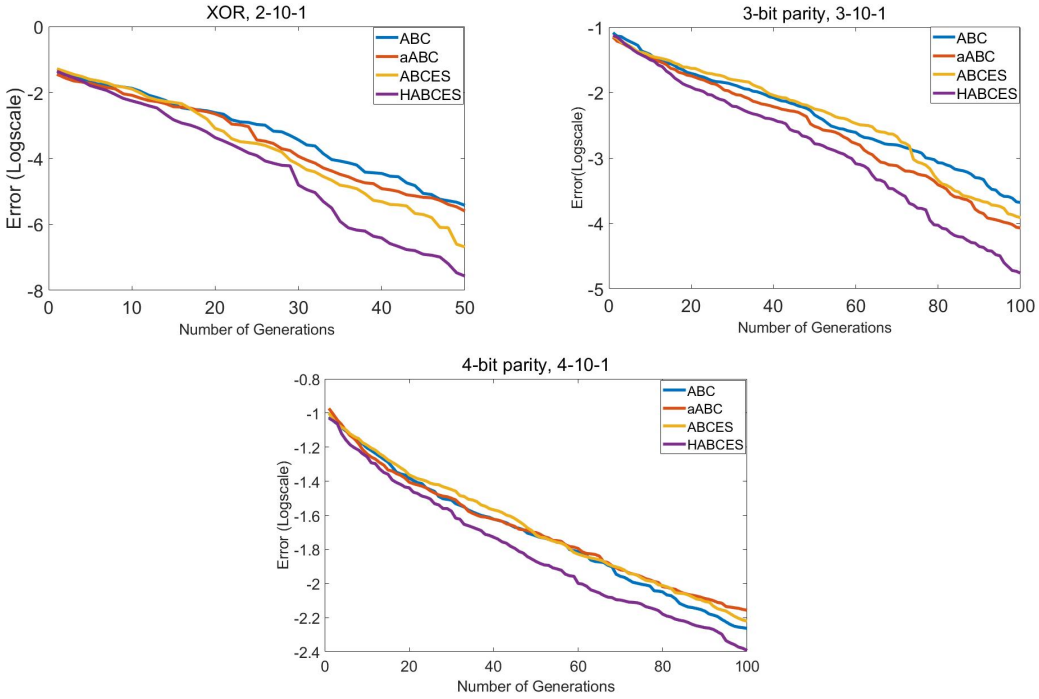


Figure 5. Comparison of the convergences of the ABC, aABC, ABCES and HABCES algorithms when solving parity problems.

Table 4. Comparison of the results of the ABC, aABC, ABCES and HABCES algorithms when solving parity problems.

Examples	Number of Generations	Network Structure	ABC		ABCES		aABC		HABCES (Proposed)	
			Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
XOR	50	2-5-1	0.0270955	0.0653951	0.0134276	0.024631	0.0106609	0.0167402	**0.00406869**	**0.00643696**
		2-10-1	0.00440856	0.00875478	0.00370912	0.0070464	0.00124831	0.00196459	**0.00051696**	**0.00079469**
3 bit parity	100	3-5-1	0.0370702	0.0325831	0.0413162	0.0306131	0.031667	0.0318723	**0.023972**	**0.0280218**
		3-10-1	0.0252005	0.0380509	0.0170671	0.0263739	0.0200067	0.0234835	**0.00855215**	**0.00924428**
4 bit parity	100	4-5-1	0.128162	0.0415436	0.135707	0.0390414	0.128552	0.0290362	**0.118425**	**0.0281676**
		4-10-1	0.104024	0.0425457	0.115735	0.0388215	0.10854	0.0391539	**0.0913625**	**0.0297292**

Secondly, the performance of the HABCES algorithm was analyzed with respect to the identification of nonlinear static systems. The information regarding the nonlinear static systems used is given in Table 5. S1 has one input. S2 and S3 have two inputs. S4 and S5 have three inputs. S6 has four inputs. All systems have one output. Neural networks with 4, 8 and 12 neurons in the hidden layer were used for each system. Colony size and the maximum number of generations belonging to the training algorithms were taken as 20 and 1000, respectively. In general, 80% of the data were used for the training process. The rest was reserved for the testing process. The results obtained when using the HABCES algorithms for nonlinear static system identification are given in Table 6. It can be seen that both the training and test results for S1 improve in parallel with increasing numbers of neurons. The best results were found when using the 1-12-1 network structure. The best training and test error values were 0.00072 and 0.00161, respectively. The effect of the number of neurons on performance was the same for S2 as it was for S1. The best training error value found was 0.00058. Additionally, the best test error value obtained was 0.00453. Increasing the number of neurons in S3 worsened the solution quality. The best error values for the training and testing process were found when using the 2-4-1 network structure. The error values were 0.00014 and 0.00297, respectively. In S4, the 3-8-1 network structure was more effective than the others. The training error value of the 3-8-1 network structure was 0.00154. The test result was also 0.00207. The best results for S5 were obtained with a network including four neurons in the hidden layer, similarly to the case of S3. The training and test error values for S5 were 0.00082 and 0.00463, respectively. Reducing the number of neurons in S6 improved performance. Effective results were achieved with the 4-4-1 network structure. The best training error value for S6 was 0.00148. Additionally, the test error value was 0.001563. Table 7 compares the results obtained when using the ABC, aABC, ABCES and HABCES algorithms for nonlinear static system identification. For all systems, the best training error value was found when using the HABCES algorithm. Other than for S1 and S6, the best test results were obtained when using the HABCES algorithm. The best test results for S1 and S6 were obtained by using the aABC algorithm. Table 8 shows the increase in performance achieved when using the HABCES algorithm. The HABCES algorithm provided a performance improvement of up to 69.57% compared to using the ABC algorithm for the training process. For the test process, this rate was 35.29%. With respect to the aABC algorithm, performance improvements of up to 54.84% (training) and 23.05% (test), respectively, were achieved. When comparing the HABCES algorithm with the ABCES algorithm, effective performance increases can be observed. A performance increase of up to 32.56% was achieved for training. For the test process, this rate was 54.11%. Figure 6 presents a comparison of the convergences of the ABC, aABC, ABCES and HABCES algorithms. For all systems, the HABCES algorithm obtained effective results by achieving fast convergence. In particular, the convergence speeds for S1, S2, S3, S5 and S6 were much better than those of the other algorithms. In some iterations of S4, the ABCES and the HABCES algorithms came close to one another. A comparison of the real output and the predicted output for all nonlinear static systems is given in Figure 7. It can be

seen that the real and predicted outputs overlap with each other for all systems. This is an indication that the HABCES algorithm can successfully solve the corresponding problems.

Table 5. Information on the nonlinear static systems used.

System	Equation	Inputs	Output	Number of Training/Test Data	Range
S_1	$y = 2\sin(\pi x_1)$	x_1	y	80/20	[0, 1]
S_2	$y = 10.391\{(x_1 - 0.4)(x_2 - 0.6) + 0.36\}$	x_1, x_2	y	80/20	[0, 1]
S_3	$y = \tanh(x_1 + x_2 - 11)$	x_1, x_2	y	80/20	[0, 1]
S_4	$y = 1 + {x_1}^{0.5} + {x_2}^{-1} + {x_3}^{-1.5}$	x_1, x_2, x_3	y	173/43	[1, 6]
S_5	$y = (x_1 - 5.5)^2 + (x_2 - 5.5)^2 + {x_3}^2$	x_1, x_2, x_3	y	100/25	[0, 1]
S_6	$y = e^{2x_1 \sin(\pi x_4)} + \sin(x_2 x_3)$	x_1, x_2, x_3, x_4	y	100/25	[0.25, 0.25]

Table 6. The results obtained when using the HABCES algorithm for nonlinear static system identification.

