Journal of Experimental & Theoretical Artificial Intelligence

ISSN: 0952-813X (Print) 1362-3079 (Online) Journal homepage: http://www.tandfonline.com/loi/teta20

Escalated convergent artificial bee colony

Shimpi Singh Jadon, Jagdish Chand Bansal & Ritu Tiwari

To cite this article: Shimpi Singh Jadon, Jagdish Chand Bansal & Ritu Tiwari (2016) Escalated convergent artificial bee colony, Journal of Experimental & Theoretical Artificial Intelligence, 28:1-2, 181-200, DOI: 10.1080/0952813X.2015.1020523

To link to this article: http://dx.doi.org/10.1080/0952813X.2015.1020523

Journal of Experimental & Theoretical Artificial Intelligence, 2016
Vol. 28, Nos. 1–2, 181–200, http://dx.doi.org/10.1080/0952813X.2015.1020523

Taylor & Francis
Taylor & Francis Group

Escalated convergent artificial bee colony

Shimpi Singh Jadon[a]*, Jagdish Chand Bansal[b] and Ritu Tiwari[a]

[a]*ABV-Indian Institute of Information Technology and Management, Gwalior, India;* [b]*South Asian University, New Delhi, India*

(*Received 29 September 2014; accepted 12 October 2014*)

Artificial bee colony (ABC) optimisation algorithm is a recent, fast and easy-to-implement population-based meta heuristic for optimisation. ABC has been proved a rival algorithm with some popular swarm intelligence-based algorithms such as particle swarm optimisation, firefly algorithm and ant colony optimisation. The solution search equation of ABC is influenced by a random quantity which helps its search process in exploration at the cost of exploitation. In order to find a fast convergent behaviour of ABC while exploitation capability is maintained, in this paper basic ABC is modified in two ways. First, to improve exploitation capability, two local search strategies, namely classical unidimensional local search and levy flight random walk-based local search are incorporated with ABC. Furthermore, a new solution search strategy, namely stochastic diffusion scout search is proposed and incorporated into the scout bee phase to provide more chance to abandon solution to improve itself. Efficiency of the proposed algorithm is tested on 20 benchmark test functions of different complexities and characteristics. Results are very promising and they prove it to be a competitive algorithm in the field of swarm intelligence-based algorithms.

Keywords: optimisation; swarm intelligence; memetic algorithm; artificial bee colony

1. Introduction

During the current decade, swarm intelligence is being used broadly to solve optimisation problems. Swarm intelligence is based on the collective behaviour of social structures, which find solutions by collaborative trial and error. Researchers designed algorithms by analysing this solution finding behaviours of social insects to deal with nonlinear, non-convex or discrete optimisation problems. Previous research (Dorigo & Di Caro, 1999; Kennedy & Eberhart, 1995; Price, Storn, & Lampinen, 2005; Vesterstrom & Thomsen, 2004) has proven that algorithms based on swarm intelligence have great potential to deal with complex real-world optimisation problems. The set of algorithms in this category includes spider monkey optimisation (Bansal, Sharma, Jadon, & Clerc, 2014), particle swarm optimisation (Kennedy & Eberhart, 1995), ant colony optimisation (Dorigo & Di Caro, 1999), bacterial foraging optimisation (Passino, 2002), firefly algorithm (Yang, 2010), artificial bee colony (ABC) optimisation (Karaboga, 2005), etc.

The ABC optimisation algorithm, introduced by Karaboga (2005), is relatively a new member in swarm intelligence-based algorithms family. The ABC algorithm is a simple population-based optimisation algorithm which mimics foraging behaviour of honeybees while searching food sources for them. Here population consists of the potential solutions in terms of

*Corresponding author. Email: shimpisingh2k6@gmail.com

the food sources for honeybees. The fitness of each food source is determined in terms of nectar amount that it has. The whole ABC procedure includes two fundamental processes: variation process and the selection process which are responsible for exploration and exploitation, respectively. But Karaboga and Akay (2009) concluded that without converging to a local optimum, it may occasionally stop proceeding towards the global optimum. Zhu and Kwong (2010) also observed that the solution search equation of the ABC algorithm has good exploration but at the cost of exploitation. Therefore, removal of the same demands hybridisation of one or more local search approaches in the basic ABC to enhance the exploitation in the search region. Recently, Sharma, Jadon, Bansal, and Arya (2013) proposed a levy flight local search (LFLS) strategy and incorporated the proposed strategy with differential evolution (DE) algorithm. Neri and Tirronen (2009) proposed the scale factor local search DE which is a memetic algorithm composed of DE and two simple local search strategies namely golden section search and hill climbing. Kang, Li, Ma, and Li (2011) proposed a Hooke Jeeves artificial bee colony algorithm (HJABC) for numerical optimisation. In HJABC, the authors incorporated a local search technique which is based on the Hooke Jeeves method (Hooke & Jeeves, 1961) with the basic ABC. Bansal, Sharma, Arya, and Nagar (2013) also incorporated the concept of memetic search in ABC to enhance its exploitation capabilities. Furthermore, Mezura-Montes and Velez-Koeppel (2010) introduced a variant of the basic ABC named elitist artificial bee colony by integrating two local search strategies. The first local search strategy is used when 30%, 40%, 50%, 60%, 70%, 80%, 90%, 95% and 97% of function evaluations have been completed while the other local search works when 45%, 50%, 55%, 80%, 82%, 84%, 86%, 88%, 90%, 91%, 92%, 93%, 94%, 95%, 96%, 97%, 98% and 99% of the function evaluations are reached. The purpose of this is to improve the best solution achieved so far by generating a set of 1000 new food sources in its neighbourhood. Many of the recent modifications and applications of the ABC algorithm can be studied in Bansal, Sharma, and Jadon (2013). Therefore, in order to work for the same aim, we proposed a memetic algorithm (escalated convergent ABC, EcABC) composed of ABC and two simple local search approaches: classical unidimensional local search (CULS) based on classical unidimensional search (Gardeux, Chelouah, Siarry, & Glover, 2009) and a local search (LFLS) based on levy flight random walk (Sharma et al., 2013), activated by a deterministic and randomised criterion, respectively. Furthermore, a new search scheme for scout bee, stochastic diffusion scout search (SDSS), is proposed based on the concept of stochastic diffusion search (SDS; Bishop, 2007) to increase the convergence capability of ABC. In SDSS, whenever a food source is exhausted, then a scout bee does not always re-initialise itself randomly in the search space like in a standard ABC but it finds a new food source in the proposed neighbourhood (refer Equation (5)) of its randomly selected neighbour having better fitness. The local search CULS and LFLS works differently for the aim of exploitation. The CULS directly exploits the best solution of each iteration further by generating a set of diverse trial solutions near it in a predefined iteratively reduced range and selects a solution which is better than the other previously visited solutions. While the LFLS on the other hand updates the best solution found so far by iteratively reducing the step size of the best solution movement in the search space.

