International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Swarm Intelligence Algorithm with Guided Exploitations: A Case Study with Artificial Bee Colony Algorithm

Syeda Shabnam Hasan[1], Md. Shahriar Rahman[2]

Ahsanullah University of Science and Technology, Department of Computer Science and Engineering,
141 & 142, Love Road, Tejgaon Industrial Area, Dhaka-1208, Bangladesh

Abstract: ***During any meta-heuristic search, two opposite processes are found in action, namely the explorations and exploitations. Although they might seem to operate in opposite directions, they are actually counterparts, and synergy between them may improve the final outcome of the algorithm. This is especially true for complex, high dimensional problems, because the search algorithm has to avoid many local optima to find a good near optimum solution. There exist many swarm intelligence algorithms that report the necessity of a proper balance between explorations and exploitations. This paper presents a concrete example of a swarm intelligence algorithm, i.e., the Artificial Bee Colony (ABC) algorithm that finds improvement by balancing between explorations and exploitations. In this paper, we have introduced ABC with Guided Exploitations (ABC-GE), a novel algorithm that improves over the basic ABC algorithm. ABC-GE augments each candidate solution with a control parameter that controls the proportion of explorative and exploitative perturbations and thus affects how new trial solutions are produced from the existing ones. This control parameter is automatically adjusted at the individual solution level, separately for each candidate solution x_i, to adjust the proportions of explorations and exploitations around x_i. ABC-GE is tested on a number of benchmark problems on continuous optimization and compared with the basic ABC algorithm. Results show that the performance of ABC-GE is overall better than the basic ABC algorithm.***

Keywords: Artificial bee colony algorithm, exploration and exploitation, continuous optimization, meta-heuristic optimization

1. Introduction

Recently the research community has seen the emergence of many evolutionary and swarm intelligence algorithms, each one trying to improve over the others on existing benchmark and real world problems. The Artificial Bee Colony (ABC) algorithm is a recently introduced [1] swarm intelligence algorithm that tries to mimic the intelligent food foraging behavior of honey bees. ABC shows very competitive and often better results in comparison to existing evolutionary and swarm intelligence algorithms [1], [2], such as the genetic algorithm (GA), differential evolution (DE) and particle swarm optimization (PSO). Since its advent, ABC has been successfully applied to wide and diverse range of problems, such as continuous optimization [2], discrete optimization [3], constrained optimization [4], multi-objective optimization [5], design optimization [6], training neural network [7], IIR filter [8], PID controller [9], parameterizing of milling processes [10] and so on [11].

The ABC algorithm is simple in concept, easy to implement and requires fewer control parameters [12]. However, similar to other population based meta-heuristic algorithms, ABC also has its own challenges and limitations. A major problem that may occur with ABC is fitness stagnation [14], where the entire population of solutions stops improving, even without converging to some local optima, because the fitness based selection scheme fails to find new, better trial solutions that can enter the population by replacing the existing solutions. This may be due to the reason that the pool of candidate solutions has prematurely converged around one or few locally optimal points, especially for complex high dimensional multimodal problems [2], [13]. The main reason behind premature convergence is too much exploitation at the expense of reduced explorations. ABC drives its search towards global optimum with two operators — perturbation and selection. The perturbation operation is responsible for explorations by random variations of existing solutions, while the fitness based selection operation performs exploitations of the search regions explored so far. However, both these operations are more aligned towards exploitations than explorations. The perturbation operation of ABC perturbs a single parameter of an existing solution and thus produces the new trial solution in the neighborhood of the original solution, which is exploitative. The selection operation of ABC can accept only the better solutions, which is exploitative too. This paper introduces ABC with Guided exploitations (ABC-GE), a novel improvement over the basic ABC algorithm that tries to automatically balance the exploitations with explorations, separately for every candidate solution. ABC-GE augments each candidate solution $\boldsymbol{x}_i$ with a control parameters $p[\boldsymbol{x}_i]$ which affects the perturbation operations on $\boldsymbol{x}_i$ by controlling the degree of explorations and exploitations around $\boldsymbol{x}_i$. The value of $p[\boldsymbol{x}_i]$ is automatically adjusted, cycle (i.e., iteration) by cycle, using an adaptive technique to increase the likelihood of producing more effective perturbations on $\boldsymbol{x}_i$. ABC-GE is tested on a benchmark suite of 30 continuous functions of different complexity. Results are compared with the basic ABC [2] algorithm which show that ABC-GE can sometimes perform better than the basic ABC algorithm.