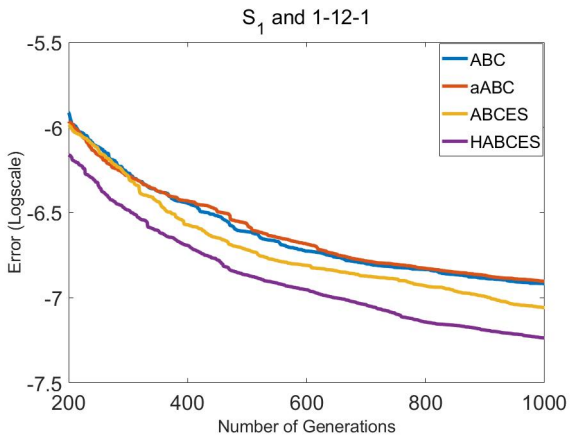
System	Network Structure	Train		Test	
		Mean	Sd.	Mean	Sd.
S_1	1-4-1	0.00150	0.00061	0.00258	0.00086
	1-8-1	0.00109	0.00050	0.00201	0.00086
	1-12-1	**0.00072**	**0.00031**	**0.00161**	**0.00068**
S_2	2-4-1	0.00128	0.00063	0.00691	0.00312
	2-8-1	0.00073	0.00063	0.00521	0.00494
	2-12-1	**0.00058**	**0.00027**	**0.00453**	**0.00233**
S_3	**2-4-1**	**0.00014**	**0.00007**	**0.00297**	**0.00173**
	2-8-1	0.00016	0.00011	0.00332	0.00213
	2-12-1	0.00025	0.00012	0.00404	0.00191
S_4	3-4-1	0.00214	0.00073	0.00253	**0.00075**
	3-8-1	**0.00154**	**0.00068**	**0.00207**	0.00078
	3-12-1	0.00174	0.00085	0.00212	0.00109
S_5	**3-4-1**	**0.00082**	**0.00050**	**0.00463**	**0.00391**
	3-8-1	0.00085	0.00053	0.00893	0.00996
	3-12-1	0.00129	0.00085	0.01145	0.00980
S_6	**4-4-1**	**0.00148**	0.00053	**0.001563**	0.00086
	4-8-1	0.00157	**0.00044**	0.00186	**0.00071**
	4-12-1	0.00164	0.00056	0.00229	0.00081

Table 7. Comparison of the results of the ABC, aABC, ABCES and HABCES algorithms for nonlinear static system identification.

Examples	Network Structure	ABC		ABCES		aABC		HABCES (Proposed)	
		Train	Test	Train	Test	Train	Test	Train	Test
S_1	1-12-1	0.00099	0.00169	0.00100	0.00187	0.00086	**0.00153**	**0.00072**	0.00161
S_2	2-12-1	0.00110	0.00590	0.00088	0.00585	0.00086	0.00480	**0.00058**	**0.00453**
S_3	2-4-1	0.00046	0.00459	0.00031	0.00386	0.00019	0.00354	**0.00014**	**0.00297**
S_4	3-8-1	0.00235	0.00278	0.00180	0.00220	0.00170	0.00211	**0.00154**	**0.00207**
S_5	3-4-1	0.00183	0.00709	0.00128	0.00570	0.00118	0.01009	**0.00082**	**0.00463**
S_6	4-4-1	0.00208	0.00227	0.00161	0.00167	0.00152	**0.00152**	**0.00148**	0.00156

Table 8. Performance increase achieved when using HABCES for nonlinear static system identification.

System	ABC-HABCES		aABC-HABCES		ABCES-HABCES	
	Train (%)	Test (%)	Train (%)	Test (%)	Train (%)	Test (%)
S_1	27.27	4.73	28.00	13.90	16.27	-
S_2	47.27	23.22	34.09	22.56	32.56	5.62
S_3	69.57	35.29	54.84	23.05	26.31	16.10
S_4	34.47	25.54	14.44	5.91	9.41	1.90
S_5	55.19	34.70	35.94	18.77	30.51	54.11
S_6	28.85	31.27	8.07	6.58	2.63	-



