Taylor & Francis
Taylor & Francis Group

Optimization

A Journal of Mathematical Programming and Operations Research

ISSN: 0233-1934 (Print) 1029-4945 (Online) Journal homepage: http://www.tandfonline.com/loi/gopt20

Self-adaptive artificial bee colony

Jagdish Chand Bansal, Harish Sharma, K.V. Arya, Kusum Deep & Millie Pant

To cite this article: Jagdish Chand Bansal, Harish Sharma, K.V. Arya, Kusum Deep & Millie Pant (2014) Self-adaptive artificial bee colony, Optimization, 63:10, 1513-1532, DOI: 10.1080/02331934.2014.917302

To link to this article: http://dx.doi.org/10.1080/02331934.2014.917302

Optimization, 2014
Vol. 63, No. 10, 1513–1532, http://dx.doi.org/10.1080/02331934.2014.917302

Taylor & Francis
Taylor & Francis Group

Self-adaptive artificial bee colony

Jagdish Chand Bansal[a], Harish Sharma[b*], K.V. Arya[c], Kusum Deep[d] and Millie Pant[d]

[a]*Department of Mathematics, South Asian University, New Delhi, India;* [b]*Computer Engineering, Vardhaman Mahaveer Open University, Kota, India;* [c]*ABV–Indian Institute of Information Technology and Management, Gwalior, India;* [d]*Department of Mathematics, Indian Institute of Technology, Roorkee, India*

(*Received 30 April 2013; accepted 26 January 2014*)

Artificial Bee Colony (ABC) optimization algorithm is a swarm intelligence-based nature inspired algorithm, which has been proved a competitive algorithm with some popular nature-inspired algorithms. ABC has been found to be more efficient in exploration as compared to exploitation. With a motivation to balance exploration and exploitation capabilities of ABC, this paper presents an adaptive version of ABC. In this adaptive version, step size in solution modification and ABC parameter 'limit' are set adaptively based on current fitness values. In the present self-adaptive ABC, good solutions are appointed to exploit the search region in their neighbourhood, while worse solutions are appointed to explore the search region. The better solutions are given higher chances to update themselves with the help of parameter 'limit', which changes adaptively in the present study. The experiments on 16 unbiased test problems of different complexities show that the proposed strategy outperforms the basic ABC and some recent variants of ABC.

Keywords: numerical optimization; swarm intelligence; artificial bee colony; exploration-exploitation

1. Introduction

Swarm intelligence has become an emerging and interesting area in the field of nature-inspired techniques during the past decade. It is based on the collective behaviour of social creatures. Optimization algorithms based on swarm intelligence find solution by collaborative trial and error process. Social creatures utilize their ability of social learning to solve complex tasks. Behaviour of learning from each other in social colonies is the main inspiration behind the development of many efficient swarm-based optimization algorithms. Researchers have analysed such behaviours and designed algorithms that can be used to solve nonlinear, nonconvex or discrete optimization problems. Previous research [1–5] has shown that algorithms based on swarm intelligence have great potential to find solutions to real world optimization problems. The algorithms imitating the social behaviour of species include ant colony optimization,[1] particle swarm optimization (PSO),[2] bacterial foraging optimization (BFO) [6], etc.

Artificial bee colony (ABC) optimization algorithm introduced by Karaboga [7] is a recent addition in this category. This algorithm is inspired by the behaviour of honey

*Corresponding author. Email: hsharma@vmou.ac.in

bees when seeking a quality food source. Like any other population-based optimization algorithm, ABC consists of a population of possible solutions. The possible solutions are food sources of honey bees. The fitness is determined in terms of the quality (nectar amount) of the food source. ABC is a relatively simple, fast and population-based stochastic search technique in the field of nature-inspired algorithms.

There are two fundamental processes which drive the swarm to update in ABC: the variation process, which enables exploring different areas of the search space, and the selection process, which ensures the exploitation of the previous experience. However, it has been shown that the ABC may occasionally stop proceeding towards the global optimum even though the population has not converged to a local optimum.[8] It can be observed that the solution search equation of ABC algorithm is good at exploration but poor at exploitation.[9] Therefore, to maintain the proper balance between exploration and exploitation behaviour of ABC, it is necessary to develop a strategy in which better solutions exploit the search space in close proximity, while less fit solutions explore the search space. Therefore, in this paper, we proposed a self-adaptive step size strategy to update a solution. In the proposed strategy, a solution takes small step sizes in position updating process if its fitness is high, i.e. it searches the solution in its vicinity, whereas a solution takes large step sizes if its fitness is low, and hence explore the search space. The proposed strategy is used for finding the global optima of a unimodal and/or multimodel functions by adaptively modifying the step sizes in updating process of the candidate solution in the search space within which the optima is known to exist. In the proposed strategy, ABC algorithm's parameter 'limit' is modified self-adaptively based on the fitness of the solution. Now, there is separate 'limit' for every solution according to the fitness. For high fitness solutions, value of 'limit' is high, while for less fit solutions, it is low. Hence, a better solution has more time to update itself in comparison to the less fit solutions. Further, to improve the diversity of the algorithm, the number of scout bees is increased. The proposed strategy is compared with ABC and its recent variants, named, Gbest-guided ABC algorithm (GABC),[9] Best-So-Far Artificial Bee Colony (BSFABC) [10] and Modified Artificial Bee Colony (MABC).[11] Finally, the performance of the proposed strategy is evaluated by solving the clustering problem on well-known benchmark data-sets and the results are compared with the basic ABC.

The rest of the paper is organized as follows: Basic ABC is explained in Section 2. Section 3 describes a brief review on recent modifications in ABC. In Section 4, self-adaptive ABC is proposed and explained. In Section 5, performance of the proposed strategy is analysed. Finally, in Section 6, paper is concluded.

2. ABC algorithm

In ABC, honey bees are classified into three groups, namely employed bees, onlooker bees and scout bees. The number of employed bees is equal to the number of onlooker bees. The employed bees are the bees which searches the food source and gather the information about the quality of the food source. Onlooker bees stay in the hive and search the food sources on the basis of the information gathered by the employed bees. The scout bee searches new food sources randomly in places of the abandoned foods sources. Similar to the other population-based algorithms, ABC solution search process is an iterative process. After, initialization of the ABC parameters and swarm, it requires the repetitive iterations of the three phases namely employed bee phase, onlooker bee phase and scout bee phase. Each phase is described as follows:

2.1. *Initialization of the swarm*

The parameters for the ABC are the number of food sources, the number of trials after which a food source is considered to be abandoned and the termination criteria. In the basic ABC, the number of food sources are equal to the employed bees or onlooker bees. Initially, a uniformly distributed initial swarm of SN food sources, where each food source $x_i (i = 1, 2, \ldots, SN)$ is a D-dimensional vector, is generated. Here, D is the number of variables in the optimization problem and x_i represents the ith food source in the swarm. Each food source is generated as follows:

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

here $x_{\min j}$ and $x_{\max j}$ are bounds of x_i in jth direction and rand[0, 1] is a uniformly distributed random number in the range [0, 1].