2. The ABC Algorithm

Honey bees in nature have to forage over a vast area in search of good sources of nectar. After an initial exploration stage, more bees are employed to collect honey from more profitable food sources whereas fewer bees are assigned to

the less worthy sources. After returning the hive, each bee goes to the 'dance floor' and performs a special dance known as the 'waggle dance' to share the information of the food source it has found. The 'onlooker' bees, waiting around the dance floor, observe the waggle dances of the 'employed' bees and pick any of them to follow and collect nectar from the vicinity of its food source. Some scout bees are also assigned for random explorations of the search space to find new food sources. The basic ABC algorithm [1,2] mimics the food foraging behavior of honey bees with the same three groups of bees — employed, onlooker and scout bees. A bee working to forage a particular food source (i.e., candidate solution) and searching only around its vicinity is called an employed bee. Onlooker bees randomly pick and follow any of the employed bees. The probability of picking an employed bee is proportional to the quality of its food source. Scout bees can perform random explorations of the search space to find new food sources. If the employed and onlooker bees, even after limit attempts, fail to find a better food position around a particular food $\boldsymbol{x}_i$, then $\boldsymbol{x}_i$ is abandoned and replaced by initiating a scout bee and its food source is placed uniformly at random across the search space. In the original implementation of the ABC algorithm, half of the colony is employed bees, the other half is onlooker bees, and scout bees are created on demand only when a food source fails to improve with several attempts. Fig. 1 presents the pseudocode for the basic ABC algorithm. Each cycle (i.e., iteration) of ABC consists of foraging by the employed bees (steps 4–5, Fig. 1), then foraging by the onlookers (steps 7–9), followed by placement of the scout bees (step 10). Each of these stages is described below.

Foraging by employed bees: Suppose, an employed bee is currently positioned at a food source position $\boldsymbol{x}_i$. During this stage, each employed bee searches in the vicinity of its current position $\boldsymbol{x}_i$ to produce new trial food source $\boldsymbol{v}_i$ using (1), where $j \in \{1, 2, \ldots, D\}$ and $k \in \{1, 2, \ldots, SN\}$ are randomly picked indices, D is dimensionality of the problem, SN is the number of food positions and φ_{ij} is a uniform random value ~ [-1, 1].

$$v_{ij} = x_{ij} + \varphi_{ij} (x_{kj} - x_{ij}) \tag{1}$$

Thus, the new solution $\boldsymbol{v}_i$ is produced from $\boldsymbol{x}_i$ by perturbing its randomly picked j-th parameter and using the information of $\boldsymbol{x}_i$ and another randomly picked solution $\boldsymbol{x}_k$. If $\boldsymbol{v}_i$ has better 'fitness' than the old food position $\boldsymbol{x}_i$, then $\boldsymbol{x}_i$ is replaced by $\boldsymbol{v}_i$. For the problem of function optimization, where f is the function to be minimized, ABC computes the 'fitness' of a candidate solution $\boldsymbol{x}_i$ using (2).

$$fitness(\boldsymbol{x}_i) = \begin{cases} \dfrac{1}{1+f(\boldsymbol{x}_i)}; & \text{if } f(\boldsymbol{x}_i) \geq 0 \\ 1+|f(\boldsymbol{x}_i)| & \text{otherwise} \end{cases} \tag{2}$$

Foraging by onlooker bees: During this stage, each onlooker bee randomly picks an employed bee to follow and forages only around the vicinity of its food source. The probability w_i that the employed bee with food source $\boldsymbol{x}_i$ would be picked by an onlooker bee is computed using (3), which makes the probability w_i to be proportional to fitness ($\boldsymbol{x}_i$).

$$w_i = \frac{fitness(x_i)}{\sum_{n=1}^{SN} fitness(x_n)} \tag{3}$$

Like the employed bees, each onlooker bee also employs (1) to produce trial food source $\boldsymbol{v}_i$ in the vicinity of its current food source position $\boldsymbol{x}_i$. If $\boldsymbol{v}_i$ has better fitness than $\boldsymbol{x}_i$, then $\boldsymbol{x}_i$ is replaced by $\boldsymbol{v}_i$. Otherwise, $\boldsymbol{v}_i$ is discarded.