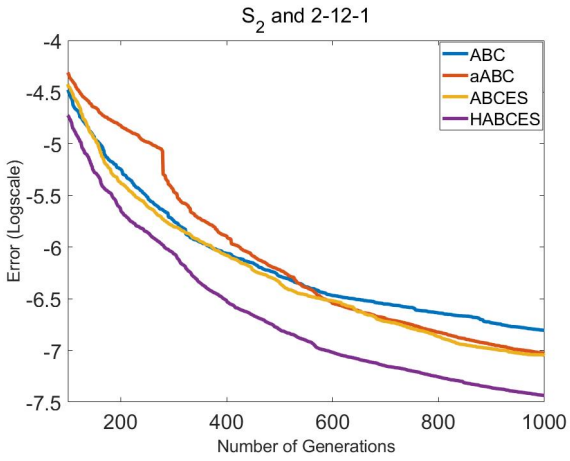


Figure 6. *Cont.*

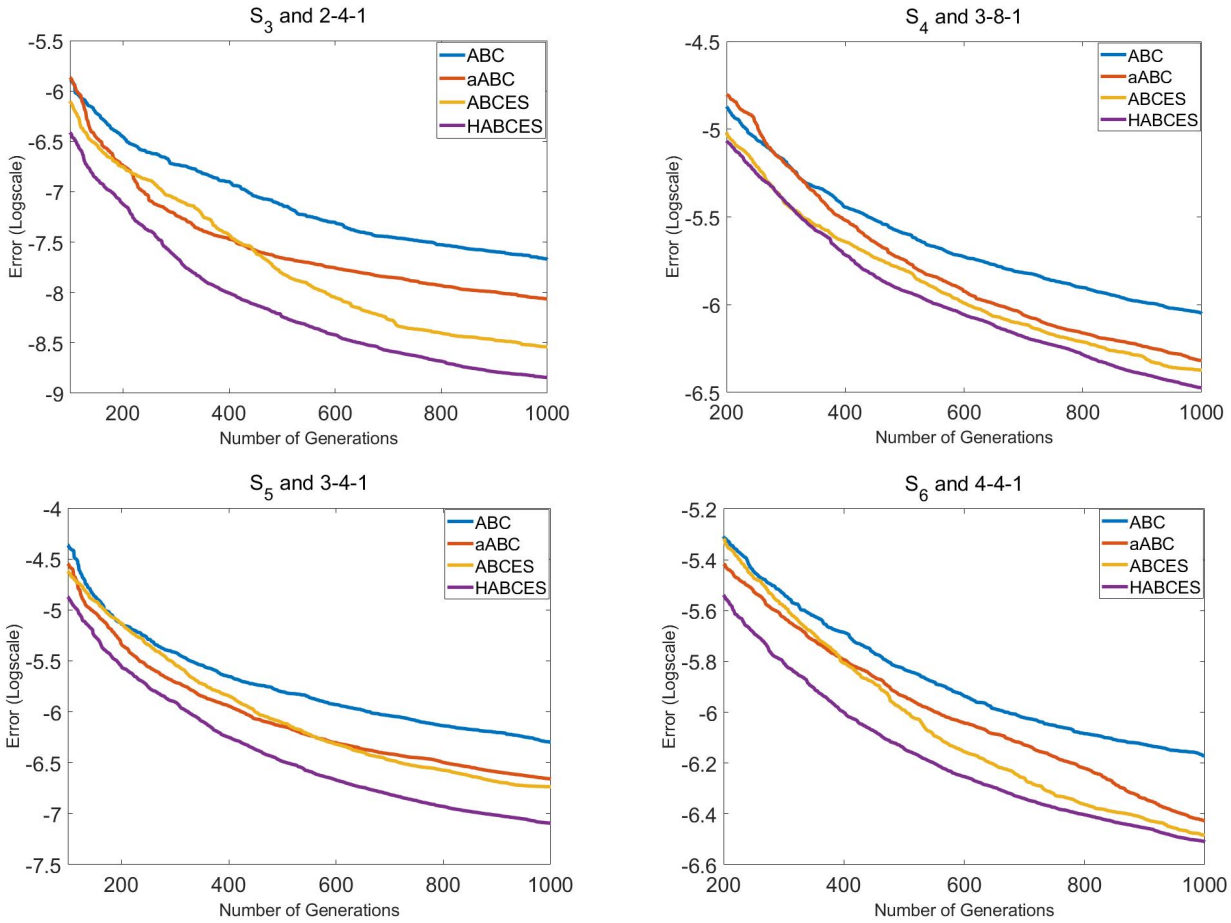


Figure 6. Comparison of the convergences of the ABC, aABC, ABCES and HABCES algorithms for nonlinear static system identification.

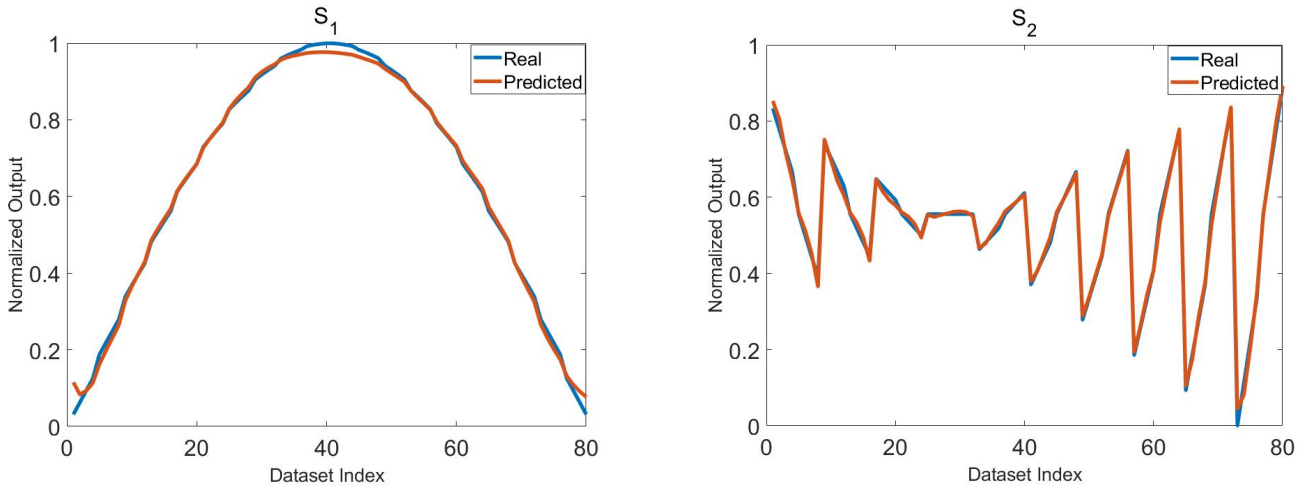


Figure 7. *Cont*.

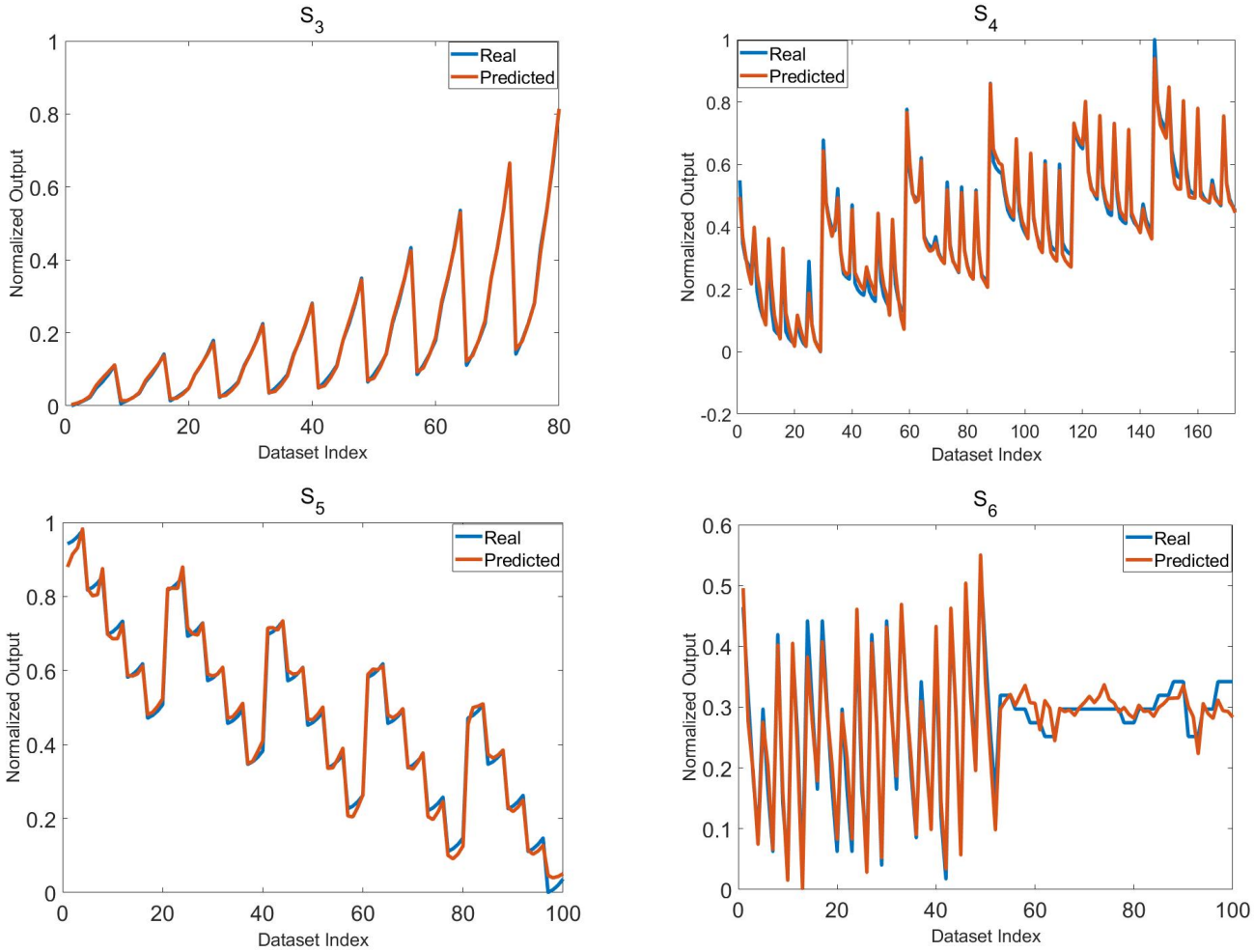


Figure 7. Comparison of the real output and the predicted output for nonlinear static system identification.