2.2. *Employed bee phase*

In the employed bee phase, modification in the current solution (food source) is done by employed bees according to the information of individual experience and the new solution fitness value. If the fitness value of the new solution is greater than that of the old solution, then the bee updates her position to the new solution and old one is discarded. The position update equation for ith candidate in this phase is

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

here $k \in \{1, 2, \ldots, SN\}$ and $j \in \{1, 2, \ldots, D\}$ are randomly chosen indices. k must be different from i. ϕ_{ij} is a random number between $[-1, 1]$.

2.3. *Onlooker bees phase*

In this phase, the new fitness information (nectar) of the new solutions (food sources) and their position information is shared by all the employed bees with the onlooker bees in the hive. Onlooker bees analyse the available information and selects a solution with a probability $prob_i$ related to its fitness, which can be calculated using following expression (there may be some other but must be a function of fitness):

$$prob_i(G) = \frac{0.9 \times fitness_i}{maxfit} + 0.1, \tag{3}$$

here $fitness_i$ is the fitness value of the ith solution and $maxfit$ is the maximum fitness of the solutions. As in the case of employed bee, it produces a modification on the position in its memory and checks the fitness of the new solution. If the fitness is higher than the previous one, the bee memorizes the new position and forgets the old one.

2.4. *Scout bees phase*

A food source is considered to be abandoned if its position is not getting updated during a predetermined number of cycles. In this phase, the bee whose food source has been abandoned becomes scout bee and the abandoned food source is replaced by a randomly

chosen food source within the search space. In ABC, predetermined number of cycles is a crucial control parameter which is called $limit$ for abandonment.

Assume that the abandoned source is x_i. The scout bee replaces this food source by a randomly chosen food source, which is generated as follows:

$$x_{ij} = x_{minj} + \text{rand}[0, 1](x_{maxj} - x_{minj}), \quad \text{for } j \in \{1, 2, \ldots, D\} \tag{4}$$

where x_{minj} and x_{maxj} are bounds of x_i in j^{th} direction.

2.5. *Main steps of the ABC algorithm*

Based on the above explanation, it is clear that the ABC search process contains three important control parameters: The number of food sources SN (equal to number of onlooker or employed bees), the value of $limit$ and the maximum number of iterations. The pseudo-code of the ABC is shown in Algorithm 1.[8]

Algorithm 1 ABC Algorithm:

Initialize the parameters;
while Termination criteria is not satisfied **do**
Step 1 Employed bee phase for generating new food sources;
Step 2 Onlooker bees phase for updating the food sources depending on their nectar amounts;
Step 3 Scout bee phase for discovering the new food sources in place of abandoned food sources;
Step 4 Memorize the best food source found so far;
end while
Output the best solution found so far.

3. Brief review on basic improvements in ABC

In order to get rid of the drawbacks of basic ABC, researchers have improved ABC in many ways. The potentials where ABC can be improved may be broadly classified into two categories:

- Fine tuning of ABC control parameters $SN, \phi_{ij}, limit$ or introducing new control parameters.
- Hybridization of ABC with other population-based probabilistic or deterministic algorithms.

Karaboga [7] observed that the value of ϕ_{ij} should be in the range of $[-1, 1]$. The value of $limit$ should be $SN \times D$, where SN is the number of solutions and D is the dimension of the problem.

Wei-feng Gao and Liu [12] proposed an improved solution search equation in ABC, which is based on the fact that to improve the exploitation the bee searches only around the best solution of the previous iteration. Banharnsakun et al. [10] introduced a new variant of ABC namely the best-so-far selection in ABC algorithm. In that modified ABC (MABC), they introduced three major changes in ABC: The best-so-far method, an adjustable search radius and an objective-value-based comparison method.

Qingxian and Haijun proposed two modifications in the ABC: one, in the initialization scheme, by making the initial group symmetrical and second, in the solution selection scheme, by employing the Boltzmann selection mechanism instead of roulette wheel selection.[13]

Tsai et al. [14] introduced the Newtonian law of universal gravitation in the onlooker phase of the basic ABC algorithm, in which onlookers are selected based on a roulette wheel to maximize the exploitation capacity of the solutions in this phase and the strategy is named as Interactive ABC (IABC). Baykasoglu et al. [15] incorporated the ABC algorithm with shift neighbourhood searches and greedy randomized adaptive search heuristic and applied it to the generalized assignment problem.

Furthermore, a modified version of the ABC algorithm is introduced and applied for efficiently solving real-parameter optimization problems by Akay and Karaboga [11]. In the proposed work, effects of the perturbation rate that controls the frequency of parameter change, the scaling factor (step size) that determines the magnitude of change in parameters, while producing a neighbouring solution and the $limit$ parameter on the performance of the ABC algorithm are investigated on real-parameter optimization. Karaboga and Akay also modified the ABC to solve constrained optimization problems.[16] They used Deb's rules [17] consisting of three simple heuristic rules and a probabilistic selection scheme for constraint handling in ABC algorithm.

Zhu and Kwong [9] proposed a variant of ABC for numerical function optimization inspired by the solution search strategy of PSO and named it Gbest-guided ABC (GABC). In GABC, the knowledge of global best (gbest) solution is incorporated in the basic ABC solution search mechanism to enhance the exploitation.

Researchers are also developing self-adaptive strategies for ABC to balance the convergence and diversity capability of the swarm. Alam et al. [18] proposed a self-adaptation of mutation step size in ABC solution update process. In their work, authors introduced exponentially distributed mutation strategy in ABC, which produces mutation steps with varying lengths and suitably adjusts the current step length.

El-Abd introduced a cooperative approach to ABC algorithm (CABC).[19] The strategy is based on the explicit space decomposition approach. In explicitly decomposing strategy, a set of algorithm instances explore the search space to find out an optimum solution of the problem. The algorithm's outcome is the concatenated solution of the best solutions, provided by these algorithm instances.

Further, Kang et al. [20] described a Rosenbrock ABC (RABC) that combines Rosenbrock's rotational direction method with ABC for accurate numerical optimization. In RABC, exploitation phase is introduced in the ABC using Rosenbrock's rotational direction method.

4. Self-adaptive ABC

Exploration and exploitation are the two important characteristics of the population-based optimization algorithms such as GA [21], PSO [2], DE [22], BFO [6] and so on. In these optimization algorithms, the exploration represents the ability to discover the global optimum by investigating the various unknown regions in the solution search space, while the exploitation represents the ability to find better solutions by implementing the knowledge of the previous good solutions. In behaviour, the exploration and exploitation contradict with each other. But both abilities should be well balanced to achieve better optimization

performance. Karaboga and Akay [8] tested different variants of ABC for global optimization and found that the ABC shows poor performance and remains inefficient in exploring the search space. In ABC, any potential solution updates itself using the information provided by a randomly selected potential solution within the current swarm. In this process, a step size, which is a linear combination of a random number $\phi_{ij} \in [-1, 1]$, current solution and a randomly selected solution are used. Now, the quality of the updated solution highly depends upon this step size. If the step size is too large, which may occur if the difference of current solution and randomly selected solution is large with high absolute value of ϕ_{ij}, then updated solution can surpass the true solution and if this step size is too small, then the convergence rate of ABC may significantly decrease. A proper balance of this step size can balance the exploration and exploitation capabilities of the ABC, simultaneously. Since this step size consists of random components so the balance cannot be done manually. Therefore, to balance the exploration and exploitation, we modified the solution update strategy and the parameter 'limit' according to the fitness of the solution. In the basic ABC, the food sources are updated, as shown in Equation (2). Inspired by PSO [2] and Gbest-guided ABC (GABC) [9] algorithms which, in order to improve the exploitation, take advantage of the information of the gbest solution to guide the search of candidate solutions, the solution search equation described by Equation (2) is modified as follows [9]:

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) + \psi_{ij}(x_{bestj} - x_{ij}),$$

here, ψ_{ij} is a uniform random number in $[0, C_2]$, where C_2 is a nonnegative constant. We call C_2 an acceleration parameter. In this paper, to balance the diversity and convergence abilities of ABC, three modifications are proposed:

(1) To enhance the exploitation capability of ABC, self-adaptive step size mechanism is introduced. In the proposed strategy, step sizes are based on the fitness of the solution. The strategy is based on the concept that a high fit solution should search in its neighbourhood to exploit the search space, while a low fit solution should explore the search area. In this way, the situation of skipping true solution can be avoided, while maintaining the speed of convergence. Therefore, the solution update strategy in employed bee phase is modified and for a solution x_i and it is shown in Algorithm 2:

Algorithm 2 Solution update in employed bee phase:

Input: solution x_i, $prob_i$ and $j \in (1, D)$;
if $U(0, 1) < prob_i$ **then**
  $\phi_{ij} = U(-R, R) \times (C_1 - prob_i)$;
  $\psi_{ij} = U(0, (C_1 - prob_i))$;
else
  $\phi_{ij} = U(-1, 1)$ (same as in ABC);
  $\psi_{ij} = U(0, C_2)$;
end if
$x'_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) + \psi_{ij}(x_{bestj} - x_{ij})$;

In Algorithm 2, $prob_i$ is a function of fitness and calculated as shown in Equation (3), while C_1 and C_2 are constants. It is clear from this algorithm that for a solution, if value of $prob_i$ is high and that is the case of high fit solution then for that solution the step size will be small as both the factor ϕ_{ij} and ψ_{ij} will be low. Therefore, it is obvious that there is more chance for the high fit solution to update in its neighbourhood compared with the low fit solution and hence, a better solution could exploit the search area in its vicinity. In other words, we can say that solutions exploit or explore the search area based on a probability, which is the function of fitness.

(2) Further, the solution update strategy for onlooker bees phase is also modified and is shown in Algorithm 3.

Algorithm 3 Solution update in Onlooker bee phase:

Input: solution x_i, $prob_i$, $j_1 \in (1, D)$ and $j_2 \in (1, D)$ where $j_1 \neq j_2$;
for $j \in \{j_1, j_2\}$ **do**
  if $U(0, 1) < \epsilon \times prob_i$ **then**
    $\phi_{ij} = U(-R, R) \times (C_1 - prob_i)$;
    $\psi_{ij} = U(0, (C_1 - prob_i))$;
  else
    $\phi_{ij} = U(-1, 1)$;(same as in ABC)
    $\psi_{ij} = U(0, C_2)$;
  end if
  $x'_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) + \psi_{ij}(x_{bestj} - x_{ij})$;
end for

We know that in onlooker bee phase, better solutions get more chance to update themselves. Therefore, it is expected that updates in two components instead of the traditional one can make the algorithm converge faster to the global optima. In Algorithm 3, the value of $prob_i$ is reduced by multiplying it by ϵ (where $\epsilon \in [0.1, 0.9]$) in the expectation that only very high fit solutions get more chance to exploit the area in their neighbourhood.

(3) To enhance the exploration capability, the number of scout bees is increased. This modification avoids the situation of stagnation of the algorithm. Therefore, in this paper, all the bees which crosses the 'limit' are treated as scout bees. However, only if this modification has been done in the basic ABC, then it may make ABC relatively less stable as the scout bees do not use the previous knowledge for generating new food solutions and hence previous learning would have been lost. But in the proposed strategy, the first and the second modifications use the experience of gbest-guided solution and give more chance for better solutions to update themselves, and hence improve the convergence. Therefore, in the first two modifications, better solutions get more chance in search process and minimize the threat of less stability, while taking advantage of third modification in exploration of the search space.

Further, in the basic ABC, the parameter 'limit' is fixed and calculated as '$limit = D \times SN$'. But, in the proposed strategy, the parameter 'limit' is self-adaptive and is a function of fitness. Algorithm 4 shows the self-adaptive strategy for calculating 'limit'. It

Algorithm 4 Setting of parameter *limit*:

```
Input: prob_i where i ∈ (1, SN);
for i = 1 to SN do
  limit_i = D × SN × prob_i;
  if limit_i < D × SN/γ then
    limit_i = D × SN/γ;
  end if
end for
```

is clear from this algorithm that the range for parameter $limit \in (D \times SN/\gamma, D \times SN)$. Here, the lower boundary, $D \times SN/\gamma$, is set to ensure that the solutions having poor fitness should also get proper time to update themselves.

4.1. *Discussion*

It is expected from a good search process that it should explore new solutions, while maintaining satisfactory performance by exploiting existing solutions.[23] These three modifications form an attempt to design a good search algorithm. Step size always plays an important role for better exploration and exploitation of the search space. It is obvious from the first two modifications that rich solutions (in terms of fitness) take small size steps and usually produces small variations that are better for exploitation around the already found solutions, while the poor solutions take large step sizes, which is likely to produce large variations to facilitate better exploration of the undiscovered regions of the search space. Further, the third modification is proposed for exploring the new solutions in the search space. In the basic ABC, scout bee is responsible for the exploration and it is the bee having a 'trial counter' higher than the 'limit' and maximum in the colony. In the third modification, we increased the number of scout bees as well as made the 'limit' a function of fitness. This modification is expected to improve the exploration with the threat of instability, which may have occurred by loosing the previous experience. This threat is moderated by incorporating the first two modifications in basic ABC to improve the exploitation and by setting up the lower boundary for the 'limit'. Since, in the proposed strategy, the step sizes and $limit$ are adaptively changing based on the fitness of the solutions, the strategy is named 'Self Adaptive Artificial Bee Colony' (SAABC) algorithm.

5. Experimental results and discussion

5.1. *Test problems under consideration*

In order to analyse the performance of $SAABC$, 16 unbiased optimization problems (solutions do not exist on axis, diagonal or origin) (f_1 to f_{16}) are selected (listed in Table 1). These are continuous optimization problems and have different degrees of complexity and multimodality. Test problems f_1 to f_3 and f_{10} to f_{16} are taken from [24] and test problems f_4 to f_9 are taken from [25] with the associated offset values.

Table 1. Test problems.