Placement of Scout bees: A scout bee is created only when a particular food source $\boldsymbol{x}_i$ failed to be improved over the last '*limit*' iterations. The bee employed to $\boldsymbol{x}_i$ now becomes a scout bee and its food source is replaced randomly across the search space using (4), where $j = 1, 2, \ldots, D$ and $[min_j, max_j]$ is the search space along the j-th dimension.

$$x_{ij} = min_j + rand\,(0,1) * (max_j - min_j) \tag{4}$$

3. Existing Variants of ABC Algorithm

There exist a number of recent studies (e.g., [15]–[24]) that try to alter the explorative and/or exploitative properties of the basic ABC algorithm. For example, ABC with self-adaptive mutation (ABC-SAM) [15] introduces an adaptive mutation scaling factor SF_i for every candidate solution $\boldsymbol{x}_i$ and tries to ensure both explorations and exploitations by periodically adjusting the value of SF_i using two different distributions — one explorative and the other exploitative. The cooperative ABC (CABC) [16] tries to enforce more explorations by decomposing the search space into multiple subspaces and by employing multiple bee colonies to explore through different subspaces. ABC with diversity strategy (DABC) [17] tries to maintain sufficient level of population diversity for conducting more explorations by alternating between two different perturbation schemes. Chaotic ABC (ChABC) [18] tries to improve the explorative characteristics of ABC by employing chaotic dynamics instead of random number generators. The Gbest-guided ABC (GABC) [20] tries to improve the exploitations and convergence speed of ABC by altering its perturbation operation using the information of the global best solution found so far. Hooke Jeeves ABC (HJABC) [21]–[22] is a hybrid ABC-variant that intensifies the exploitative operations by hybridizing ABC with a local search technique (i.e., the Hooke Jeeves pattern search). The Elitist ABC (EABC) [24] is another exploitative ABC variant that hybridizes ABC with two different local search operators to intensify the degree of exploitations around the best candidate solution found so far. Thus, most existing ABC-variants try to improve either the exploitative (e.g., [20]–[24]) or the explorative ([15]–[19]) characteristics of the basic ABC algorithm. The exploitative improvements are

Algorithm: Artificial Bee Colony (ABC) Algorithm
1: Initialize a population of *SN* food source positions (candidate solutions) $\boldsymbol{x_i}$, for $i = 1, 2, \ldots, SN$. Each $\boldsymbol{x_i}$ is a vector of *D* parameters: $\boldsymbol{x_i} = [x_{i1}, x_{i2}, \ldots, x_{iD}]^T$
2: Evaluate the fitness of each food source position using (2).
3: **repeat**
4: For each employed bee, perturb its food source position $\boldsymbol{x_i}$ to produce a new food position $\boldsymbol{v_i}$ using (1).
5: Evaluate each new solution $\boldsymbol{v_i}$ by (2). If $\boldsymbol{v_i}$ has higher fitness than $\boldsymbol{x_i}$, then accept $\boldsymbol{v_i}$ to replace $\boldsymbol{x_i}$. Else, discard $\boldsymbol{v_i}$.
6. Calculate the probability w_i associated with each food source position $\boldsymbol{x_i}$ using (3).
7: For each onlooker bee, assign it to a food source $\boldsymbol{x_i}$, proportionally based on the probability w_i.
8: For each onlooker bee, perturb its food source position $\boldsymbol{x_i}$ to produce a new food position $\boldsymbol{v_i}$ using (1).
9: Evaluate each new solution$\boldsymbol{v_i}$ using (2). If $\boldsymbol{v_i}$ is better than $\boldsymbol{x_i}$, then accept $\boldsymbol{v_i}$ to replace $\boldsymbol{x_i}$. Else, discard $\boldsymbol{v_i}$.
10: If a food source has not improved during the last *limit* cycles, then abandon it and replace it with a new randomly placed scout bee with its food source $\boldsymbol{x_i}$ produced by (4).
11: Memorize the best food source position found so far
12: Set cycle counter $C=C+1$
13: **until** C = Maximum cycle number (*MCN*)
14: **return** the best food source position (i.e., candidate solution) found so far

Figure 1: Algorithm for the basic Artificial Bee Colony (ABC) algorithm

usually based on intensifying the search around the best solution(s) found so far (e.g., [20], [21], [24]) and hybridizing efficient local search operators with ABC (e.g., [21], [23], [24]), while the explorative improvements can be based on more population diversity (e.g., [16], [17]) and more explorative selection and/or perturbation operations ([15], [18], [19]). But none of these algorithms considers the individual explorative/exploitative requirements of each candidate solution separately; rather they employ some population-wide global strategy, identically for all candidate solutions, which is significantly improved in the proposed algorithm — ABC-GE, as described in the following section.