Thirdly, the algorithms were applied for the identification of nonlinear dynamic systems. Eight nonlinear dynamic systems were used. The nonlinear dynamic systems utilized are presented in Table 9. D3, D5 and D8 consist of two inputs. D1, D2, D4 and D7 have three inputs. D6 consists of four inputs. Except for D6, 80% of the data were used for training. The rest were designated for the testing process. For D6, these rates were 90% and 10%, respectively. Network structures with 4, 8 and 12 neurons in the hidden layer were utilized. The colony size and maximum number of generations used were 20 and 1000, respectively, as in the identification of nonlinear static systems. The results obtained using the HABCES algorithms in nonlinear dynamic system identification are given in Table 10. The best training and test error values for D1 were obtained using the 3-8-1 network structure. The related error values were 0.00039 and 0.00072, respectively. Increasing the number of neurons in D2 decreased the error value. The best results were obtained when using the 3-12-1 network structure. The training error value for the 3-12-1 network structure was 0.00066. The test error value was 0.00082. The most effective results for D3 and D4 were obtained when using network structures featuring eight neurons in the hidden layer. The training error values of D3 and D4 were 0.00057 and 0.00348, respectively. In addition, their test error values were 0.00070 and 0.00367, respectively. For D5, effective results were generally found when using all network structures. The 2-8-1 network structure was slightly more successful than the others. Increasing the number of neurons in D6 decreased performance. Namely, the results got worse. The best training and testing results were obtained when using the 4-4-1 network structure. The best training error value of D6 was 0.00067. The test error value was also 0.00105. The 3-8-1 network structure was effective for D7. There was a clear difference between the error values obtained for 3-8-1

and those obtained for the other network structures. The best training and test error values for D7 were 0.00030 and 0.00050, respectively. According to the training and test results in D8, the 2-8-1 network structure is more effective. The best training error value obtaind for D8 was 0.00017. The test error value was 0.00031. The performance of the HABCES algorithm was compared with the ABC, aABC and ABCES algorithms, and the results are presented in Table 11. It can be seen that the HABCES algorithm is better than the others with respect to the values of training and test error for all systems. A serious increase in performance was achieved when using the HABCES algorithm. The performance increase rates according to the systems are given in Table 12. With the HABCES algorithm, a performance increase of up to 46.82% was achieved in the training process compared to when using the ABC algorithm. This ratio was 39.53% for the test. Compared to the aABC algorithm, the HABCES algorithm achieved performance improvements of up to 37.96% and 35.00%, respectively, for the training and testing processes. At the same time, the HABCES algorithm was more successful than the ABCES algorithm. With respect to the ABCES algorithm, a 20.48% performance increase was achieved in the training process. For the testing process, this ratio was 25.77%. Figure 8 presents a comparison of the convergences of the ABC, aABC, ABCES and HABCES algorithms. In D1, D2, D4, D6, D7 and D8, the HABCES algorithm showed a rapid convergence and reached effective solutions in a short time. In D3 and D5, the HABCES and ABCES algorithms had good convergence. A comparison of the real output and the predicted output for nonlinear dynamic system identification is given in Figure 9. It can be understood from the overlapping output graphics that successful modeling for all systems was achieved when using the HABCES algorithm.

Table 9. Nonlinear dynamic systems used.

System	Equation	Inputs	Output	Number of Training/Test Data
D_1	$y(k+1) = \frac{y(k)[y(k-1)+2][y(k)+2.5]}{8.5+[y(k)]^2+[y(k-1)]^2} + u(k)u(k) =$ $\begin{cases} 2\cos(2\pi k/100), & k \le 200 \\ 1.2\sin(2\pi k/20), & 200 < k \le 500 \end{cases}$	$y(k), y(k-1), u(k)$	$y(k+1)$	400/100
D_2	$y(k+1) = \frac{y(k)y(k-1)[y(k)+2.5]}{1+[y(k)]^2+[y(k-1)]^2} + u(k)u(k) = \sin(2\pi k/25)$	$y(k), y(k-1), u(k)$	$y(k+1)$	200/50
D_3	$y(k+1) = \frac{y(k)}{1+y(k)^2} + u(k)^3u(k) = \sin(2\pi k/100)$	$y(k), u(k)$	$y(k+1)$	200/50
D_4	$y(k+1) = \frac{y(k)}{1+y(k-1)} + u(k)^3u(k) = \sin(\pi k/25)$	$y(k), y(k-1), u(k)$	$y(k+1)$	200/50
D_5	$y(k+1) = (1+y(k)^2)^{-1}(-0.9y(k)+u(k))u(k) = 0.75\sin(\pi(k-1)/180)$	$y(k), u(k)$	$y(k+1)$	400/100
D_6	$y(k+1) = 0.72y(k) + 0.025y(k-1)u(k) + 0.01u(k-1)^2 + 0.2u(k-2)u(k) =$ $\begin{cases} \sin(\pi k/25), & k < 250 \\ 1.0, & 250 \le k < 500 \\ -1.0, & 500 \le k < 750 \\ 0.3\sin(\pi k/25) + 0.1\sin(\pi k/32) + 0.6\sin(\pi k/10), & 750 \le k \le 1000 \end{cases}$	$y(k), y(k-1), u(k), u(k-1),$	$y(k+1)$	900/100
D_7	$y(k+1) = 0.3y(k) + 0.6y(k-1) + N(u(k))N(u) =$ $0.6\sin(\pi u) + 0.3\sin(3\pi u) + 0.1\sin(5\pi u)u(k) = \sin\frac{2\pi k}{250}$	$y(k), y(k-1), u(k)$	$y(k+1)$	400/100
D_8	$y(k+1) = \frac{y(k)y(k-1)y(k-2)u(k-1)[y(k-2)-1]+u(k)}{1+y(k-1)^2+y(k-2)^2}u(k) =$ $\begin{cases} \sin(2\pi k/250), & k \le 500 \\ 0.8\sin(2\pi k/250) + 0.2\sin(2\pi k/25), & k > 500 \end{cases}$	$y(k), u(k)$	$y(k+1)$	800/200

Table 10. The results obtained by using the HABCES algorithms in nonlinear dynamic system identification.