TP	Objective function	Search Range	Optimum value	D	AE
Beale function	$f_1(x) = [1.5-x_1(1-x_2)]^2+[2.25-x_1(1-x_2^2)]^2+[2.625-x_1(1-x_2^3)]^2$	$[-4.5,4.5]$	$f(3, 0.5) = 0$	2	$1.0E-05$
Colville function	$f_2(x) = 100[x_2 - x_1^2]^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1)$	$[-10,10]$	$f(\vec{1}) = 0$	4	$1.0E-05$
Kowalik function	$f_3(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2+b_i x_2)}{b_i^2+b_i x_3+x_4}\right]^2$	$[-5,5]$	$f(0.1928,\ 0.1908, 0.1231, 0.1357) = 3.07E-04$	4	$1.0E-05$
Shifted Rosenbrock	$f_4(x) = \sum_{i=1}^{D-1}(100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2) + f_{bias}, z = x - o + 1, x = [x_1, x_2, \ldots, x_D], o = [o_1, o_2, \ldots, o_D]$	$[-100,100]$	$f(o) = f_{bias} = 390$	10	$1.0E-01$
Shifted Sphere	$f_5(x) = \sum_{i=1}^{D} z_i^2 + f_{bias}, z = x - o, x = [x_1, x_2, \ldots, x_D], o = [o_1, o_2, \ldots, o_D]$	$[-100,100]$	$f(o) = f_{bias} = -450$	10	$1.0E-05$
Shifted Rastrigin	$f_6(x) = \sum_{i=1}^{D}(z_i^2 - 10\cos(2\pi z_i) + 10) + f_{bias}$, z $= (x - o)$, x $= (x_1, x_2, \ldots, x_D)$, o $= (o_1, o_2, \ldots, o_D)$	$[-5,5]$	$f(o) = f_{bias} = -330$	10	$1.0E-02$
Shifted Schwefel	$f_7(x) = \sum_{i=1}^{D}(\sum_{j=1}^{i} z_j)^2 + f_{bias}, z = x - o, x = [x_1, x_2, \ldots, x_D], o = [o_1, o_2, \ldots, o_D]$	$[-100,100]$	$f(o) = f_{bias} = -450$	10	$1.0E-05$
Shifted Griewank	$f_8(x) = \sum_{i=1}^{D}\frac{z_i^2}{4000} - \prod_{i=1}^{D}\cos(\frac{z_i}{\sqrt{i}}) + 1 + f_{bias}, z = (x - o), x = [x_1, x_2, \ldots, x_D], o = [o_1, o_2, \ldots, o_D]$	$[-600,600]$	$f(o) = f_{bias} = -180$	10	$1.0E-05$
Shifted Ackley	$f_9(x) = -20\exp\left(\text{-}0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}\right) - \exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)) + 20+e+f_{bias}, z = (x-o), x = (x_1, x_2, \ldots, x_D), o = (o_1, o_2, \ldots, o_D)$	$[-32,32]$	$f(o) = f_{bias} = -140$	10	$1.0E-05$
Goldstein-Price	$f_{10}(x) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2))(30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2))$	$[-2,2]$	$f(0, -1) = 3$	2	$1.0E-14$
Easom's function	$f_{11}(x) = -\cos x_1 \cos x_2 e^{((-(x_1-\Pi)^2-(x_2-\Pi)^2))}$	$[-10,10]$	$f(\pi, \pi) = -1$	2	$1.0E-13$

(Continued)

Table 1. (*Continued*).

TP	Objective function	Search Range	Optimum Value	D	AE
Dekkers and Aarts	$f_{12}(x) = 10^5 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4$	[-20,20]	$f(0, 15) = f(0, -15) = -24777$	2	$5.0E-01$
McCormick	$f_{13}(x) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - \frac{3}{2}x_1 + \frac{5}{2}x_2 + 1$	$-1.5 \leq x_1 \leq 4, -3 \leq x_2 \leq 3$	$f(-0.547, -1.547) = -1.9133$	30	$1.0E-04$
Meyer and Roth	$f_{14}(x) = \sum_{i=1}^{5}(\frac{x_1 x_3 t_i}{1+x_1 t_i + x_2 v_i} - y_i)^2$	$[-10,10]$	$f(3.13, 15.16, 0.78) = 0.4E-04$	3	$1.0E-03$
Shubert	$f_{15}(x) = -\sum_{i=1}^{5} i\cos((i+1)x_1+1)\sum_{i=1}^{5} i\cos((i+1)x_2+1)$	$[-10,10]$	$f(7.0835, 4.8580) = -186.7309$	2	$1.0E-05$
Sinusoidal	$f_{16}(x) = -[A\prod_{i=1}^{D}\sin(x_i - z) + \prod_{i=1}^{D}\sin(B(x_i - z))]$, $A = 2.5$, $B = 5$, $z = 30$	[0,180]	$f(90\vec{+}z) = -(A+1)$	10	$1.0E-02$

Note: TP: test problem, AE: acceptable error.

5.2. *Experimental setting*

In algorithms 2 and 3, $U(-R, R)$ represents a uniform random number in the range $(-R, R)$. The sensitivity analysis for this constant R is carried out in Figure 1(a) and found that the suitable range for SAABC is $[-0.5, 0.5]$. Further, the effect of constant C_1 described by algorithms 2 and 3 on the performance of SAABC is investigated. Its sensitivity with respect to different values of C_1 in the range $[1.1, 2.5]$, is examined in the Figure 1(b). It can be observed from this figure that SAABC is very sensitive towards C_1, and value 1.1 gives comparatively better results. A sensitivity analysis for the constant ϵ (refer algorithm 3) and constant γ (refer algorithm 4) is also carried out in Figure 1(c) and (d), respectively. It is observed from these figures that the SAABC performs better for $\epsilon = 0.5$ and $\gamma = 4$. Therefore, $C_1 = 1.1$, $\epsilon = 0.5$ and $\gamma = 4$ are selected for the experiments in this paper.

Further, Figure 1(e) shows the variation of the step size in the proposed strategy, with respect to the $prob_i$ at an iteration of a randomly selected solution for function f_{11}. It is clear from this figure that the step size is generally low for the high value of $prob_i$ and vice versa. The variation in number of scout bees with respect to iterations is shown in Figure 1(f) for function f_{11}. Figure 1(g) and (h) show a variation in the parameter 'limit' of SAABC and $prob_i$ with respect to iterations for function f_7. Figure 1(g) shows this variations in the range of 63 to 250 as for function f_7, $D = 10$ and $SN = 25$, while Figure 1(h) shows this variation along with $prob_i$ with respect to iterations.

To prove the efficiency of SAABC, it is compared with ABC and recent variants of ABC, namely GABC,[9] Best-So-Far ABC (BSFABC) [10] and MABC.[11] To test SAABC, ABC, GABC, BSFABC and MABC over considered problems, following experimental setting is adopted:

- Colony size $NP = 50$,[26,27]
- $\phi_{ij} = \text{rand}[-1, 1]$,
- Number of food sources $SN = NP/2$,
- $limit = D \times SN$,[11,28]
- The stopping criteria is either maximum number of function evaluations (which is set to be 200000) is reached or the acceptable error (mentioned in Table 1) has been achieved,
- The number of simulations/run = 100,
- $C_2 = 1.5$,[9]
- $C_1 = 1.1$, (refer Figure 1(b))
- Parameter settings for the algorithms GABC, BSFABC and MABC are similar to their original research papers.

5.3. *Results comparison*

Numerical results with experimental setting of Section 5.2 are given in Table 2. In Table 2, standard deviation (SD), mean error (ME), average number of function evaluations (AFE) and success rate (SR) are reported. Table 2 shows that most of the time SAABC outperforms in terms of reliability, efficiency and accuracy as compared to the basic ABC, GABC, BSFABC and MABC. Some more intensive analyses based on acceleration rate (AR),[29] performance indices and boxplots have also been carried out for results of ABC and its variants.