4. The Proposed Algorithm — ABC-GE

ABC-GE tries to improve over the basic ABC algorithm by adapting and customizing the degree of explorations and exploitations at the individual solution level, i.e., separately for every candidate solution. ABC-GE includes a control parameter $p[\boldsymbol{x}_i]$ within each bee (i.e., candidate solution) $\boldsymbol{x_i}$. The control parameter $p[\boldsymbol{x}_i]$ automatically controls the proportion of explorative and exploitative perturbations on the candidate solution $\boldsymbol{x}_i$. The value of $p[\boldsymbol{x}_i]$ is gradually adapted to achieve higher rate of 'successful' perturbations. A perturbation is considered 'successful' only if the new trial solution $\boldsymbol{v_i}$ has higher fitness value than the original solution $\boldsymbol{x_i}$. A detailed description of how the control parameter $p[\boldsymbol{x}_i]$ is employed and adapted is as follows. ABC-GE employs two different perturbation schemes — one for explorations, the other for exploitations. Both the perturbation schemes are based on the same expression (1), but they differ in how the random value φ_{ij} is picked. For explorative perturbations, $\varphi_{ij}$is picked uniformly at random from the interval [-2.5, 2.5], while the exploitative perturbations pick φ_{ij}uniformly at random from an exploitative, smaller interval of [-0.5, 0.5].

Explorative perturbation:
$v_{ij} = x_{ij} + \varphi_{ij} (x_{kj} - x_{ij})$, where $\varphi_{ij} \sim UR$ (-2.5, 2.5) (5)

Exploitative perturbation:
$v_{ij} = x_{ij} + \varphi_{ij} (x_{kj} - x_{ij})$, where $\varphi_{ij} \sim UR$ (-0.5, 0.5) (6)

Here, UR (a, b) produces a uniform random value from the interval (a, b). The explorative perturbation is likely to produce a larger value of φ_{ij} from the wider interval, which is more likely to produce the new candidate solution $\boldsymbol{v}_i$ farther from the parent candidate solution $\boldsymbol{x}_i$, and hence ensures more search space explorations. In contrast, the narrow interval of [-0.5, 0.5] is likely to produce smaller φ_{ij} values which produces the offspring $\boldsymbol{v}_i$ in the vicinity of its parent $\boldsymbol{x}_i$, and hence is likely to be an exploitation.

But how does ABC-GE decide on whether to perform explorative or exploitative perturbation on $\boldsymbol{x}_i$? This is done probabilistically — the current values of $p[\boldsymbol{x}_i]$ and $1- p[\boldsymbol{x}_i]$ denote the probability of exploitative and explorative perturbations on $\boldsymbol{x}_i$, respectively. The value of $p[\boldsymbol{x}_i]$ is automatically adapted using the incremental learning experience of $\boldsymbol{x}_i$, which includes the number of successes and failures by explorative and exploitative perturbations on $\boldsymbol{x}_i$. Initially, $p[\boldsymbol{x}_i]$ is set to 0.5 for every solution $\boldsymbol{x}_i$, which makes exploitative and explorative perturbations equally desired. After the initial learning period of t_1 cycles, ABC-GE starts adjusting the $p[\boldsymbol{x}_i]$ value for each $\boldsymbol{x}_i$. To do this, ABC-GE keeps record of the number of successes and failures by exploitative and explorative perturbations on $\boldsymbol{x}_i$ over the last t_1 cycles. Suppose,

s_1 : Number of ***successes*** by ***explorative*** perturbations on $\boldsymbol{x}_i$
f_1 : Number of ***failures*** by ***explorative*** perturbations on $\boldsymbol{x}_i$
s_2 : Number of successes by ***exploitative*** perturbations on $\boldsymbol{x}_i$
f_2 : Number of ***failures*** by ***exploitative*** perturbations on $\boldsymbol{x}_i$

Now, the effectiveness of the explorative perturbation (eff_{ER}) and exploitative perturbation (eff_{ET}) on $\boldsymbol{x}_i$ are computed as:

$$eff_{ER} = (s_1) / (s_1 + f_1) \text{ and } eff_{ET} = (s_2) / (s_2 + f_2)$$

Now, the adjusted probability of exploitative perturbation on $\boldsymbol{x}_i$ (i.e., the adjusted value of $p[\boldsymbol{x}_i]$) is computed using (7).

$$p[\boldsymbol{x}_i] = \frac{eff_{ET}}{eff_{ER} + eff_{ET}} \quad (7)$$

ABC-GE also ensures that $p[\boldsymbol{x}_i]$ never drops below p_{low} or rises above p_{high} to avoid the complete domination by either mode of exploitative or explorative perturbations.