The rest of the paper is organised as follows: Section 2 briefs the standard ABC. Section 3 explains the proposed algorithm EcABC with description of working principles of each introduced component. The proposed algorithm is tested and compared with the basic ABC and its recent variants, namely Gbest guided artificial bee colony algorithm (GABC) (Zhu & Kwong, 2010), best-so-far artificial bee colony (BSFABC) (Banharnsakun, Achalakul, & Sirinaovakul, 2011) and modified artificial bee colony (MABC) (Akay & Karaboga, 2012) in Section 4. Finally, in Section 5, we conclude the paper.

2. Artificial bee colony algorithm

The ABC algorithm is relatively a simple and recent swarm intelligence-based algorithm. The ABC algorithm is a simulation of food foraging behaviour of honeybees. In ABC, each solution of the problem is called as food source of honeybees. The fitness of each food source is proportional to its nectar quality. In ABC, bees are categorised into three groups: employed bees, onlooker bees and scout bees. The number of employed bees is equal to the number of onlooker bees. The employed bees start searching the food sources and store the information about the quality of the food source. Onlooker bees wait in the hive for the employed and select the food sources based on the information provided by the employed bees. The scout bee finds new food sources in search space randomly in place of the exhausted food sources. The ABC solution search process is also an iterative process like the other population-based algorithms. In ABC, first swarm initialisation is done, then it is a iterative procedure of three phases: employed bee phase, onlooker bee phase and scout bee phase. Each of the phases is explained as follows.

2.1 Initialisation of the swarm

If D is the number of variables in the optimisation problem, then each food source $x_i (i = 1, 2, \ldots, \mathrm{SN})$ is a D-dimensional vector and is generated using a uniform distribution as

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

where x_i represents the ith food source in the swarm, SN is the total number of food sources, $x_{\min j}$ and $x_{\max j}$ are bounds of x_i in jth direction and $\mathrm{rand}\,[0, 1]$ is a uniformly distributed random number in the range [0, 1]. After initialisation phase, ABC requires the cycle of the three phases, namely employed bee phase, onlooker bee phase and scout bee phase to be executed.

2.2 Employed bee phase

In this phase, for each ith potential solution, a new trial position is generated by modifying its single randomly selected dimension using the following equation:

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

where v_{ij} is new trial position, $k \in \{1, 2, \ldots, \mathrm{SN}\}$ and $j \in \{1, 2, \ldots, D\}$ are randomly chosen indices and $k \neq i$. ϕ_{ij} is a random number between $[-1, 1]$. After generating the new position, the position with better fitness between the newly generated and the old one is selected.

2.3 Onlooker bees phase

In this phase, employed bees share the information associated with their food sources such as quality (nectar) and position of the food source with the onlooker bees in the hive. Onlooker bees evaluate the available information about the food sources, and based on fitness they select solutions with a probability prob_i to search new positions around these. Here prob_i is calculated as a function of fitness (there may be some other method):

$$\mathrm{prob}_i(G) = \frac{0.9 \times \mathrm{fitness}_i}{\mathrm{maxfit}} + 0.1, \tag{3}$$

where $\mathrm{fitness}_i$ is the fitness value of the ith solution and maxfit is the maximum fitness amongst

all the solutions. Again by applying greedy selection, if the fitness is higher than the previous one, the onlooker bee stores the new position in its memory and forgets the old position.

2.4 Scout bees phase

If for a predetermined number of times, any bee's position is not being modified, then that food source is considered as abandoned or exhausted, i.e. it is not worth exploiting any more and the associated bee becomes a scout bee. In this phase, the exhausted food source is replaced by a randomly generated food source (as in initialisation phase) within the search space. In ABC, the number of times after which a particular food source becomes exhausted is known as *limit* and is a crucial control parameter.

Furthermore, Zhu and Kwong (2010) introduced an improved version of ABC, called GABC to improve the basic ABC algorithmic performance. In GABC, the authors incorporated information about the best bee found so far in the position update Equation (2) as

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) + \psi_{ij}(y_j - x_{kj}), \tag{4}$$

where y_j is jth dimension of the global best bee found so far and ψ is a random number between $(0, C)$, C is a user-defined positive constant.

2.5 Main steps of the ABC algorithm

The pseudo-code of the ABC is shown in Algorithm **1**.

Algorithm 1. ABC algorithm:

Initialise the parameters;
while Termination criterion 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: Memorise the best food source found so far;
end while
Output the best solution found so far.

3. Escalated convergent ABC

Karaboga and Akay (2009) analysed and compared the different variants of ABC for global optimisation and found that the ABC shows poor performance, and concluded that without converging to a local optimum, it may occasionally stop proceeding towards the global optimum. Zhu and Kwong (2010) also observed that the solution search equation of the ABC algorithm has good exploration but at the cost of exploitation. The exploration in an algorithm refers to the ability of exploring the various unknown regions of the solution space in order to avoid being trapped in local optima, while exploitation in an algorithm refers to the ability of refining the existing previous good solutions to find better solutions in their surroundings in order to avoid skipping true optima. So, to achieve good performance of the optimisation algorithm, both exploitation and exploration abilities are necessary and should be well balanced. 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, is 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 the current solution and randomly selected solution is large with high absolute value of ϕ_{ij}, then the 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 then balance the exploration and exploitation capabilities of ABC. But, since this step size consists of a random component, the balance cannot be done manually.

One way of avoiding the situation of skipping a true solution while maintaining the speed of convergence is the incorporation of one or more local searches into the basic ABC process. The local search algorithm, in case of large step sizes, can search within the area that is jumped by the basic ABC. During the iterations, the local search algorithm exhibits very strong exploitation capability due to the execution of efficient local search on solutions (Wang, Wang, & Yang, 2009).

This paper aims at enhancing the performance of ABC by incorporation of two competitive local search strategies: CULS based on CUS (Gardeux et al., 2009) and a local search (LFLS) based on levy flight random walk (Sharma et al., 2013). This is done in expectation of increased exploitation capability of ABC. The scout bee search scheme has also been modified using SDS (Bishop, 2007) for the exhausted food source. The proposed algorithm, namely EcABC, is explained in the following section.

3.1 Algorithmic framework

At the beginning of the algorithm search process, an initial set of solutions, i.e. food positions, $x_i (i = 1, 2, \ldots, SN)$ is generated randomly in dimensional bound using Equation (1). Now for each solution x_{ij}, a trial position is generated by modifying one of its dimensions using Equation (4), and one with better fitness between old and new food source is stored in memory of the employed. Then the onlooker bee generates the new position using the same Equation (4) for only particular bees selected based on the probability defined in Equation (3) and memorises the best position. Now each food source is checked for its limit counter, i.e. if any food source is not being updated to a user-defined parameter called 'limit', then the corresponding bee becomes scout. In the proposed EcABC algorithm, scout bee searches a new food position using the proposed strategy: SDSS, which is explained in Section 3.1.1. Now the bee with best fitness is selected and then based on the probability, at a time one of the two introduced local searches, CULS and LFLS (explained in Section 3.1.2), plays the expected role of exploitation of the available knowledge about the problem. This whole process is repeated until the termination criterion is satisfied. The pseudo-code of the proposed EcABC algorithm is shown in Algorithm **2**.