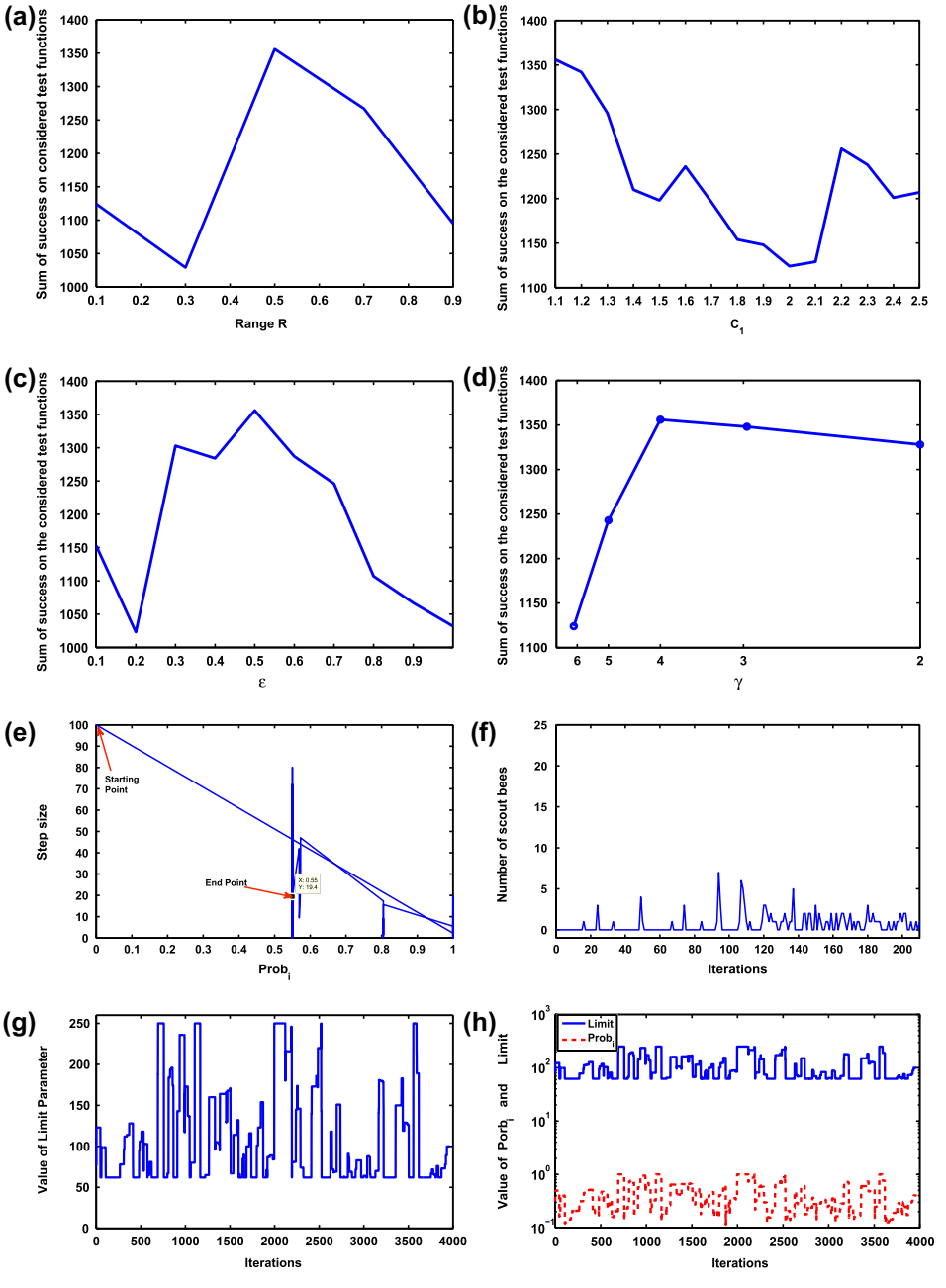

Figure 1. (a) Effect of range $U(-R, R)$ on sum of successes of all test functions, (b) Effect of parameter C_1 on sum of successes of all test functions, (c) Effect of ϵ on sum of successes of all test functions, (d) Effect of γ on sum of successes of all test functions, (e) Step size v/s $prob_i$ for f_{11}, (f) Number of scout bees with respect to iterations for f_{11}, (g) & (h) Variation in parameter 'limit' according to $prob_i$ with respect to iterations for f_7.

Table 2. Comparison of the results of test problems.

TP	Algorithm	SD	ME	AFE	SR
	ABC	1.66E−06	8.64E−06	16520.09	100
	GABC	3.05E−06	5.03E−06	9314.71	100
f_1	BSFABC	5.64E−05	1.98E−05	47522.01	95
	MABC	2.68E−06	5.47E−06	10350.53	100
	SAABC	2.99E−06	5.67E−06	1815.59	100
	ABC	1.03E−01	1.67E−01	199254.48	1
	GABC	1.71E−02	1.95E−02	151300.35	46
f_2	BSFABC	1.61E−02	1.86E−02	153393.46	47
	MABC	8.26E−03	1.25E−02	147787.15	52
	SAABC	1.77E−03	8.53E−03	12817.47	100
	ABC	7.33E−05	1.76E−04	180578.91	18
	GABC	2.15E−05	8.68E−05	90834.53	97
f_3	BSFABC	7.57E−05	1.41E−04	147931.24	50
	MABC	8.02E−05	2.02E−04	187320.13	13
	SAABC	3.26E−05	9.72E−05	43669.72	98
	ABC	1.05E+00	6.36E−01	176098.02	23
	GABC	1.60E−02	8.45E−02	99219.48	99
f_4	BSFABC	3.79E+00	2.34E+00	179970.99	19
	MABC	9.19E−01	6.99E−01	180961.73	23
	SAABC	2.01E+00	7.93E−01	115099.24	72
	ABC	2.42E−06	7.16E−06	9013.5	100
	GABC	2.08E−06	6.83E−06	5585.5	100
f_5	BSFABC	2.18E−06	7.44E−06	18122	100
	MABC	1.61E−06	8.23E−06	8702	100
	SAABC	1.81E−06	8.05E−06	4868.5	100
	ABC	1.21E+01	8.91E+01	200011.71	0
	GABC	9.24E+00	8.56E+01	200006.8	0
f_6	BSFABC	1.77E+01	1.20E+02	200036.53	0
	MABC	1.15E+01	8.00E+01	200015.14	0
	SAABC	1.08E+01	8.20E+01	200023.12	0
	ABC	3.54E+03	1.11E+04	200029.02	0
	GABC	3.00E+03	1.08E+04	200016.04	0
f_7	BSFABC	7.33E+03	2.70E+04	200035.12	0
	MABC	2.76E+03	1.03E+04	200015.92	0
	SAABC	2.18E+03	8.45E+03	200024.75	0
	ABC	2.21E−03	6.95E−04	61650.9	90
	GABC	7.35E−04	7.88E−05	38328.96	99
f_8	BSFABC	6.34E−03	4.76E−03	115441.96	58
	MABC	2.21E−03	6.24E−04	85853.52	92
	SAABC	2.86E−03	1.14E−03	102258.18	86
	ABC	1.80E−06	7.90E−06	16767	100
	GABC	1.37E−06	8.31E−06	9366	100
f_9	BSFABC	1.35E−06	8.39E−06	31224	100
	MABC	9.96E−07	8.93E−06	14189.06	100
	SAABC	1.52E−06	8.57E−06	19051.5	100

(Continued)

Table 2. (*Continued*).