5. Experimental Studies

To evaluate the performance of ABC-GE and to compare it with the basic ABC [2] algorithm, this paper uses a set of benchmark problems which has 30 standard functions, including 18 scalable high dimensional functions with dimensionality D=30, 60, as well as 12 low dimensional multimodal functions with $D \leq 10$. The suite contains both unimodal (i.e., f_1–f_9) and multimodal (i.e., f_{10}–f_{30}), separable (e.g., f_1, f_3, f_8) and non-separable (e.g., f_2, f_4, f_5), high dimensional (i.e., f_1–f_{18}) and low dimensional (i.e., f_{19}–f_{30}) functions. These functions have been widely used with many other evolutionary and swarm intelligence algorithms (e.g., [2], [15]–[18], [26]–[28]). Each function is briefly presented in Table 1. More details can be found in [2], [21], [28].

Based on their properties, the benchmark functions (Table 1) can be divided into three groups – functions with no local minima (i.e., unimodal functions f_1–f_9), large number of local minima (i.e., high dimensional multimodal functions f_{10}–f_{18}) and only a few local minima (i.e., low dimensional multimodal functions f_{19}–f_{30}). To minimize a multimodal function, the optimization algorithm should have both explorative and exploitative capabilities, because it has to avoid being trapped around the locally minimal points and continue both explorations and exploitations until it locates the neighbourhood of a global minimum. Some of the multimodal functions can have tens or even hundreds of local minima, even with just two dimensions (e.g., Rastrigin function f_{10}). The number of local minima can increase exponentially with the number of dimensions, which makes the optimization extremely difficult.

Both ABC and ABC-GE have three parameters in common, which are the population size SN, maximum cycle number MCN and $limit$. For functions f_1–f_{18} with D = 30, ABC-GE used SN = 100, MCN = 1000 and $limit$ = 100. For the larger variants with D = 60, the value of SN is kept the same (i.e., 100), but $limit$ and MCN are set to 200 and 2000, respectively. For the low dimensional f_{19}–f_{30}, ABC-GE sets SN = 100, MCN = 100 and $limit = 10 * D$. The other parameters of ABC-GE are set as: t_1 = 30, p_{low} = 0.05 and p_{high} = 0.95. These values are chosen with some initial experiments and are not meant to be the optimum.

Table 2 presents the results of ABC-GE on the 30 standard benchmark functions and compares the results with the basic ABC [2] algorithm. All the algorithms have made 50 independent runs on each function and the mean andstandard deviation of the best found solutions are presented in Table 2. We summarize our observations on the experimental results in the following few points.

- On the simpler functions, i.e., the unimodal functions f_1 – f_9, and the low dimensional $f_{19} - f_{30}$, the performance of ABC is slightly better than the proposed algorithm ABC-GE. On either of these two function families, ABC performs better than ABC-GE on two functions, while ABC-GE performs better on one function only. On the remaining functions, their performance is similar, i.e., there is no statistical significance of their performance difference, as tested by the t-test with α = 0.95.
- On the most complex function family, i.e., the high dimensional multimodal functions $f_{10} - f_{18}$, the performance of ABC-GE is better than the original ABC algorithm. On these functions, ABC-GE significantly outperforms ABC on four functions, while ABC performs better on two functions only.
- In summary, we can conclude that ABC-GE is better suited for more complex multimodal and high dimensional functions. This is because ABC-GE puts more emphasis on explorations (rather than exploitations), which is usually necessary for more complex multimodal functions, while the simpler functions may require more exploitations (rather than explorations).

6. Conclusion and Suggestion for Further Study

This paper introduces ABC-GE — an improvement over the basic ABC algorithm [2] that tries to adaptively control the degree of explorations and exploitations, separately for each candidate solution. ABC-GE includes a control parameter — $p[\boldsymbol{x}_i]$ within each candidate solution $\boldsymbol{x}_i$ and employs an adaptive technique to adjust its values gradually, separately for each candidate solution. The control parameter $p[\boldsymbol{x}_i]$ controls the proportion of exploitative and explorative perturbations on $\boldsymbol{x}_i$ and is gradually adapted by ABC-GE based on the previous successes and failures of the exploitative and explorative perturbations on $\boldsymbol{x}_i$. The results indicate that putting a balanced emphasis on the explorations make ABC-GE more suitable for complex multimodal and high dimensional functions.

There may be several possible future research directions based on this study. Firstly, ABC-GE uses a simple strategy to adjust the control parameters $p[\boldsymbol{x}_i]$ for each candidate solution $\boldsymbol{x}_i$. A more sophisticated strategy, such as considering the properties of fitness landscape around $\boldsymbol{x}_i$, or using a strategy parameterized by the maturity of the optimization process may be more effective to balance between exploitations and explorations around $\boldsymbol{x}_i$. Secondly, the quality of the final solution could be improved further by using an exploitative and efficient local searcher. This may pinpoint the global minimum more precisely. Thirdly, ABC-GE can be hybridized with other existing evolutionary, swarm intelligence and machine learning techniques to further improve its results. Finally, ABC-GE has been employed on the continuous optimization problems. It would be interesting to examine how well ABC-GE can perform on many other existing problems, especially the discrete and real world ones.