System	Network Structure	Train		Test	
		Mean	Sd.	Mean	Sd.
D_1	3-4-1	0.00051	0.00024	0.00088	0.00032
	3-8-1	**0.00039**	**0.00015**	**0.00072**	**0.00030**
	3-12-1	0.00052	0.00022	0.00106	0.00075
D_2	3-4-1	0.00084	0.00026	0.00093	**0.00031**
	3-8-1	0.00075	0.00024	0.00096	0.00053
	3-12-1	**0.00066**	**0.00023**	**0.00082**	0.00041
D_3	2-4-1	0.00070	0.00021	0.00095	0.00046
	2-8-1	**0.00057**	**0.00009**	**0.00070**	**0.00022**
	2-12-1	0.00067	0.00017	0.00085	0.00039
D_4	3-4-1	0.00426	0.00078	0.00440	0.00081
	3-8-1	**0.00348**	**0.00038**	**0.00367**	**0.00048**
	3-12-1	0.00354	0.00040	0.00367	0.00052
D_5	2-4-1	0.00026	0.00012	0.00026	0.00014
	2-8-1	**0.00024**	0.00013	**0.00024**	**0.00011**
	2-12-1	0.00026	**0.00011**	0.00030	0.00022
D_6	**4-4-1**	**0.00067**	0.00031	**0.00105**	**0.00045**
	4-8-1	0.00079	**0.00029**	0.00133	0.00048
	4-12-1	0.00086	0.00042	0.00121	0.00063
D_7	3-4-1	0.00035	0.00017	0.00063	0.00031
	3-8-1	**0.00030**	**0.00014**	**0.00050**	**0.00022**
	3-12-1	0.00034	0.00020	0.00059	0.00031
D_8	2-4-1	0.00021	0.00009	0.00040	0.00011
	2-8-1	**0.00017**	**0.00006**	**0.00031**	**0.00008**
	2-12-1	0.00017	0.00006	0.00035	0.00011

Table 11. Comparison of the results of the ABC, aABC, ABCES and HABCES algorithms for nonlinear dynamic system identification.

Examples	Network Structure	ABC		ABCES		aABC		HABCES (Proposed)	
		Train	Test	Train	Test	Train	Test	Train	Test
D_1	3-8-1	0.00061	0.00095	0.00046	0.00087	0.00047	0.00097	**0.00039**	**0.00072**
D_2	3-12-1	0.00092	0.00105	0.00082	0.00090	0.00083	0.00101	**0.00066**	**0.00082**
D_3	2-8-1	0.00070	0.00090	0.00070	0.00098	0.00060	0.00077	**0.00057**	**0.00070**
D_4	3-8-1	0.00372	0.00389	0.00370	0.00384	0.00367	0.00375	**0.00348**	**0.00367**
D_5	2-4-1	0.00039	0.00043	0.00034	0.00040	0.00028	0.00029	**0.00026**	**0.00026**
D_6	4-4-1	0.00126	0.00143	0.00108	0.00160	0.00079	0.00110	**0.00067**	**0.00105**
D_7	3-8-1	0.00049	0.00067	0.00041	0.00064	0.00036	0.00056	**0.00030**	**0.00050**
D_8	2-8-1	0.00028	0.00045	0.00023	0.00039	0.00019	0.00037	**0.00017**	**0.00031**

Table 12. Performance increase achieved using HABCES for nonlinear dynamic system identification.

System	ABC-HABCES		aABC-HABCES		ABCES-HABCES	
	Train (%)	Test (%)	Train (%)	Test (%)	Train (%)	Test (%)
D_1	36.06	24.21	15.21	17.24	17.02	25.77
D_2	28.26	21.90	19.51	8.89	20.48	18.81
D_3	18.57	22.22	18.57	28.57	5.00	9.09
D_4	6.45	5.65	5.94	4.42	5.17	2.13
D_5	33.33	39.53	23.53	35.00	7.14	10.34
D_6	46.82	26.57	37.96	34.37	15.19	4.55
D_7	38.77	25.37	26.82	21.87	16.66	10.71
D_8	39.28	31.11	26.08	20.51	10.52	16.21

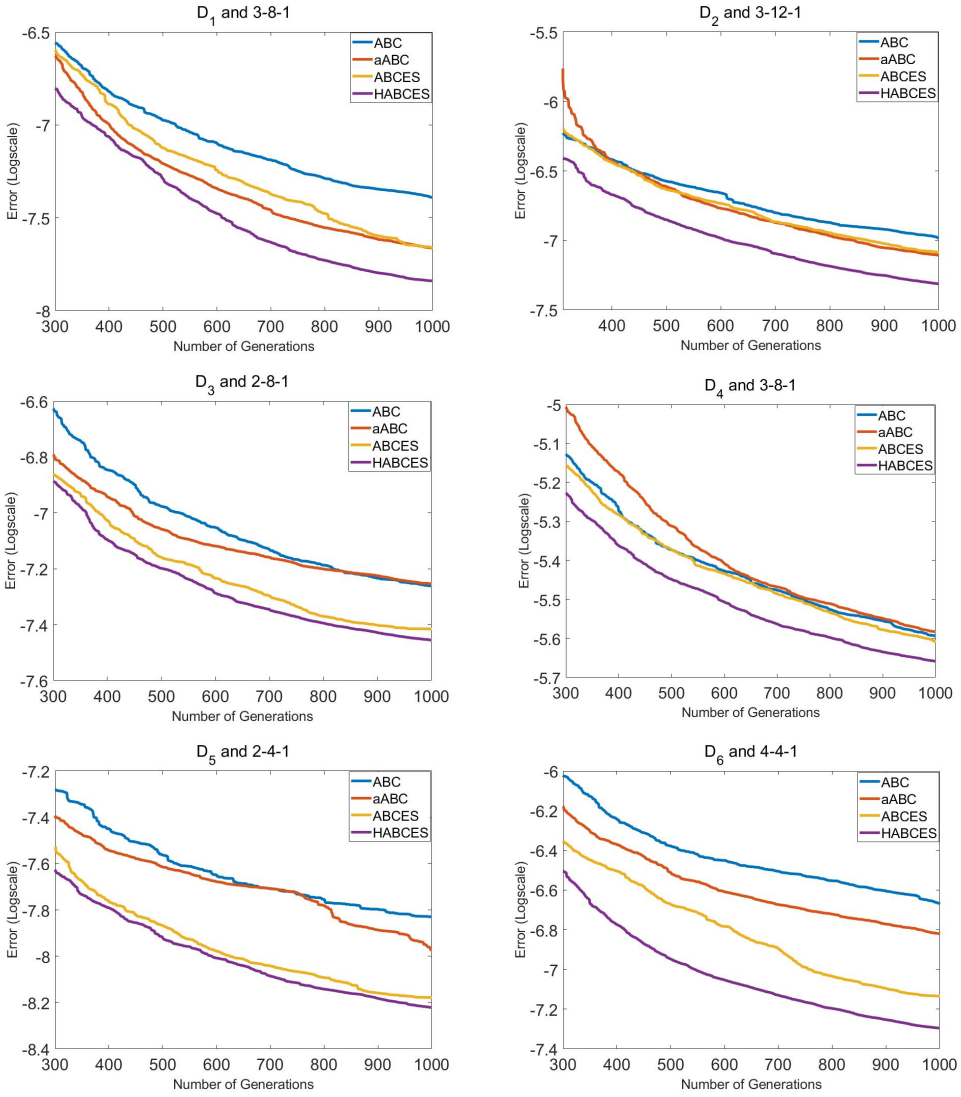


Figure 8. *Cont.*

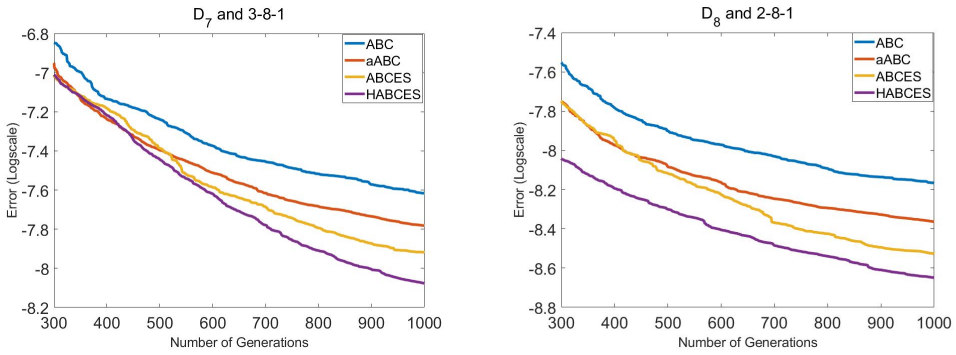


Figure 8. Comparison of the convergences of the ABC, aABC, ABCES and HABCES algorithms for nonlinear dynamic system identification.