TP	Algorithm	SD	ME	AFE	SR
	ABC	1.66E−06	8.64E−06	16520.09	100
	ABC	5.16E−06	1.04E−06	109879.46	62
f_1	GABC	4.37E−15	4.87E−15	3956.05	100
f_{10}	BSFABC	4.90E−15	6.62E−15	14031.79	100
	MABC	4.11E−15	4.73E−15	14228.59	100
	SAABC	4.75E−15	5.68E−15	3712.08	100
	ABC	4.44E−05	1.60E−05	181447.91	17
	GABC	2.79E−14	4.02E−14	46909.7	100
f_{11}	BSFABC	3.03E−14	3.71E−14	4880.22	100
	MABC	1.45E−03	6.64E−04	199872.75	1
	SAABC	2.98E−14	5.09E−14	11411.95	100
	ABC	5.33E−03	4.91E−01	1460.56	100
	GABC	5.40E−03	4.90E−01	792	100
f_{12}	BSFABC	5.82E−03	4.90E−01	2802.92	100
	MABC	5.74E−03	4.91E−01	2370.5	100
	SAABC	5.69E−03	4.90E−01	674.01	100
	ABC	6.67E−06	8.92E−05	1166.5	100
	GABC	6.45E−06	8.79E−05	622	100
f_{13}	BSFABC	6.34E−06	8.91E−05	958.51	100
	MABC	6.15E−06	8.95E−05	1702.28	100
	SAABC	6.84E−06	8.88E−05	592.22	100
	ABC	2.89E−06	1.94E−03	24476.88	100
	GABC	2.74E−06	1.95E−03	5127.73	100
f_{14}	BSFABC	2.98E−06	1.94E−03	15703.99	100
	MABC	2.79E−06	1.95E−03	9019.7	100
	SAABC	2.89E−06	1.95E−03	2037.28	100
	ABC	5.34E−06	4.86E−06	4752.21	100
	GABC	5.72E−06	5.07E−06	2550.57	100
f_{15}	BSFABC	5.94E−06	5.27E−06	9036.83	100
	MABC	5.60E−06	4.83E−06	33268.91	100
	SAABC	5.82E−06	5.20E−06	2738.25	100
	ABC	1.83E−03	7.77E−03	54159.26	99
	GABC	2.09E−03	7.87E−03	49230.85	100
f_{16}	BSFABC	2.08E−03	7.65E−03	66317.6	100
	MABC	1.03E−01	6.44E−01	200035.08	0
	SAABC	1.95E−03	7.74E−03	41851.58	100

Note: TP: test problem.

SAABC, ABC, GABC, BSFABC and MABC are compared based on SR, ME and AFE mentioned in Table 2. First SR is compared for all these algorithms and if it is not possible to distinguish the algorithms based on SR, then comparison is made on the basis of AFE. ME is used for comparison if it is not possible to compare the algorithms on the basis of SR and AFE both. The outcome of this comparison is summarized in Table 3. In Table 3, '+' indicates that SAABC is better than the considered algorithms and '−' indicates that the

Table 3. Summary of Table 2 outcome.

Function	SAABC Vs. ABC	SAABC Vs. GABC	SAABC Vs. BSFABC	SAABC Vs. MABC
f_1	+	+	+	+
f_2	+	+	+	+
f_3	+	+	+	+
f_4	+	−	+	+
f_5	+	+	+	+
f_6	+	−	+	+
f_7	+	+	+	+
f_8	−	−	+	−
f_9	−	−	+	−
f_{10}	+	+	+	+
f_{11}	+	+	−	+
f_{12}	+	+	+	+
f_{13}	+	+	+	+
f_{14}	+	+	+	+
f_{15}	+	−	+	+
f_{16}	+	+	+	+
Total number of + sign	16	11	15	14

algorithm is not better or the difference is very small. The last row of Table 3, establishes the superiority of SAABC over ABC, GABC, BSFABC and MABC over most of the considered test functions.

Further, the convergence characteristics of the considered ABCs are graphically presented in Figure 2. It is clear from this figure that SAABC converges fast to the global optima for functions f_1, f_2, f_3 and f_4, while for f_5 and f_8, ABC and GABC converge fast, respectively.

We also compare the convergence speed of the considered algorithms by measuring the AFEs. A smaller AFEs means higher convergence speed. In order to minimize the effect of the stochastic nature of the algorithms, the reported function evaluations for each test problem are averaged over 100 runs. In order to compare convergence speeds, we use the AR [29] which is defined as follows, based on the AFEs for the two algorithms $ALGO$ and $SAABC$:

$$\mathrm{AR} = \frac{\mathrm{AFE}_{\mathrm{ALGO}}}{\mathrm{AFE}_{\mathrm{SAABC}}} \tag{5}$$

where ALGO $\in$ {ABC, GABC, BSFABC, MABC} and $AR > 1$ means that SAABC is faster than the ALGO. In order to investigate the AR of the proposed algorithm, as compared to the basic ABC and its variants, results of Table 2 are analysed and the value of AR is calculated using Equation (5). Table 4 shows a clear comparison between SAABC and ABC, SAABC and GABC, SAABC and BSFABC, and SAABC and MABC in terms of AR. It is clear from Table 4 that the convergence speed of SAABC is faster among all the considered algorithms.