3.1.1 Stochastic diffusion scout search

In this paper, we proposed a new search strategy for scout bees based on SDS. Initially, SDS was proposed by Bishop (2007) and Omran and Salman (2012) as a population-based best-fit pattern-matching algorithm. Bishop suggested that SDS can be recast for optimisation purpose by defining the objective function $f(x)$ for a hypothesis x. So, we utilised this fact and modelled SDS as per demand of our proposed algorithm EcABC as a search strategy for the scout bee. Originally, SDS has a swarm of n agents and is an iterative process of two phases: TEST and DIFFUSION. Here, initialisation (agents) means occupying initial position in the search space.

Algorithm 2. EcABC:

```
Initialise the parameters;
while Termination criteria do
   Step 1: Employed bee phase for generating new food sources using Equation (4);
   Step 2: Onlooker bees phase for updating the food sources based on probability (defined in Equation 3)
           using Equation (4);
   Step 3: Apply SDSS using proposed Algorithm 4 for scout bee phase for discovering the new food source
           in place of abandoned food source;
   Step 4: Find the best solution so far;
   Step 5:
   If (rand(0, 1) < 0.5) then
      Apply CULS on best solution so far using Algorithm 5;
   Else
      Apply LFLS on best so far solution using Algorithm 6;
   end if
end while
Print best solution.
```

Algorithm 3. SDS:

```
Initialise (agents);
while Until convergence (agents) do
   Step 1: TEST (agents);.
   Step 2: DIFFUSE (agents).
end while
```

In TEST (agents) phase, each agent i is assigned a status active or inactive based on fitness. For each agent i, another agent j is randomly selected from the swarm; if agent i has better fitness than j, then status of agent i is set as active otherwise inactive. In DIFFUSION phase, agents share their surroundings among them through inter-agent communication. In standard SDS, each inactive agent i communicates with a randomly selected member j of the swarm; if this member j is active, then agent i shifts itself in the neighbourhood of agent j and if j is inactive then agent i re-initialises its position randomly across the entire search space. In EcABC, we recast SDS as per our objective and applied it for repositioning of the scout bee. We eliminated test phase in SDS as we require it only for the scout bee and consider the scout bee as an inactive agent. Now for scout bee i (say), we randomly select another bee j (say) from the colony. If bee j has better fitness than scout bee i, then scout bee i generates its new position in the neighbourhood of jth bee, else scout bee re-initialises itself randomly in the search space as in standard ABC. Definition of neighbourhood of a bee x (say) is adopted from Omran and Salman (2012)) and is defined as $L \leq x \leq U$, where L and U are defined as follows:

$$\begin{aligned} L &= x - 0.5 * \text{rand} * (x_{\max} - x_{\min}), \\ U &= x + 0.5 * \text{rand} * (x_{\max} - x_{\min}), \end{aligned} \tag{5}$$

where rand is a uniform random number between 0 and 1 and $x_{\max}, x_{\min}$ are bounds of x in search space. If any of the L and U crosses its bounds, then following bounds clamping is adopted.

if $(L < x_{\min})$ **then**
$L = x_{\min}$ and $U = U + (x_{\min} - L)$;
else if $(U > x_{\max})$ **then**
$U = x_{\max}$ and $L = L - (U - x_{\max})$;
end if

The SDSS which will be executed in scout phase in our EcABC Algorithm **2** is explained in Algorithm **4**.

Algorithm 4. SDSS:

Input scout x_i and objective function $Minf(x)$;
Select a random member x_k from the swarm;
if $(f(x_k) < f(x_i))$ **then**
Generate neighbourhood $(L\ ,\ U)$ of x_k using Equation (5) with bounds clamping defined in Section 3.1.1,
for $j = 1$ to D **do**
$x_{ij} = L_{kj} + \text{rand}[0,1](U_{kj} - L_{kj})$
end for
else
for $j = 1$ to D **do**
$x_{ij} = x_{\min j} + \text{rand}[0,1](x_{maxj} - x_{\min j})$
end for
end if

3.1.2 *Local search strategies*

While adopting the local search techniques in general global search algorithms, all or some individuals of the population try to identify better solutions in their small neighbourhoods. So, these techniques are implemented in between the running iterations of population-based global search algorithms. In these algorithms, the global search ability of population-based algorithms tries to get the most promising regions in the search space, while the incorporated local search strategy examines closely the surroundings of some already found regions, i.e. focuses rightly on exploitation. Here the basic ABC is modified by incorporating two local searches CULS and LFLS for exploiting the best solution in its surroundings. Neri & Tirronen (2009) have shown that the combined use of two local searches is more beneficial than the use of a single local search integrated into global optimisation algorithms. So, here two different local search strategies are chosen in accordance with the principle of conducting a search under complementary perspectives. Activation of the local search is performed by a simple probabilistic criterion of 50% probability as shown in Algorithm **2**.

- **Classical unidimensional local search**: CULS algorithm works as a local search algorithm in which only the best individual of the current swarm updates itself in its neighbourhood. In CULS, the step size, required to update the best individual in the current swarm, is controlled by the CUS approach (Gardeux et al., 2009). The proposed CULS is shown in Algorithm **5**:

It is clear from Algorithm **5** that two new solutions are generated around the best solution by adding a small step size. Then, through the greedy selection process, a best solution among the

Algorithm 5. CULS:

Input best solution so far $best_solution$;
Initialise $h_{\min}$
for $j = 1$ to D **do**
  $h_j = \frac{x_{maxj} - x_{minj}}{10}$.
end for;
$j = U(1, D)$
while $h_j > h_{\min}$ **do**
  $next,\ previous = best\_solution$;
  $next_j = best_solution_j + h_j$;
  $previous_j = best_solution_j - h_j$;
  evaluate next and previous fitness;
  x = best of next, previous and $best solution$;
  while better solution found **do**
    $x_j = x_j \pm h_j$ in best direction
  end while
  $best_solution = x$;
  $h_j = h_j \times 0.5$;
  $j = U(1, D)$;
end while

three solutions (next, previous and best_solution) is selected based on fitness. In this algorithm, h_i represents the step size, which is required to update a randomly selected dimension of the best_solution. Initially, h_i is set equal to the 10th part of the difference between the upper and lower bound of the ith direction to the search space. Here, the difference is divided by 10 to keep the initial step size small so that the newly generated solution could not skip the true solution or jump out side the search space. Then this algorithm iterates until a stopping criterion is reached. At each iteration, the algorithm focuses on optimising the best solution found so far only in one dimension i. After each iteration, h is decreased by a ratio value fixed to 0.5 until h becomes smaller than a pre-defined value of minimum h_i denoted by $h_{\min}$ (in this paper it is fixed to the 100th part of the difference between the upper and lower bounds of the ith direction to the search space), and then h is not decreased further. The minimum value of step size is fixed to maintain the progressiveness of the search process or to avoid the stagnation of the solutions.