Table 1: Benchmark functions for experimental study. D: dimensionality of the function, S: search space, f_{min}: the value of the function at the global minimum, C: function characteristics with values — U: Unimodal, M: Multimodal, S: Separable and N: Non-Separable.

No	Function	D	S	C	f_{min}
f_1	Sphere	30 and 60	$[-100, 100]^D$	*US*	0
f_2	Schwefel 2.22	30 and 60	$[-10, 10]^D$	*UN*	0
f_3	Schwefel 2.21	30 and 60	$[-10, 10]^D$	*US*	0
f_4	Schwefel 1.2	30 and 60	$[-100, 100]^D$	*UN*	0
f_5	Powell	24	$[-4, 5]^D$	*UN*	0
f_6	Dixon-Price	30 and 60	$[-10, 10]^D$	*UN*	0
f_7	Rosenbrock	30 and 60	$[-30, 30]^D$	*UN*	0
f_8	Step	30 and 60	$[-100, 100]^D$	*US*	0
f_9	Quartic	30 and 60	$[-1.28, 1.28]^D$	*US*	0
f_{10}	Rastrigin	30 and 60	$[-5.12, 5.12]^D$	*MS*	0
f_{11}	Non-continuous Rastrigin	30 and 60	$[-5.12, 5.12]^D$	*MS*	0
f_{12}	Schwefel	30 and 60	$[-500, 500]^D$	*MS*	0
f_{13}	Ackley	30 and 60	$[-32, 32]^D$	*MN*	0
f_{14}	Griewank	30 and 60	$[-600, 600]^D$	*MN*	0
f_{15}	Alpine	30 and 60	$[-10, 10]^D$	*MS*	0
f_{16}	Weierstrass	30 and 60	$[-0.5, 0.5]^D$	*MS*	0
f_{17}	Penalized	30 and 60	$[-50, 50]^D$	*MN*	0
f_{18}	Penalized2	30 and 60	$[-50, 50]^D$	*MN*	0
f_{19}	Foxholes	2	$[-65.536, 65.536]^D$	*MS*	1
f_{20}	Kowalik	4	$[-5, 5]^D$	*MN*	3.07e-04
f_{21}	Six Hump Camel Back	2	$[-5, 5]^D$	*MN*	-1.0316
f_{22}	Branin	2	$[-5, 10]$ x $[0, 15]$	*MS*	0.398
f_{23}	Hartman3	3	$[0, 1]^D$	*MN*	-3.86
f_{24}	Hartman6	6	$[0, 1]^D$	*MN*	-3.32
f_{25}	Shekel5	4	$[0, 10]^D$	*MN*	-10.15
f_{26}	Shekel7	4	$[0, 10]^D$	*MN*	-10.40
f_{27}	Shekel10	4	$[0, 10]^D$	*MN*	-10.55
f_{28}	Fletcher Powell	10	$[-\pi, \pi]^D$	*MN*	0
f_{29}	Michalewicz	10	$[0, \pi]^D$	*MS*	-9.66015
f_{30}	Langerman	10	$[0, 10]^D$	*MN*	-1.4

Table 2: Comparison of ABC-GE with the original ABC [2] algorithm on the 30 standard benchmark functions.