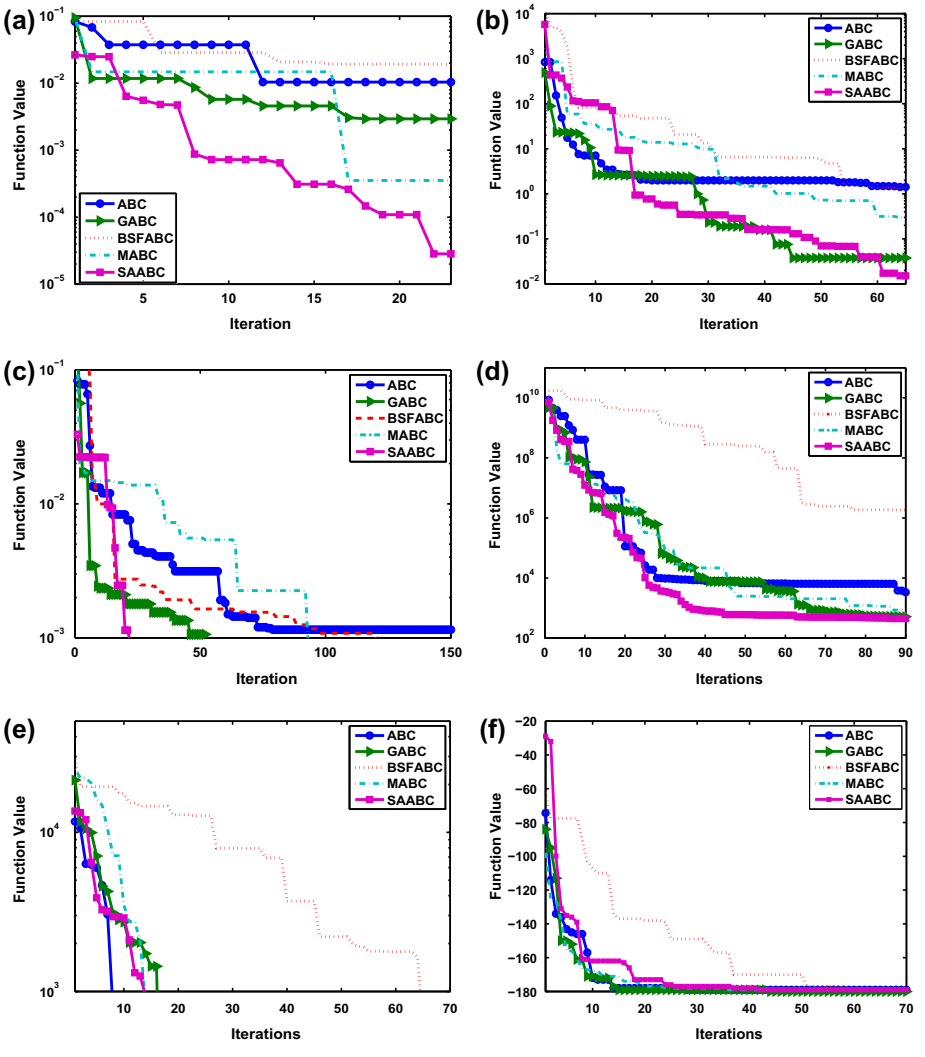

Figure 2. Convergence performance of the considered ABCs on (a) f_1, (b) f_2, (c) f_3, (d) f_4, (e) f_5 and (f) f_8 test functions.

For the purpose of comparison in terms of consolidated performance, boxplot analyses have been carried out for all the considered algorithms. The empirical distribution of data is efficiently represented graphically by the boxplot analysis tool.[30] The boxplots for ABC, GABC, BSFABC, MABC and SAABC are shown in Figure 3(a). It is clear from this figure that SAABC is better than the considered algorithms as interquartile range and median are comparatively low.

Further, to compare the considered algorithms, by giving weighted importance to the SR, the SD and the AFEs, performance indices (PI) are calculated.[31] The values of

Table 4. Acceleration Rate (AR) of $SAABC$ as compared to the basic ABC, GABC, BSFABC and MABC.

TP	ABC	GABC	BSFABC	MABC
f_1	9.099020153	5.130403891	26.17441713	5.700918159
f_2	15.54553902	11.80422892	11.96753025	11.53013426
f_3	4.135105744	2.080034633	3.387501454	4.289474034
f_4	1.529966836	0.862034189	1.563615798	1.5722235
f_5	1.016350003	0.629813384	2.043412076	0.981225686
f_6	0.999942957	0.999918409	1.000067042	0.999960105
f_7	1.000021347	0.999956455	1.000051844	0.999955855
f_8	0.602894556	0.374825368	1.12892641	0.839576061
f_9	0.880088182	0.491614833	1.638926069	0.744773902
f_{10}	29.60050969	1.06572326	3.780034374	3.833050473
f_{11}	15.89981642	4.110577071	0.427641201	17.51433804
f_{12}	1.499532859	0.813133335	2.877711728	2.433753247
f_{13}	1.685157898	0.898558262	1.384689838	2.459160383
f_{14}	12.01448991	2.516949069	7.708312063	4.427324668
f_{15}	1.735491646	0.931459874	3.300220944	12.1496978
f_{16}	1.29407922	1.176319986	1.584590116	4.779630303

Note: TP: test problems.

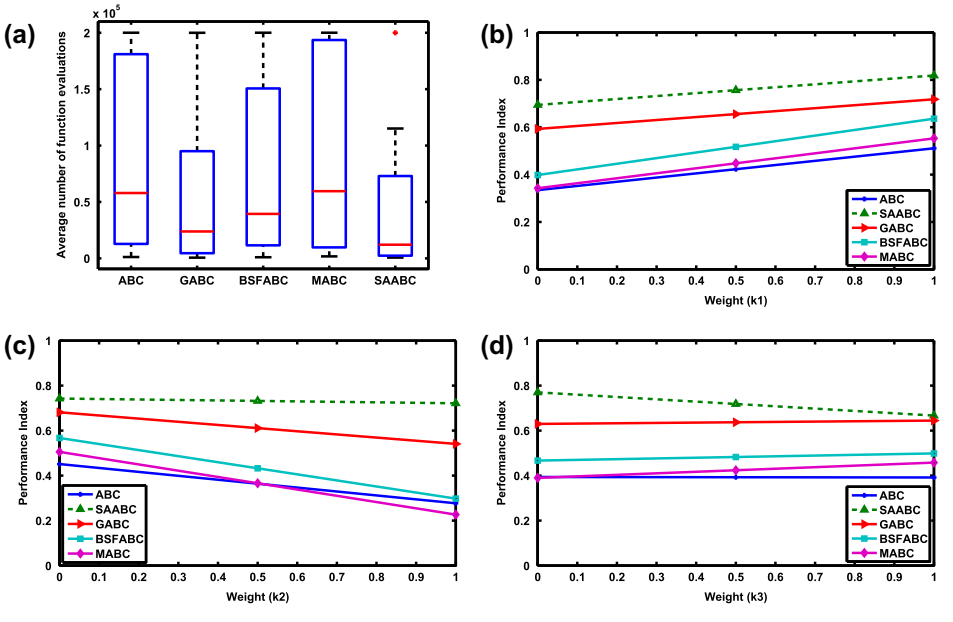

Figure 3. (a) Boxplot graphs for average function evaluation, performance index for test problems: (b) for case (1), (c) for case (2) and (d) for case (3).

PI for the ABC, SAABC, GABC, BSFABC and MABC are calculated using following equations:

$$PI = \frac{1}{N_p}\sum_{i=1}^{N_p}\left(k_1\alpha_1^i + k_2\alpha_2^i + k_3\alpha_3^i\right)$$

where $\alpha_1^i = \frac{Sr^i}{Tr^i}$; $\alpha_2^i = \begin{cases} \frac{Mf^i}{Af^i}, & \text{if } Sr^i > 0. \\ 0, & \text{if } Sr^i = 0. \end{cases}$; and $\alpha_3^i = \frac{Mo^i}{Ao^i}$ $i = 1, 2, \ldots, N_p$

- Sr^i = Successful simulations/runs of ith problem.
- Tr^i = Total simulations of ith problem.
- Mf^i = Minimum of AFEs used for obtaining the required solution of ith problem.
- Af^i = AFEs used for obtaining the required solution of ith problem.
- Mo^i = Minimum of SD obtained for the ith problem.
- Ao^i = SD obtained by an algorithm for the ith problem.
- N_p = Total number of optimization problems evaluated.