- **Levy flight random walk based local search**: Sharma et al. (2013) proposed a local search strategy (LFLS) based on levy flight and incorporated with the DE algorithm. We incorporated the same concept here also in EcABC. The levy flight is a random walk in which the amount of movement is defined as step-length. The random step lengths are drawn from a certain probability distribution called levy distribution and defined as Equation (6) (Sharma et al., 2013):

$$L(s) \sim |s|^{-1-\beta}, \quad \text{where } \beta\,(0 < \beta \leq 2) \text{ is an index and } s \text{ is the step length.} \tag{6}$$

The local search utilises Mantegna's algorithm (Yang, 2011) to generate random step sizes using a symmetric levy stable distribution. Here the term 'symmetric' means that the step size may be positive or negative. In Mantegna's algorithm, the step length s is calculated by

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

where u and v are normally distributed numbers, i.e. $u \sim N(0, \sigma_u^2)$, $v \sim N(0, \sigma_v^2)$ where

$$\sigma_u = \left\{ \frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\beta\Gamma[(1+\beta)/2]2^{(\beta-1)/2}} \right\}^{1/\beta}, \quad \sigma_v = 1. \tag{8}$$

This distribution (for s) obeys the expected levy distribution for $|s| \geq |s_0|$, where s_0 is the smallest step length (Yang, 2011). Here $\Gamma(.)$ is the Gamma function and is calculated as follows:

$$\Gamma(1+\beta) = \int_0^\infty t^\beta \mathrm{e}^{-t} \mathrm{d}t. \tag{9}$$

In a special case when β is an integer, then we have $\Gamma(1+\beta) = \beta!$.

In the proposed strategy, levy distribution is used to generate the step sizes to exploit the search area and is calculated as follows:

$$step\_size(t) = 0.01 \times s(t) \times \mathrm{SLC}, \tag{10}$$

where t is the iteration counter for the local search strategy, $s(t)$ is calculated using levy distribution as shown in Equation (7) and SLC is the social learning component of the global search algorithm. In levy flights, the step sizes are too aggressive and, therefore, new solutions may be generated outside the domain or on the boundary. Since the main task of local search algorithms is to exploit the available knowledge about a problem, steps sizes are responsible for exploitation of the identified region. Hence, 0.01 multiplier is used in Equation (10) so that new solutions remain in search space.

The solution update equation based on the proposed local search strategy is as follows:

$$\dot{x_{ij}}(t+1) = x_{ij}(t) + step\_size(t) \times U(0,1), \tag{11}$$

where x_{ij} is the current position of ith individual and $step_size(t) \times U(0,1)$ is the actual amount of flight drawn from the levy distribution which is being added to move an individual to the next position.

Algorithm 6. LFLS:

Input optimisation function $Minf(x)$ and β;
Select an individual x_i in the swarm which is going to be modified;
Initialise $t = 1$ and $\sigma_v = 1$;
Compute σ_u using Equation (8);
while $(t < \varepsilon)$**do**
  Compute *step_size* using Equation (10);
  Generate a new solution $x_i^{'}$ using Equation (11);
  Calculate $f(x_i^{'})$
  If $(f(x_i^{'}) < f(x_i))$ **then**
    $x_i = x_i^{'}$;
  end if
  $t = t + 1$;
end while

The pseudo-code of the LFLS is shown in Algorithm 6 (Sharma et al., 2013). In algorithm **6**, ε determines the termination of local search which is set to be10 for the current algorithm as suggested in the original article.

4. Experimental results and discussion

4.1 Test problems under consideration

For performance analysis purpose of the proposed algorithm EcABC, we applied it on 20 different continuous optimisation benchmark problems (f_1–f_{20}) of different degrees of complexities and modalities (listed in Table 1). Test problems f_1–f_{12} and f_{15}–f_{20} are taken from Ali, Khompatraporn, and Zabinsky (2005) and the test problems f_{12}–f_{14} are taken from Suganthan et al. (2005) with the associated offset values.

4.2 Experimental setting

To prove the superiority of the proposed algorithm EcABC, results are compared with the basic ABC (Karaboga, 2005) and with recent variants of ABC namely BSFABC (Banharnsakun et al., 2011), GABC (Zhu & Kwong, 2010) and MABC (Akay & Karaboga, 2012). The following parameter setting is adopted to test EcABC and other considered algorithms:

- Colony size NP = 50 and number of food sources SN = NP/2 (Diwold, Aderhold, Scheidler, & Middendorf, 2011; El-Abd, 2012).
- ϕ_{ij} = rand$[-1, 1]$ and limit = 1500 (Akay & Karaboga, 2012; Karaboga & Akay, 2011).
- The stopping criterion is either maximum number of function evaluations (which is set to be 200,000) is reached or the acceptable error (mentioned in Table 1) has been achieved.
- The termination criterion ε of LFLS is set to be 10 as suggested in Sharma et al. (2013).

4.3 Results comparison

Numerical results of the proposed and other considered algorithms with parameter setting in Section 4.2 are given in Table 2 in terms of success rate (SR), average number of function evaluations (AFEs) and mean error (ME). Table 3 shows that EcABC outperforms the other algorithms in terms of reliability, efficiency and accuracy most of the time. In order to analyse relative performance of EcABC statistically, boxplots, acceleration rate (AR) and PIs have been carried out.

EcABC, ABC, BSFABC, GABC and MABC are compared through success rate (SR), mean error (ME) and average number of function evolutions (AFE) in Table 2. Now, pairwise comparison is done between algorithms as first SR is compared between the algorithms, and if they have achieved the same SR, then comparison is made on the basis of AFE. ME is then used for comparison if it is not possible to distinguish them based on both SR and AFE. This comparison analysis outcome is presented in Table 3. In Table 3, ‘+’ indicates that the EcABC is better than the considered algorithm and ‘−’ indicates that the algorithm is not better. The last row of Table 3 reflects the superiority of our proposed algorithm EcABC over ABC, BSFABC, GABC and MABC, as it contains 72 ‘+’ signs out of 80 comparisons. GABC beats proposed algorithm on four functions (f_1,f_2,f_{14} and f_{20}), ABC beats EcABC on two functions (f_{10} and f_{16}) while BSFABC beats EcABC on only a single function f_{10}.

To compare the convergence speed of the algorithms, we calculated the AR based on the AFEs using Equation (12). Smaller AFEs means higher convergence speed. The number of

Table 1. Test problems.