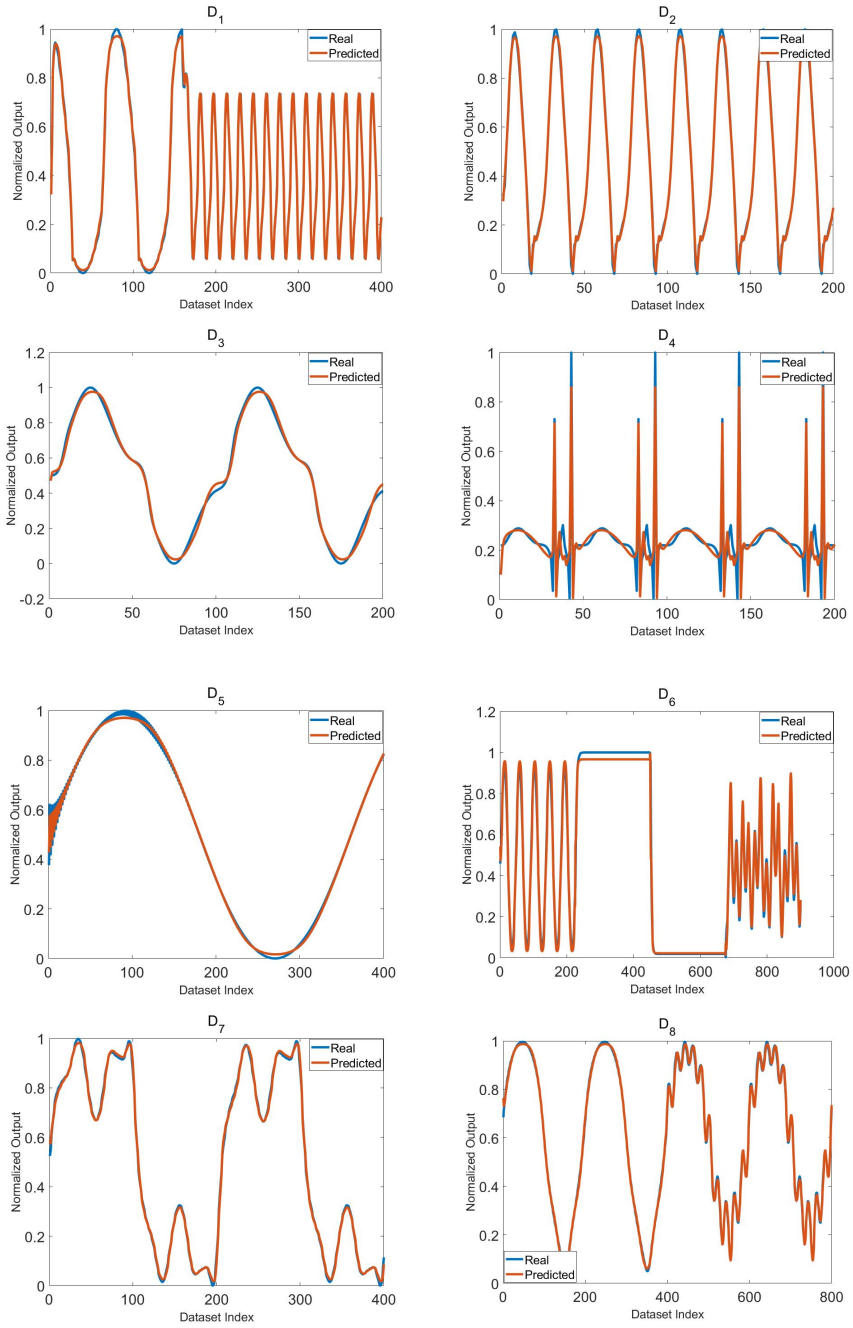


Figure 9. Comparison of the real output and the predicted output for nonlinear dynamic system identification.

4. Discussion

This study aims to develop a variant with strong global and local search capability for neural network training. The HABCES algorithm is a hybrid algorithm consisting of a combination of the aABC algorithm and ABCES algorithm. The strengths of both algorithms have been taken, making the HABCES algorithm a powerful algorithm with effective global and local convergence capability.

The arithmetic crossover feature was adapted for the HABCES algorithm, and knowledge of the global best solution was used. At the same time, the width of the search area was determined adaptively. This adaptive feature generally resulted in improved solution quality at each iteration. These two features provided a strong convergence feature. In particular, the determination of the solutions in the scout bee phase by means of an effective solution generation mechanism, rather than being randomly generated, ensured the preservation of the gains obtained. All of these changes strengthened the convergence ability of the HABCES algorithm and improved its solution quality. Although the HABCES algorithm was developed as a neural network training algorithm, it should be emphasized that it was also successful at solving global optimization problems. This fact expands the usage area of the algorithm. To interpret the performance of the HABCES algorithm with respect to different variants, it was compared with the standard ABC, aABC and ABCES algorithms. The results showed that the HABCES algorithm was more successful than the aABC and ABCES algorithms both at solving global optimization problems and at training the neural network.

The HABCES algorithm was first applied to solving global optimization problems. Forty-three benchmark test functions were used. These test functions were examined under two groups: high-dimensional and low-dimensional problems. High-dimensional problems can be more difficult or complex due to their higher number of dimensions. Due to the fast convergence of the HABCES algorithm, it was found that it was able to approach the optimal solution within a short period of time. The other variants fell short of the performance achieved by the HABCES algorithm. For low-dimensional optimization problems, it may be easier to reach the optimal result in a low number of iterations. Therefore, with increasing difficulty or size of a problem, the success of the HABCES algorithm could be more clearly observed.

In the training of neural networks, the algorithm was applied for the modeling of parity problems, the identification of nonlinear static systems, and the identification of nonlinear dynamic systems. The amount of data used in the training process for solving XOR, 3-bit parity, and 4-bit parity problems was limited. At the same time, the problem became more complex with increasing numbers of inputs. Parallel to this situation, the error value also increased. For these problems, there was a significant performance advantage compared to the other variants of the HABCES algorithm. In particular, the results were obtained with a low number of iterations. The fact that the HABCES algorithm was able to achieve an effective solution in a short time is an important indicator of its success.

Most of the problems encountered in daily life exhibit nonlinear behavior. Therefore, it is important to identify nonlinear systems. Nonlinear systems are inherently difficult problems. Effective training algorithms are required in order to obtain successful results. Nonlinear systems can be static or dynamic. In this study, the algorithm was applied for both system types. It can be observed that the HABCES algorithm was able to achieve effective results in a short period of time for the identification of nonlinear systems. When examining the output graphs obtained for these systems, the HABCES algorithm can be seen to have been successful. The graphs of real and predicted output overlap. The success of the HABCES algorithm at performing neural network training for the identification of these systems shows that it can also be used for solving different problems. The aABC algorithm stands out as a result of its arithmetic crossover and adaptive step size. The ABCES algorithm also stands out due to its adaptive limit value, as well as its solution generation mechanism at the scout bee stage. These features affect the result differently. In the identification of static systems, it can be seen that adaptive limit values and the effective

scout bee stage have a more significant positive effect on the results than the arithmetic crossover and adaptive step size. In the identification of dynamic systems, different system behaviors are observed depending on the type of system.