The weights assigned to the SR, the AFEs and the SD are represented by k_1, k_2 and k_3, respectively, where $k_1 + k_2 + k_3 = 1$ and $0 \leq k_1, k_2, k_3 \leq 1$. To calculate the PIs, equal weights are assigned to two variables, while weight of the remaining variable vary from 0 to 1 as given in [31]. Following are the resultant cases:

(1) $k_1 = W$, $k_2 = k_3 = \frac{1-W}{2}$, $0 \leq W \leq 1$;
(2) $k_2 = W$, $k_1 = k_3 = \frac{1-W}{2}$, $0 \leq W \leq 1$;
(3) $k_3 = W$, $k_1 = k_2 = \frac{1-W}{2}$, $0 \leq W \leq 1$

The graphs corresponding to each of the cases (1), (2) and (3) for ABC, SAABC, GABC, BSFABC and MABC are shown in Figure 3(b)–(d), respectively. In these figures, the weights k_1, k_2 and k_3 are represented by horizontal axis, while the PI is represented by the vertical axis.

In case (1), AFEs and SD are given equal weights. PIs of the considered algorithms are superimposed in Figure 3(b) for comparison of the performance. It is observed that PI of SAABC is higher than the considered algorithms. In case (2), equal weights are assigned to SR and SD and in case (3), equal weights are assigned to SR and AFEs. It is clear from Figure 3(c) and (d) that the algorithms perform same as in case (1).

After this intensive statistical analysis of numerical results of SAABC, ABC and some variants of ABC, it can be stated that the proposed SAABC is quite good candidate to be an algorithm for faster convergence and high probability of reaching to the global optima.

6. Conclusion

In this paper, to improve the exploitation in ABC, a self-adaptive solution update strategy is proposed and incorporated with ABC. Further, to give more time to better solutions to update themselves, 'limit' parameter of ABC is modified self-adaptively based on the fitness of the solutions. This setting of 'limit' makes low fit solutions less stable, which helps in exploration. To enhance the exploration, scout bees are increased. The so obtained MABC is named as, self-adaptive ABC (SAABC). It is shown that in the proposed strategy,

better solutions exploit the search space in their neighbourhood, while less fit solutions explore the search area. Further, the proposed algorithm is compared to the recent variants of ABC, namely GABC, BSFABC and MABC. With the help of numerical experiments over test problems, it is shown that the SAABC is a competitive algorithm to the considered algorithms in terms of reliability, efficiency and accuracy.

References

[1] Dorigo M, Di Caro G. Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99. Vol. 2. IEEE; 1999, Washington, DC.
[2] Kennedy J, Eberhart R. Particle swarm optimization. In: Neural networks, 1995. Proceedings IEEE international conference. Vol. 4. IEEE; 1995, Perth, Australia. p. 1942–1948.
[3] Price KV, Storn RM, Lampinen JA. Differential evolution: a practical approach to global optimization. Berlin: Springer Verlag; 2005.
[4] Vesterstrom J, Thomsen R. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In: Congress on Evolutionary Computation, 2004. CEC2004. Vol. 2, IEEE; Denmark: Aarhus University; 2004. p. 1980–1987.
[5] Kim S-S, Byeon J-H, Liu H, Abraham A, McLoone S. Optimal job scheduling in grid computing using efficient binary artificial bee colony optimization. Soft Comput. 2013;17:867–882.
[6] Passino KM. Biomimicry of bacterial foraging for distributed optimization and control. Control Syst. Mag. IEEE. 2002;22:52–67.
[7] Karaboga D. An idea based on honey bee swarm for numerical optimization. Tech. Rep. TR06. Erciyes: Erciyes University Press; 2005.
[8] Karaboga D, Akay B. A comparative study of artificial bee colony algorithm. Appl. Math. Comput. 2009;214:108–132.
[9] Zhu G, Kwong S. Gbest-guided artificial bee colony algorithm for numerical function optimization. Appl. Math. Comput. 2010;217:3166–3173.
[10] Banharnsakun A, Achalakul T, Sirinaovakul B. The best-so-far selection in artificial bee colony algorithm. Appl. Soft Comput. 2011;11:2888–2901.
[11] Akay B, Karaboga D. A modified ABC algorithm for real-parameter optimization. Inf. Sci. 2012;192:120–142.
[12] Gao W, Liu S. A modified artificial bee colony algorithm. Comput. Oper. Res. 2011;9:687–697.
[13] Haijun D, Qingxian F. Bee colony algorithm for the function optimization. Science Paper Online. August 2008.
[14] Tsai PW, Pan JS, Liao BY, Chu SC. Enhanced artificial bee colony optimization. Int. J. Innovative Comput. Inf. Control. 2009;5:5081–5092.
[15] Baykasoglu A, Ozbakir L, Tapkan. P. Artificial bee colony algorithm and its application to generalized assignment problem. In: Chan FTS, Tiwari MK, editors. Swarm intelligence: focus on ant and particle swarm optimization. Vienna: Itech Education and Publishing; 2007. p. 113–144.
[16] Karaboga D, Akay B. A modified artificial bee colony (ABC) algorithm for constrained optimization problems. Appl. Soft Comput. 2011;11:3021–3031.
[17] Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. Comput. Meth. Appl. Mech. Eng. 2000;186:311–338.
[18] Alam MS, Ul Kabir MW, Islam MM. Self-adaptation of mutation step size in artificial bee colony algorithm for continuous function optimization. In: 13th International Conference on Computer and Information Technology (ICCIT), 2010; Dhaka: Ahsanullah University of Science & Technology; 2010. IEEE. p. 69–74.

[19] El-Abd M. A cooperative approach to the artificial bee colony algorithm. In: 2010 IEEE Congress on Evolutionary Computation (CEC). IEEE; Canberra: University of New South Wales at ADFA; 2010. p. 1–5.
[20] Kang F, Li J, Ma Z. Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions. Inf. Sci. 2011;181:3508–3531.
[21] Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Boston (MA): Addison-Wesley; 1989.
[22] Storn R, Price K. Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. J. Global Optim. 1997;11:341–359.
[23] Hofmann K, Whiteson S, de Rijke M. Balancing exploration and exploitation in learning to rank online. Adv. Inf. Retrieval. 2011;5:251–263.
[24] Ali MM, Khompatraporn C, Zabinsky ZB. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. J. Global Optim. 2005;31:635–672.
[25] Suganthan PN, Hansen N, Liang JJ, Deb K, Chen YP, Auger A, Tiwari S. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. In: CEC 2005; Australia: Queensland University; 2005.
[26] Diwold K, Aderhold A, Scheidler A, Middendorf M. Performance evaluation of artificial bee colony optimization and new selection schemes. Memetic Comput. 2011;3:1–14.
[27] El-Abd M. Performance assessment of foraging algorithms vs. evolutionary algorithms. Inf. Sci. 2011;182:243–263.
[28] Karaboga D, Basturk B. Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems. In: Foundations of fuzzy logic and soft computing; Cancun; 2007. p. 789–798.
[29] Rahnamayan S, Tizhoosh HR, Salama MMA. Opposition-based differential evolution. IEEE Trans. Evol. Comput. 2008;12:64–79.
[30] Williamson DF, Parker RA, Kendrick JS. The box plot: a simple visual method to interpret data. Ann. Internal Med. 1989;110:916–921.
[31] Deep K, Thakur M. A new crossover operator for real coded genetic algorithms. Appl. Math. Comput. 2007;188:895–911.