Test problem	Objective function	Search range	Optimum value	D	Acceptable error
De Jong f4	$f_1(x) = \sum_{i=1}^{D} i \cdot (x_i)^4$	$[-5.12\ 5.12]$	$f(\vec{0}) = 0$	30	$1.0E-05$
Griewank	$f_2(x) = 1 + \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right).$	$[-600\ 600]$	$f(\vec{0}) = 0$	30	$1.0E-05$
Rosenbrock	$f_3(x) = \sum_{i=1}^{D} \left(100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right)$	$[-30\ 30]$	$f(\vec{1}) = 0$	30	$1.0E-02$
Salomon problem	$f_4(x) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{D} x_i^2}\right) + 0.1\left(\sqrt{\sum_{i=1}^{D} x_i^2}\right)$	$[-100\ 100]$	$f(\vec{0}) = 0$	30	$1.0E-01$
Sum of different powers	$f_5(x) = \sum_{i=1}^{D} \lvert x_i\rvert^{i+1}$	$[-1\ 1]$	$f(\vec{0}) = 0$	30	$1.0E-05$
Inverted cosine wave	$f_6(x) = -\sum_{i=1}^{D-1} \left(\exp\left(\frac{-\left(x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}\right)}{8}\right) \times I\right)$	$[-5\ 5]$	$f(\vec{0}) = -D + 1$	10	$1.0E-05$
Neumaier 3 Problem (NF3)	where, $I = \cos\left(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}}\right)$ $f_7(x) = \sum_{i=1}^{D}(x_i - 1)^2 - \sum_{i=2}^{D} x_i x_{i-1}$	$[-D^2\ \ D^2]$	$f_{\min} = -((D(D+4)(D-1))/6)$	10	$1.0E-01$
Beale	$f_8(x) = [1.5 - x_1(1 - x_2)]^2 + \left[2.25 - x_1\left(1 - x_2^2\right)\right]^2 + \left[2.625 - x_1\left(1 - x_2^3\right)\right]^2$	$[-4.5\ 4.5]$	$f(3, 0.5) = 0$	2	$1.0E-05$
Colville function	$f_9(x) = 100\left[x_2 - x_1^2\right]^2 + (1 - x_1)^2 + 90\left(x_4 - x_3^2\right)^2 + (1 - x_3)^2 + 10.1\left[(x_2 - 1)^2 + (x_4 - 1)^2\right] + 19.8(x_2 - 1)(x_4 - 1)$	$[-10\ 10]$	$f(\vec{1}) = 0$	4	$1.0E-05$
Branins's function	$f_{10}(x) = a\left(x_2 - bx_1^2 + cx_1 - d\right)^2 + e(1 - f)\cos x_1 + e$	$-5 \le x_1 \le 10, 0 \le x_2 \le 15$	$f(-\pi, 12.275) = 0.3979$	2	$1.0E-05$
Kowalik function	$f_{11}(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1\left(b_i^2 + b_i x_2\right)}{b_i^2 + b_i x_3 + x_4}\right]^2$	$[-5\ 5]$	$f(0.1928, 0.1908, 0.1231, 0.1357) = 3.07E-04$	4	$1.0E-05$
2D Tripod	$f_{12}(x) = p(x_2)(1 + p(x_1)) + \lvert(x_1 + 50p(x_2)(1 - 2p(x_1)))\rvert + \lvert(x_2 + 50(1 - 2p(x_2)))\rvert$	$[-100\ 100]$	$f(0, -50) = 0$	2	$1.0E-04$
Shifted Rosenbrock	$f_{13}(x) = \sum_{i=1}^{D-1} \left(100\left(z_i^2 - z_{i+1}\right)^2 + (z_i - 1)^2\right) + f_{\text{bias}},$ $z = x - o + 1, \quad x = [x_1, x_2, \ldots . x_D], \quad o = [o_1, o_2, \ldots o_D]$	$[-100\ 100]$	$f(o) = f_{\text{bias}} = 390$	10	$1.0E-01$

(*Continued*)

Table 1 – *continued*

Test problem	Objective function	Search range	Optimum value	D	Acceptable error
Shifted Griewank	$f_{14}(x) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} \cos(\frac{z_i}{\sqrt{i}}) + 1 + f_{\text{bias}}$, $z = (x - o), \quad x = [x_1, x_2, \ldots x_D], \quad o = [o_1, o_2, \ldots o_D]$	[− 600 600]	$f(o) = f_{\text{bias}} = -180$	10	1.0E − 05
Goldstein-Price	$f_{15}(x) = \left(1 + (x_1 + x_2 + 1)^2 \cdot \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right) \cdot \left(30 + (2x_1 - 3x_2)^2 \cdot \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right)$	[− 2 2]	$f(0, -1) = 3$	2	1.0E − 14
Six-hump camel back	$f_{16}(x) = \left(4 - 2.1x_1^2 + x_1^4/3\right)x_1^2 + x_1x_2 + \left(-4 + 4x_2^2\right)x_2^2$	[− 5 5]	$f(-0.0898, 0.7126) = -1.0316$	2	1.0E − 05
Easom's function	$f_{17}(x) = -\cos x_1 \cos x_2 e^{((-(x_1 - \pi)^2 - (x_2 - \pi)^2))}$	[− 10 10]	$f(\pi, \pi) = -1$	2	1.0E − 13
Meyer and Roth problem	$f_{18}(x) = \sum_{i=1}^{5} \left(\frac{x_1x_3t_i}{1 + x_1t_i + x_2v_i} - y_i\right)^2$	[− 10 10]	$f(3.13, 15.16, 0.78) = 0.4E - 04$	3	1.0E − 03
Shubert	$f_{19}(x) = -\sum_{i=1}^{5} i\cos\left((i + 1)x_1 + 1\right)\sum_{i=1}^{5} i\cos((i + 1)x_2 + 1)\right)$	[− 10 10]	$f(7.0835, 4.8580) = -186.7309$	2	1.0E − 05
Moved axis parallel hyper-ellipsoid	$f_{20}(x) = \sum_{i=1}^{D} 5i \times x_i^2$	[− 5.12 5.12]	$f(x) = 0$; $x(i) = 5{*}i, \quad i = 1 : D$	30	1.0E − 15

Table 2. Comparison of the results of test problems, TP: test problem.

TP	Algorithm	ME	AFE	SR
f_1	EcABC	5.24E − 06	8917	100
	ABC	4.90E − 06	9579	100
	BSFABC	5.31E − 06	24,525	100
	GABC	5.51E − 06	8388	100
	MABC	8.63E − 06	22,584	100
f_2	EcABC	5.04E − 06	35845	100
	ABC	2.28E − 04	45,943	97
	BSFABC	5.96E − 06	43,314	100
	GABC	6.33E − 06	33,335	99
	MABC	9.24E − 06	41,967	100
f_3	EcABC	3.42E − 01	162,762	51
	ABC	1.34E + 00	186,025	22
	BSFABC	1.68E + 00	179,325	23
	GABC	6.34E + 00	176,041	25
	MABC	3.60E + 01	189,783	9
f_4	EcABC	9.18E − 01	30,156	98
	ABC	9.79E − 01	188,482	56
	BSFABC	9.62E − 01	158,846	62
	GABC	9.30E − 01	121,725	98
	MABC	9.39E − 01	27,650	99
f_5	EcABC	5.55E − 06	7381	100
	ABC	5.46E − 06	16,229	100
	BSFABC	6.08E − 06	14,299	100
	GABC	5.92E − 06	12,018	100
	MABC	8.01E − 06	9389	100
f_6	EcABC	7.48E − 06	42,757	100
	ABC	2.63E − 02	87,183	86
	BSFABC	6.85E − 02	99,189	88
	GABC	7.22E − 06	68,526	98
	MABC	8.40E − 06	68,784	100
f_7	EcABC	1.02E − 01	36,243	90
	ABC	9.32E + 00	198,559	4
	BSFABC	4.24E + 00	193,324	6
	GABC	1.62E + 00	184,896	23
	MABC	1.29E + 00	129,183	38
f_8	EcABC	5.34E − 06	5651	100
	ABC	8.39E − 06	27,162	95
	BSFABC	2.19E − 05	26,843	100
	GABC	5.76E − 06	14,568	100
	MABC	5.30E − 06	10,051	100
f_9	EcABC	1.07E − 02	100,321	57
	ABC	1.51E − 01	200,000	0
	BSFABC	2.20E − 02	157,940	32
	GABC	1.47E − 02	162,248	31
	MABC	1.40E − 02	141,156	36
f_{10}	EcABC	5.52E − 06	22,879	89
	ABC	5.12E − 06	2069	100
	BSFABC	5.42E − 06	18,816	94
	GABC	5.63E − 06	22,961	89
	MABC	6.29E − 06	30,512	86
f_{11}	EcABC	8.57E − 05	78,150	98