One of the important parameters is standard deviation. In all problem types, effective standard deviation values were obtained in parallel with the solution. Each application was repeated 30 times, and the mean error/fitness values were obtained. The effective standard deviation values show that the obtained results can be achieved again within a certain tolerance. In particular, meta-heuristic algorithms usually start with random solutions. The closeness of the obtained results to one another also increases the confidence in the algorithm.

It can be observed that the network structures created in these applications affect the performance. In particular, an increase in the number of neurons resulted in an increase in performance in some cases, while decreasing the quality of the solution in other cases. This is related to the difficulty of the problems.

This study has limitations in terms of population size, maximum number of generations, network structures, and problem types. A value of 20, one of the values for which the ABC algorithm is effective, was used as the colony size. For the maximum number of generations, different values were selected depending on the problem type. In neural network training, it is known that the network structure affects performance. In this study, the applications were realized using a limited number of network structures for each problem. Two different types of problem were used. These were the solving of global optimization problems and the training of a neural network. The success of the HABCES algorithm should be evaluated within the context of these limitations.

5. Conclusions

This study proposed a new variant referred to as the HABCES algorithm for neural network training. Effective solution generation mechanisms were used in the employed bee, onlooker bee, and scout bee stages of the HABCES algorithm. The algorithm basically incorporates four changes. The first is the usage of arithmetic crossover between the current solution and the global best solution. Secondly, an adaptive approach was used in the selection of the local search area. Specifically, it changes depending on the number of generations. This can be thought of as a dynamic step size. Third, the limit control parameter was dynamically adjusted depending on the generation number. Fourth, in the scout bee stage, new solutions were created using an effective solution generation mechanism instead of randomly generating them.

Unlike the standard ABC algorithm, the HABCES algorithm was able to reach effective solutions in a short time as a result of these four fundamental innovations. In particular, as the dimensions and difficulty of the problem increased, the success of the HABCES algorithm could be more clearly observed. This was particularly evident with the solution of global optimization problems. At the same time, the HABCES algorithm was generally more successful than the standard ABC, ABC and ABCES algorithms for solving global optimization problems.

To create effective models with ANN, a successful training algorithm is required. Therefore, this study focused on developing a more successful algorithm for ANN training. In ANN training, there are different parameters that affect performance in addition to the training algorithm. Number of inputs, number of layers, number of neurons, activation function, and transfer functions are among them. This was observed in application. The structure of the neural network affected the performance. This effect varied depending on the type of problem. In some problems, increasing the number of neurons improved the solution quality. In others, the opposite situation was observed. Within its limitations, the HABCES algorithm seemed to be effective on the basis of convergence graphs, output graphs, and simulation results.

Nonlinear systems are inherently complex and difficult. Effective results were obtained when using the HABCES algorithm for the identification of the related systems. In

particular, it had a significant advantage over variants such as aABC and ABCES. In nonlinear static systems, although it varies according to the system, the training performance increased by up to 54.84% with reference to the aABC algorithm. In comparison with the ABCES algorithm, this improvement reached 32.56%. Similar success was achieved for nonlinear dynamic systems. For nonlinear dynamic systems, these improvements were 37.96% and 20.48%, respectively. These rates increased even more with respect to the standard ABC algorithm. Compared to the ABC algorithm, performance increases of up to 69.57% and 46.82% were achieved.

The proposed variant was used for ANN training for the identification of nonlinear systems in this study. As is widely known, ANN is used in modeling of many systems. In future studies, the HABCES algorithm can be used as a training algorithm for modeling different systems, or new variants can be suggested by creating new updates to increase the performance of the HABCES algorithm.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Abiodun, O.I.; Jantan, A.; Omolara, A.E.; Dada, K.V.; Mohamed, N.A.; Arshad, H. State-of-the-art in artificial neural network applications: A survey. *Heliyon* **2018**, *4*, e00938. [CrossRef] [PubMed]
2. Devikanniga, D.; Vetrivel, K.; Badrinath, N. Review of meta-heuristic optimization based artificial neural networks and its applications. *J. Phys. Conf. Ser.* **2019**, *1362*, 012074. [CrossRef]
3. Nur, A.S.; Radzi, N.H.M.; Ibrahim, A.O. Artificial neural network weight optimization: A review. *TELKOMNIKA Indones. J. Electr. Eng.* **2014**, *12*, 6897–6902. [CrossRef]
4. Kumar, D. Meta-heuristic Techniques to Train Artificial Neural Networks for Medical Image Classification: A Review. *Recent Adv. Comput. Sci. Commun. (Former. Recent Pat. Comput. Sci.)* **2022**, *15*, 513–530.
5. Tayarani-N, M.-H.; Yao, X.; Xu, H. Meta-heuristic algorithms in car engine design: A literature survey. *IEEE Trans. Evol. Comput.* **2014**, *19*, 609–629. [CrossRef]
6. Yang, B.; Wang, J.; Zhang, X.; Yu, T.; Yao, W.; Shu, H.; Zeng, F.; Sun, L. Comprehensive overview of meta-heuristic algorithm applications on PV cell parameter identification. *Energy Convers. Manag.* **2020**, *208*, 112595. [CrossRef]
7. Wu, Y. A survey on population-based meta-heuristic algorithms for motion planning of aircraft. *Swarm Evol. Comput.* **2021**, *62*, 100844. [CrossRef]
8. Yang, B.; Wang, J.; Yu, L.; Shu, H.; Yu, T.; Zhang, X.; Yao, W.; Sun, L. A critical survey on proton exchange membrane fuel cell parameter estimation using meta-heuristic algorithms. *J. Clean. Prod.* **2020**, *265*, 121660. [CrossRef]
9. Faramarzi, A.; Heidarinejad, M.; Stephens, B.; Mirjalili, S. Equilibrium optimizer: A novel optimization algorithm. *Knowl.-Based Syst.* **2020**, *191*, 105190. [CrossRef]
10. Faramarzi, A.; Heidarinejad, M.; Mirjalili, S.; Gandomi, A.H. Marine Predators Algorithm: A nature-inspired metaheuristic. *Expert Syst. Appl.* **2020**, *152*, 113377. [CrossRef]
11. Li, S.; Chen, H.; Wang, M.; Heidari, A.A.; Mirjalili, S. Slime mould algorithm: A new method for stochastic optimization. *Future Gener. Comput. Syst.* **2020**, *111*, 300–323. [CrossRef]
12. Abualigah, L.; Abd Elaziz, M.; Sumari, P.; Geem, Z.W.; Gandomi, A.H. Reptile Search Algorithm (RSA): A nature-inspired meta-heuristic optimizer. *Expert Syst. Appl.* **2022**, *191*, 116158. [CrossRef]
13. Zhao, S.; Zhang, T.; Ma, S.; Chen, M. Dandelion Optimizer: A nature-inspired metaheuristic algorithm for engineering applications. *Eng. Appl. Artif. Intell.* **2022**, *114*, 105075. [CrossRef]
14. Ahmadianfar, I.; Heidari, A.A.; Gandomi, A.H.; Chu, X.; Chen, H. RUN beyond the metaphor: An efficient optimization algorithm based on Runge Kutta method. *Expert Syst. Appl.* **2021**, *181*, 115079. [CrossRef]
15. Ahmadianfar, I.; Heidari, A.A.; Noshadian, S.; Chen, H.; Gandomi, A.H. INFO: An efficient optimization algorithm based on weighted mean of vectors. *Expert Syst. Appl.* **2022**, *195*, 116516. [CrossRef]
16. Karaboga, D. Artificial bee colony algorithm. *Scholarpedia* **2010**, *5*, 6915. [CrossRef]
17. Karaboga, D.; Basturk, B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *J. Glob. Optim.* **2007**, *39*, 459–471. [CrossRef]
18. Yang, X.-S.; Deb, S. Cuckoo search via Lévy flights. In Proceedings of the 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), Coimbatore, India, 9–11 December 2009; pp. 210–214.
19. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95-International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; pp. 1942–1948.