No	f_{min}	D	G	ABC		ABC-GE		Better Performance by
				Mean	Std. Dev.	Mean	Std. Dev.	
f_1	0	30	1000	2.45e–11	7.72e–12	7.20e–09	8.59e–10	ABC
		60	2000	3.75e–10	2.01e–11	6.82e–07	1.65e–07	
f_2	0	30	1000	5.05e–07	1.74e–07	2.76e–05	5.35e–06	ABC
		60	2000	5.58e–06	1.17e–06	3.96e–04	6.98e–05	
f_3	0	30	1000	4.18e+01	5.90	4.36e+01	7.81	Similar
		60	2000	7.31e+01	6.88	6.83e+01	8.53	
f_4	0	30	1000	8.32e–10	9.75e–11	7.66e–10	9.03e–11	Similar
		60	2000	4.50e–09	5.64e–10	5.32e–09	1.04e–09	
f_5	0	24	1000	6.61e+00	1.07e+00	3.76e–01	5.94e–02	ABC-GE
f_6	0	30	1000	6.67e–01	1.21e–01	5.97e–01	2.82e–01	Similar
		60	2000	6.66e–01	1.05e–01	5.65e–01	2.05e–01	
f_7	0	30	1000	4.25e–01	1.18e–01	3.87e–01	1.74 e–01	Similar
		60	2000	2.02e–01	6.92e–02	1.85e–01	1.09 e–01	
f_8	0	30	1000	0	0	0	0	Similar
		60	2000	0	0	0	0	
f_9	0	30	1000	8.60e–13	8.32e–13	7.16e–13	2.54e–13	Similar
		60	2000	9.31e–12	7.17e–12	8.06e–12	3.56e–12	
f_{10}	0	30	1000	1.72e–14	1.56e–14	8.11e–17	1.84e–17	ABC-GE
		60	2000	2.84e–13	8.01e–14	3.24e–13	1.57e–13	
f_{11}	0	30	1000	2.33e–08	7.49e–09	3.16e–08	1.76e–08	Similar
		60	2000	6.64e–07	1.51e–07	7.71e–07	4.80e–08	
f_{12}	–12569.5	30	1000	–11346.79	2.77e+02	–11326.20	3.86e+02	Similar
	–25138.9	60	2000	–22530.82	4.08e+02	–22543.29	5.12e+02	
f_{13}	0	30	1000	2.93e–06	3.38e–07	8.22e–09	3.36e–09	ABC-GE
		60	2000	4.65e–06	1.07e–06	5.17e–09	2.16e–09	
f_{14}	0	30	1000	4.55e–08	6.54e–09	3.13e–04	7.82e–05	ABC
		60	2000	8.01e–07	2.64e–07	7.26e–04	1.01e–05	
f_{15}	0	30	1000	3.34e–04	3.76e–05	4.02e–03	1.05e–03	ABC
		60	2000	7.49e–03	9.58e–04	6.72e–02	2.28e–02	
f_{16}	0	30	1000	3.36e–01	9.58e–02	2.88e–01	7.07e–02	Similar
		60	2000	8.99e–01	3.09e–01	9.40–01	3.45e–01	
f_{17}	0	30	1000	5.47e–12	2.09e–13	6.35e–15	8.20e–16	ABC-GE
		60	2000	7.47e–12	1.74e–12	3.19e–14	6.05e–15	
f_{18}	0	30	1000	2.63e–03	1.89e–04	1.86e–04	7.22e–05	ABC-GE
		60	2000	2.66e–03	7.90e–04	1.11e–04	6.59e–05	
f_{19}	1	2	100	1.04	0.04	1.035	0.02	Similar
f_{20}	3.07e–04	4	100	5.98e–04	7.22e–05	5.96e–04	8.10e–05	Similar
f_{21}	–1.0316	2	100	–1.0316	0	–1.0316	0	Similar
f_{22}	0.398	2	100	0.398	7.12e–08	0.398	6.67e–07	Similar
f_{23}	–3.86	3	100	–3.86	7.09e–07	–3.86	2.25e–08	Similar
f_{24}	–3.32	6	100	–3.32	4.74e–13	–3.32	6.96e–13	Similar
f_{25}	–10.15	4	100	–9.61	0.14	–10.05	0.08	ABC-GE

f_{26}	–10.40	4	100	–10.40	8.61e–03	–10.40	3.63e–03	Similar
f_{27}	–10.54	4	100	–10.52	0.08	–10.40	3.12e–06	ABC
f_{28}	0	10	100	13.77	3.80	14.15	0.69	Similar
f_{29}	-9.66015	10	100	-9.66015	0	-9.66015	0	Similar
f_{30}	-1.4	10	100	–0.78	0.09	–0.24	0.08	ABC