(*Continued*)

Table 2 – *continued*

TP	Algorithm	ME	AFE	SR
	ABC	1.76E − 04	180,759	20
	BSFABC	1.42E − 04	144,038	55
	GABC	8.80E − 05	93,359	88
	MABC	1.95E − 04	176,378	24
f_{12}	EcABC	6.93E − 05	8029	100
	ABC	7.57E − 05	19,168	89
	BSFABC	8.11E − 05	9796	96
	GABC	7.04E − 05	8151	100
	MABC	1.91E − 04	18,621	90
f_{13}	EcABC	8.59E − 02	55,839	99
	ABC	6.68E + 00	184,801	19
	BSFABC	2.70E + 00	170,663	21
	GABC	8.15E − 01	100,011	61
	MABC	9.49E − 01	159,602	32
f_{14}	EcABC	2.53E − 04	50,925	98
	ABC	1.09E − 03	84,618	88
	BSFABC	4.29E − 03	98,951	80
	GABC	7.89E − 05	40,248	99
	MABC	6.28E − 04	82,942	92
f_{15}	EcABC	4.40E − 14	4703	100
	ABC	2.19E − 06	126,899	50
	BSFABC	4.42E − 07	93,493	57
	GABC	5.88E − 07	97,443	53
	MABC	5.49E − 07	98,707	54
f_{16}	EcABC	1.06E − 04	90,326	55
	ABC	1.20E − 05	989	100
	BSFABC	1.63E − 04	98,426	51
	GABC	1.93E − 04	122,223	39
	MABC	1.70E − 04	102,787	49
f_{17}	EcABC	4.26E − 14	4608	100
	ABC	4.89E − 05	191,885	8
	BSFABC	4.97E − 14	25,290	100
	GABC	5.27E − 14	43,731	100
	MABC	8.04E − 06	180,174	12
f_{18}	EcABC	1.95E − 03	3612	100
	ABC	1.95E − 03	24,724	100
	BSFABC	1.95E − 03	19,190	99
	GABC	1.94E − 03	5933	100
	MABC	1.95E − 03	9916	100
f_{19}	EcABC	4.29E − 06	1878	100
	ABC	4.93E − 06	9942	100
	BSFABC	5.29E − 06	9479	100
	GABC	4.58E − 06	4816	100
	MABC	5.02E − 06	10,168	100
	EcABC	7.28E − 16	44,393	100
f_{20}	ABC	9.24E − 16	72,981	98
	BSFABC	7.15E − 16	71,056	100
	GABC	9.22E − 16	39,628	100
	MABC	9.14E − 16	60,021	100

Table 3. Summary of Table 2 outcome.

Test problems	EcABC vs. ABC	EcABC vs. BSFABC	EcABC vs. GABC	EcABC 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}	−	+	+	+
f_{17}	+	+	+	+
f_{18}	+	+	+	+
f_{19}	+	+	+	+
f_{20}	+	+	−	+
Total number of + signs	18	19	16	19

function evaluations for each test problem is averaged over 100 runs to reduce the effect of stochastic nature of the algorithms.

$$\mathrm{AR} = \frac{\mathrm{AFE}_{\mathrm{ALGO}}}{\mathrm{AFE}_{\mathrm{EcABC}}}, \tag{12}$$

where ALGO $\in \{\mathrm{ABC}, \mathrm{BSFABC}, \mathrm{GABC}, \mathrm{MABC}\}$. Table 4 reports the outcome of comparison between EcABC and ABC, EcABC and BSFABC, EcABC and GABC, and EcABC and MABC in terms of AR. It is clear from Equation (12) that AR > 1 means EcABC is faster. Table 4 shows that convergence speed of EcABC is faster among all the considered algorithms for most of the functions. The same conclusion can also be drawn from Figure 1.

For consolidated performance comparison, boxplot analyses have been executed for all considered algorithms. The empirical distribution of data is efficiently represented graphically by the boxplot analysis tool (Williamson, Parker, & Kendrick, 1989). The boxplots for EcABC, ABC, BSFABC, GABC and MABC are shown in Figure 2. It is clear from this figure that EcABC is better than the considered algorithms as interquartile range and median are comparatively low.

Furthermore, to compare the considered algorithms based on all success rates, the average number of function evaluations and the mean error simultaneously, PI (Deep & Thakur, 2007) are calculated by giving weighted importance to the SR, AFE and ME. The values of PI for EcABC, ABC, BSFABC, GABC and MABC are calculated by using the following equations:

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

Table 4. AR of EcABC compared with other considered algorithms.

Test problems	EcABC vs. ABC	EcABC vs. BSFABC	EcABC vs. GABC	EcABC vs. MABC
f_1	1.074137163	2.750188116	0.940633975	2.5325796
f_2	1.281695394	1.208370123	0.929955489	1.170765525
f_3	1.142926921	1.101762395	1.081588705	1.166018991
f_4	6.250232126	5.267475793	4.036510147	0.916898793
f_5	2.198756535	1.937273997	1.628236863	1.272051581
f_6	2.039041221	2.319838267	1.602690188	1.608724308
f_7	5.478547582	5.334105896	5.10156444	3.564357255
f_8	4.806523366	4.75007388	2.577918872	1.778601221
f_9	1.993600542	1.574346348	1.617288504	1.407043391
f_{10}	0.090419871	0.82241525	1.003596357	1.333652116
f_{11}	2.31298955	1.84311226	1.194618751	2.256930338
f_{12}	2.387360738	1.220043717	1.015183804	2.31926217
f_{13}	3.309555998	3.056354234	1.791066419	2.858263661
f_{14}	1.661605999	1.94305674	0.79032774	1.628696704
f_{15}	26.98026959	19.87781817	20.71756602	20.98630778
f_{16}	0.010943852	1.089677345	1.353137317	1.137955353
f_{17}	41.64171007	5.48828125	9.490234375	39.10026042
f_{18}	6.845113082	5.312907843	1.642676198	2.745289992
f_{19}	5.294211619	5.04765962	2.564566803	5.414558816
f_{20}	1.643971328	1.60059748	0.892661046	1.352034133