20. Storn, R.; Price, K. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]
21. Yang, X.-S. Flower pollination algorithm for global optimization. In Proceedings of the International Conference on Unconventional Computing and Natural Computation, Orléan, France, 3–7 September 2012; pp. 240–249.
22. Karaboga, D.; Gorkemli, B.; Ozturk, C.; Karaboga, N. A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. *Artif. Intell. Rev.* **2014**, *42*, 21–57. [CrossRef]
23. Karaboga, D.; Akay, B. Artificial bee colony (ABC) algorithm on training artificial neural networks. In Proceedings of the 2007 IEEE 15th Signal Processing and Communications Applications, Eskisehir, Turkey, 11–13 June 2007; pp. 1–4.
24. Karaboga, D.; Akay, B.; Ozturk, C. Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks. In Proceedings of the International Conference on Modeling Decisions for Artificial Intelligence, Kitakyushu, Japan, 16–18 August 2007; pp. 318–329.
25. Ozturk, C.; Karaboga, D. Hybrid artificial bee colony algorithm for neural network training. In Proceedings of the 2011 IEEE Congress of Evolutionary Computation (CEC), New Orleans, LA, USA, 5–8 June 2011; pp. 84–88.
26. Karaboga, D.; Ozturk, C. Neural networks training by artificial bee colony algorithm on pattern classification. *Neural Netw. World* **2009**, *19*, 279.
27. Kaya, E. A Comprehensive Comparison of the Performance of Metaheuristic Algorithms in Neural Network Training for Nonlinear System Identification. *Mathematics* **2022**, *10*, 1611. [CrossRef]
28. Uzlu, E.; Akpınar, A.; Özturk, H.T.; Nacar, S.; Kankal, M. Estimates of hydroelectric generation using neural networks with the artificial bee colony algorithm for Turkey. *Energy* **2014**, *69*, 638–647. [CrossRef]
29. Xu, F.; Pun, C.-M.; Li, H.; Zhang, Y.; Song, Y.; Gao, H. Training feed-forward artificial neural networks with a modified artificial bee colony algorithm. *Neurocomputing* **2020**, *416*, 69–84. [CrossRef]
30. Kaya, E.; Başтemur Kaya, C. A novel neural network training algorithm for the identification of nonlinear static systems: Artificial bee colony algorithm based on effective scout bee stage. *Symmetry* **2021**, *13*, 419. [CrossRef]
31. Ghanem, W.A.; Jantan, A. Training a neural network for cyberattack classification applications using hybridization of an artificial bee colony and monarch butterfly optimization. *Neural Process. Lett.* **2020**, *51*, 905–946. [CrossRef]
32. Shah, H.; Tairan, N.; Garg, H.; Ghazali, R. A quick gbest guided artificial bee colony algorithm for stock market prices prediction. *Symmetry* **2018**, *10*, 292. [CrossRef]
33. Yildiz, A.R. A new hybrid artificial bee colony algorithm for robust optimal design and manufacturing. *Appl. Soft Comput.* **2013**, *13*, 2906–2912. [CrossRef]
34. Karaboga, D.; Kaya, E. An adaptive and hybrid artificial bee colony algorithm (aABC) for ANFIS training. *Appl. Soft Comput.* **2016**, *49*, 423–436. [CrossRef]
35. Jadon, S.S.; Tiwari, R.; Sharma, H.; Bansal, J.C. Hybrid artificial bee colony algorithm with differential evolution. *Appl. Soft Comput.* **2017**, *58*, 11–24. [CrossRef]
36. Li, H.; Pun, C.-M.; Xu, F.; Pan, L.; Zong, R.; Gao, H.; Lu, H. A hybrid feature selection algorithm based on a discrete artificial bee colony for Parkinson's diagnosis. *ACM Trans. Internet Technol.* **2021**, *21*, 1–22. [CrossRef]
37. Kefayat, M.; Ara, A.L.; Niaki, S.N. A hybrid of ant colony optimization and artificial bee colony algorithm for probabilistic optimal placement and sizing of distributed energy resources. *Energy Convers. Manag.* **2015**, *92*, 149–161. [CrossRef]
38. Duan, H.-B.; Xu, C.-F.; Xing, Z.-H. A hybrid artificial bee colony optimization and quantum evolutionary algorithm for continuous optimization problems. *Int. J. Neural Syst.* **2010**, *20*, 39–50. [CrossRef]
39. Awadallah, M.A.; Bolaji, A.L.; Al-Betar, M.A. A hybrid artificial bee colony for a nurse rostering problem. *Appl. Soft Comput.* **2015**, *35*, 726–739. [CrossRef]
40. Mazini, M.; Shirazi, B.; Mahdavi, I. Anomaly network-based intrusion detection system using a reliable hybrid artificial bee colony and AdaBoost algorithms. *J. King Saud Univ. Comput. Inf. Sci.* **2019**, *31*, 541–553. [CrossRef]
41. Stephan, P.; Stephan, T.; Kannan, R.; Abraham, A. A hybrid artificial bee colony with whale optimization algorithm for improved breast cancer diagnosis. *Neural Comput. Appl.* **2021**, *33*, 13667–13691. [CrossRef]
42. Gaidhane, P.J.; Nigam, M.J. A hybrid grey wolf optimizer and artificial bee colony algorithm for enhancing the performance of complex systems. *J. Comput. Sci.* **2018**, *27*, 284–302. [CrossRef]
43. Gupta, S.; Deep, K. Hybrid sine cosine artificial bee colony algorithm for global optimization and image segmentation. *Neural Comput. Appl.* **2020**, *32*, 9521–9543. [CrossRef]
44. Badem, H.; Basturk, A.; Caliskan, A.; Yuksel, M.E. A new hybrid optimization method combining artificial bee colony and limited-memory BFGS algorithms for efficient numerical optimization. *Appl. Soft Comput.* **2018**, *70*, 826–844. [CrossRef]
45. Mallala, B.; Papana, V.P.; Sangu, R.; Palle, K.; Chinthalacheruvu, V.K.R. Multi-Objective Optimal Power Flow Solution Using a Non-Dominated Sorting Hybrid Fruit Fly-Based Artificial Bee Colony. *Energies* **2022**, *15*, 4063. [CrossRef]
46. Zhang, L.; Xuan, J.; Shi, T. Obtaining More Accurate Thermal Boundary Conditions of Machine Tool Spindle Using Response Surface Model Hybrid Artificial Bee Colony Algorithm. *Symmetry* **2020**, *12*, 361. [CrossRef]