References

[1] D. Karaboga, "An idea based on honey bee swarm for numerical optimization", Erciyes University, Kayseri, Turkey, Technical Report-TR06, 2005.
[2] D. Karaboga, B. Akay, "A comparative study of artificial bee colony algorithm", Applied Mathematics and Computation 214 (1) (2009) 108–132.
[3] Q. Bai, X. Yun, "A new hybrid artificial bee colony algorithm for the traveling salesman problem", in: Proc. 3rd Int. Conf. Communication Software and Networks (ICCSN), 2011, pp. 155–159.
[4] N. Stanarevic, M. Tuba, N. Bacanin, "Modified artificial bee colony algorithm for constrained problems optimization", Int. Journal of Mathematical Models and Methods in Applied Sciences 5 (3) (2011) 644–651.
[5] S. Omkar, J. Senthilnath, R. Khandelwal, G. Naik, S. Gopalakrishnan, "Artificial bee colony (ABC) for multi-objective design optimization of composite structures", Applied Soft Computing 11 (1) (2011) 489–499.
[6] F. Kang, J. Li, Q. Xu, "Structural inverse analysis by hybrid simplex artificial bee colony algorithms", Computers and Structures 87 (13–14) (2009) 861–870.
[7] R. Irani, R. Nasimi, "Application of artificial bee colony-based neural network in bottom hole pressure prediction in underbalanced drilling", Journal of Petroleum Science and Engineering 78 (1) (2011) 6–12.
[8] N. Karaboga, "A new design method based on artificial bee colony algorithm for digital IIR filters", Journal of the Franklin Institute 346 (4) (2009) 328–348.
[9] D. Karaboga, B. Akay, "PID controller design by using artificial bee colony, harmony search and bees algorithms", in: Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering 224 (7) (2010) 869–883.
[10] R. Rao, P. Pawar, "Parameter optimization of a multi pass milling process using non-traditional optimization algorithms", Applied Soft Computing 10 (2) (2010) 445-456.
[11] D. Karaboga, B. Gorkemli, C. Ozturk, N. Karaboga, "A comprehensive survey: artificial bee colony (ABC) algorithm and applications", Artificial Intelligence Review (2012) 1–37.
[12] L. Bao, J. Zeng, "Comparison and analysis of the selection mechanism in the artificial bee colony algorithm", in: Proc. 9th Int. Conf. Hybrid Intelligent Systems, 2009, pp. 411–416.
[13] W. Gao, S. Liu, "A modified artificial bee colony algorithm", Computers and Operations Research 39 (3) (2012) 687–697.
[14] J. Lampinen, I. Zelinka, "On stagnation of the differential evolution algorithm", in: Proc. 6th Int. Mendel Conf. on Soft Computing, 2000, pp. 76–83.
[15] M. S. Alam, M. M. Islam, "Artificial bee colony algorithm with self-adaptive mutation: A novel approach for numeric optimization", in: Proc. 2011 IEEE Int. Conf. on Trends and Developments in Converging Technology (TENCON), 2011, pp. 49–53.
[16] M. Abd, "A cooperative approach to the artificial bee colony algorithm", in: IEEE Congress on Evolutionary Computation (CEC), 2010 1–5.
[17] W. Lee, W. Cai, "A novel artificial bee colony algorithm with diversity strategy", in: Proc. 7th Int. Conf. Natural Computation, 2011, pp. 1441–1444.
[18] B. Wu, S. Fan, "Improved Artificial Bee Colony Algorithm with Chaos", in: Y. Yu, Z. Yu, J. Zhao (Eds.): Computer Science for Environmental Engineering and EcoInformatics, Part I, Communications in Computer and Information Science, vol. 158, 2011, pp. 51-56.
[19] L. Fenglei, D. Haijun, F. Xing, "The parameter improvement of bee colony algorithm in TSP problem", Science Paper Online, November 2007.
[20] G. Zhu, S. Kwong, "Gbest-guided artificial bee colony algorithm for numerical function optimization", Applied Mathematics & Computation 217 (7) (2010) 3166–3173.
[21] F. Kang, J. Li, Z. Ma, H. Li, "Artificial bee colony algorithm with local search for numerical optimization", Journal of Software 6 (3) (2011) 490–497.
[22] F. Qingxian, D. Haijun, "Bee colony algorithm for the function optimization", Science Paper Online, 2008.
[23] H. Quan, X. Shi, "On the analysis of performance of the improved ABC algorithm", in: 4th IEEE Int. Conf. Natural Computation (ICNC), 2008, pp. 654–658.
[24] E. Montes, R. Koeppel, "Elitist artificial bee colony for constrained real-parameter optimization", IEEE Congress on Evolutionary Computation 11 (2010), pp. 1–8.
[25] S. Nieberg, H. Beyer, "Self-adaptation in evolutionary algorithms", Parameter Setting in Evolutionary Algorithm (2007) 47–76.
[26] J. Liang, A. Qin, P. Suganthan, S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions", IEEE Trans. on Evolutionary Comput. 10 (3) (2006) 281 295.
[27] C. Lee, X. Yao, "Evolutionary programming using mutations based on the Lévy probability distribution", IEEE Transactions on Evolutionary Computation 8 (1) (2004) 1–13.
[28] X. Yao, Y. Liu, G. Lin, "Evolutionary programming made faster", IEEE Transactions on Evolutionary Computation 3 (2) (1999) 82–102.