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

$$i = 1, 2, \ldots, N_p$$

- Sr^i = successful simulations/runs of ith problem.
- Tr^i = total simulations of ith problem.
- Mf^i = minimum of average number of function evaluations used for obtaining the

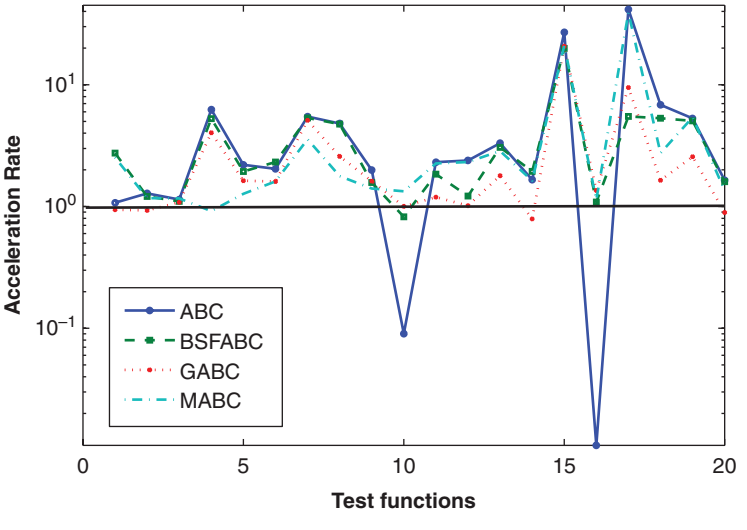

Figure 1. AR of EcABC compared with the basic ABC, BSFABC, GABC and MABC.

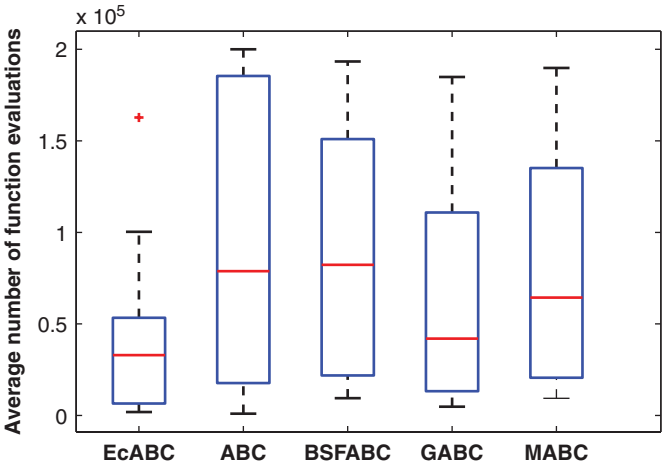

Figure 2. Boxplots graphs for average number of function evaluations.

required solution of *i*th problem.

- Af^i = average number of function evaluations used for obtaining the required solution of *i*th problem.
- Mo^i = minimum of mean error obtained for the *i*th problem.
- Ao^i = mean error obtained by an algorithm for the *i*th problem.
- N_p = total number of optimisation problems evaluated.

The weights assigned to the success rate, the average number of function evaluations and the mean error 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 weights of the remaining variables vary from 0 to 1 as given in Deep and Thakur (2007). Following are the resultant cases:

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

The graphs corresponding to each of the cases (1), (2) and (3) for EcABC, ABC, GABC, BSFABC and MABC are shown in Figure 3(a)–(c), 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), the average number of function evaluations and the mean error are given equal weights, and weight to success rate varies. PIs of the considered algorithms are superimposed in Figure 3(a) for comparison of the performance. It is observed that PI of EcABC is higher than the considered algorithms. In case (2), success rate and mean error are assigned equal weights, and in case (3), equal weights are given to the success rate and average function evaluations. It is clear from Figure 3(b) and 3(c) that in these cases also EcABC algorithm performs better than other algorithms.

Overall, we found that EcABC has high convergent rate to the global optima as compared with other well-established versions of ABC, which justifies its name as escalated convergent ABC.

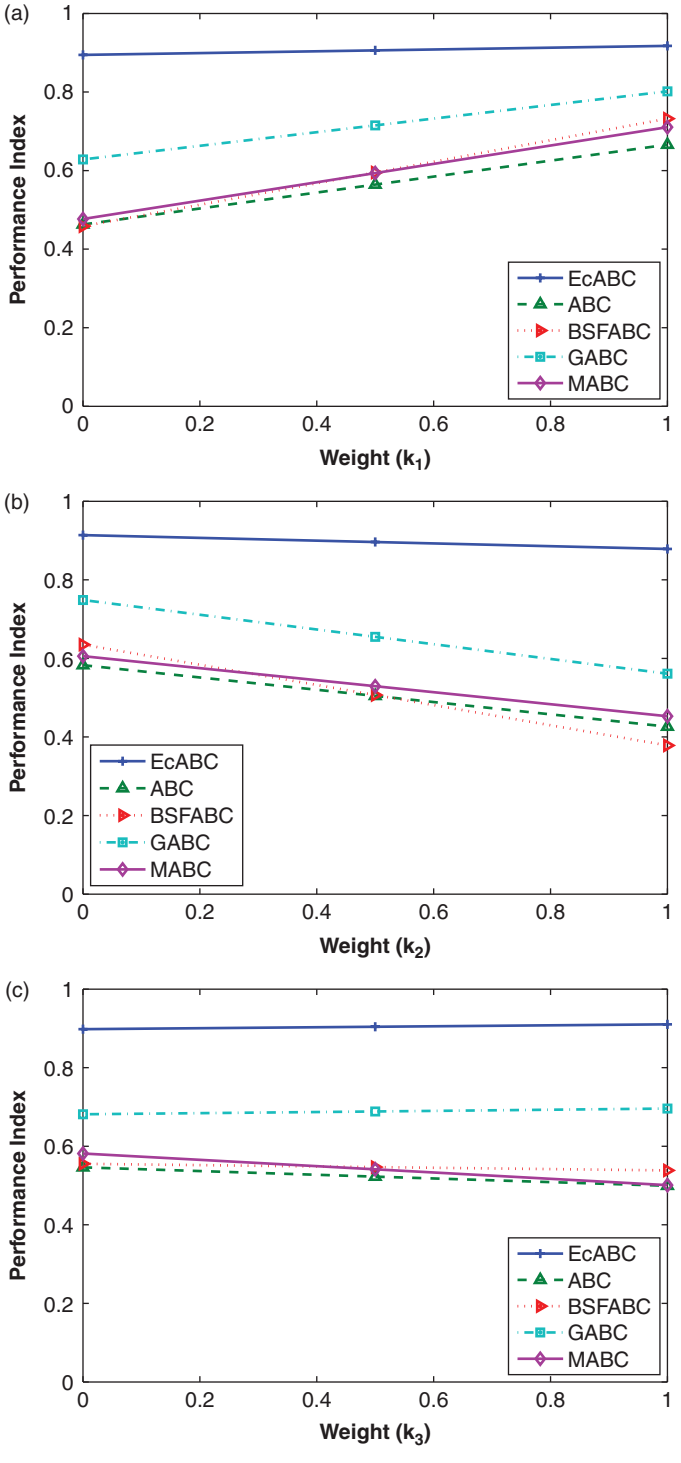

Figure 3. PI for test problems; (a) for case (1), (b) for case (2) and (c) for case (3).

5. Conclusion

This paper proposes a modified ABC, namely EcABC which is a memetic algorithm composed of a standard ABC and two local search strategies CULS and LFLS. Moreover, this paper proposes a new search strategy for scout bee, namely SDSS based on SDS. Based on the probability, at a time, one of the CULS and LFLS local search strategies is applied on the best individual of each iteration in the ABC algorithm in order to find more promising solutions in the territory of the best individual. Because of local search strategies and a modified scout search procedure, EcABC has shown a fast convergent behaviour towards the global optima. EcABC is compared with ABC and with its recent variants, namely BSFABC, GABC and MABC, and experimental results over the test problems show the improved and promising performance of the proposed algorithm EcABC in terms of reliability, efficiency and accuracy. In future, we will focus to check the efficiency of this proposed version of ABC on complex real-world problems.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Akay, B., & Karaboga, D. (2012). A modified Artificial Bee Colony algorithm for real-parameter optimization. *Information Sciences*, *192*, 120–142. doi:10.1016/j.ins.2010.07.015

Ali, M., Khompatraporn, C., & Zabinsky, Z. (2005). A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization*, *31*, 635–672. doi:10.1007/s10898-004-9972-2

Banharnsakun, A., Achalakul, T., & Sirinaovakul, B. (2011). The best-so-far selection in Artificial Bee Colony algorithm. *Applied Soft Computing*, *11*, 2888–2901. doi:10.1016/j.asoc.2010.11.025

Bansal, J. C., Sharma, H., Arya, K., & Nagar, A. (2013). Memetic search in artificial bee colony algorithm. *Soft Computing*, *17*, 1911–1928. doi:10.1007/s00500-013-1032-8

Bansal, J. C., Sharma, H., & Jadon, S. S. (2013). Artificial bee colony algorithm: A survey. *International Journal of Advanced Intelligence Paradigms*, *5*, 123–159. doi:10.1504/IJAIP.2013.054681

Bansal, J. C., Sharma, H., Jadon, S. S., & Clerc, M. (2014). Spider monkey optimization algorithm for numerical optimization. *Memetic Computing*, *6*, 31–47. doi:10.1007/s12293-013-0128-0

Bishop, J. M. (2007). Stochastic diffusion search. *Scholarpedia*, *2*, 3101. doi:10.4249/scholarpedia.3101

Deep, K., & Thakur, M. (2007). A new crossover operator for real coded genetic algorithms. *Applied Mathematics and Computation*, *188*, 895–911. doi:10.1016/j.amc.2006.10.047

Diwold, K., Aderhold, A., Scheidler, A., & Middendorf, M. (2011). Performance evaluation of artificial bee colony optimization and new selection schemes. *Memetic Computing*, *3*, 149–162. doi:10.1007/s12293-011-0065-8

Dorigo, M., & Di Caro, G. (1999). Ant colony optimization: A new meta-heuristic. *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*. doi:10.1109/cec.1999.782657

El-Abd, M. (2012). Performance assessment of foraging algorithms vs. evolutionary algorithms. *Information Sciences*, *182*, 243–263. doi:10.1016/j.ins.2011.09.005

Gardeux, V., Chelouah, R., Siarry, P., & Glover, F. (2009). Unidimensional search for solving continuous high-dimensional optimization problems. *2009 Ninth International Conference on Intelligent Systems Design and Applications*. doi:10.1109/isda.2009.191

Hooke, R., & Jeeves, T. (1961). ‘ Direct search’ solution of numerical and statistical problems. *Journal of the ACM (JACM)*, *8*, 212–229. doi:10.1145/321062.321069

Kang, F., Li, J., Ma, Z., & Li, H. (2011). Artificial Bee Colony Algorithm with Local Search for Numerical Optimization. *Journal of Software*, *6*, 490–497. doi:10.4304/jsw.6.3.490-497

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Technical Report TR06 . Erciyes: Erciyes University Press.

Karaboga, D., & Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, *214*, 108–132. doi:10.1016/j.amc.2009.03.090

Karaboga, D., & Akay, B. (2011). A modified artificial bee colony (ABC) algorithm for constrained optimization problems. *Applied Soft Computing*, *11*, 3021–3031. doi:10.1016/j.asoc.2010.12.001

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95 – International Conference on Neural Networks*. doi:10.1109/icnn.1995.488968

Mezura-Montes, E., & Velez-Koeppel, R. (2010). Elitist artificial bee colony for constrained real-parameter optimization. In *Congress on Evolutionary Computation (CEC'2010)* (pp. 2068–2075). Barcelona: IEEE Service Center.

Neri, F., & Tirronen, V. (2009). Scale factor local search in differential evolution. *Memetic Computing*, *1*, 153–171. doi:10.1007/s12293-009-0008-9

Omran, M. G. H., & Salman, A. (2012). Probabilistic stochastic diffusion search. *Lecture Notes in Computer Science*, 300–307. doi:10.1007/978-3-642-32650-9_31

Passino, K. (2002). Biomimicry of bacterial foraging for distributed optimization and control. *Control Systems Magazine*, *22*, 52–67. doi:10.1109/MCS.2002.1004010

Price, K., Storn, R., & Lampinen, J. (2005). *Differential evolution: A practical approach to global optimization*. Springer Verlag.

Sharma, H., Jadon, S. S., Bansal, J. C., & Arya, K. V. (2013). Lèvy flight based local search in differential evolution. *Swarm, Evolutionary, and Memetic Computing*, 248–259. doi:10.1007/978-3-319-03753-0_23

Suganthan, P., Hansen, N., Liang, J., Deb, K., Chen, Y., Auger, A., & Tiwari, S. (2005). Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. *KanGAL Report* 2005005.

Vesterstrom, J., & Thomsen, R. (2004). A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Evolutionary Computation, 2004. CEC, 2004*. Congress on IEEE (Vol. 2, pp. 1980–1987).

Wang, H., Wang, D., & Yang, S. (2009). A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, *13*, 763–780.

Williamson, D., Parker, R., & Kendrick, J. (1989). The box plot: A simple visual method to interpret data. *Annals of Internal Medicine*, *110*, 916. doi:10.7326/0003-4819-110-11-916

Yang, X. (2010). Firefly algorithm, Levy flights and global optimization. *Research and Development in Intelligent Systems*, *XXVI*, 209–218.

Yang, X. (2011). *Nature-inspired metaheuristic algorithms*. Luniver Press.

Zhu, G., & Kwong, S. (2010). Gbest-guided artificial bee colony algorithm for numerical function optimization. *Applied Mathematics and Computation*, *217*, 3166–3173. doi:10.1016/j.amc.2010.08.049