Artificial Bee Colony Algorithm with Improved Explorations for Numerical Function Optimization

Mohammad Shafiul Alam[1], Md. Monirul Islam[2], Kazuyuki Murase[3]

[1] Department of Computer Science and Engineering, Ahsanullah University of Science and Technology, Dhaka 1208, Bangladesh
shuvo23@gmail.com
[2] Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000, Bangladesh
mdmonirulislam@cse.buet.ac.bd
[3] Department of Human and Artificial Intelligence Systems, University of Fukui, Fukui 910-8507, Japan
murase@synapse.his.fukui-u.ac.jp

Abstract. A major problem with Artificial Bee Colony (ABC) algorithm is its premature convergence to local optima, which originates from lack of explorative search capability of the algorithm. This paper introduces ABC with Improved Explorations (ABC-IX), a novel algorithm that modifies both the selection and perturbation operations of the basic ABC algorithm in an explorative way. Unlike the basic ABC algorithm, ABC-IX employs a probabilistic, explorative selection scheme based on simulated annealing which can accept both better and worse candidate solutions. ABC-IX also maintains a self-adaptive perturbation rate, separately for each candidate solution, to promote more explorations. ABC-IX is tested on a number of benchmark problems for numerical optimization and compared with several recent variants of ABC. Results show that ABC-IX often outperforms the other ABC-variants on most of the problems.

Keywords: Artificial bee colony algorithm, exploration and exploitation.

1 Introduction

The Artificial Bee Colony (ABC) algorithm [1] is a recently introduced swarm intelligence algorithm inspired by the intelligent food foraging behavior of honey bees. ABC and its variants have frequently showed superior performance in comparison to many other existing evolutionary and swarm intelligence algorithms [2]. Over the last few years, ABC has been successfully applied to wide and diverse range of problems, such as numerical optimization, discrete optimization, multi-objective optimization, machine learning, design of IIR filters, PID controller, software testing and so on [3].

In comparison to other greedy and local search based algorithms, ABC is more resilient against local optima, because the population of candidate solutions provides an advantage of preserving diversity and continuing explorations of the search space. However, from practical experiences, it is often found that the evolving population of

candidate solutions loses its diversity and explorative capability too soon and the solutions prematurely get trapped around the locally optimal points of the fitness landscape. Another problem often found with ABC is the fitness stagnation, where the population of solutions stops progressing towards the global optimum for no apparent reason, even without converging to some local optima [4]. The risk of fitness stagnation and premature convergence usually depends on the mix of explorative and exploitative operations during the search. Similar to other population based metaheuristic algorithms, ABC drives its search towards global optimum with two operators – perturbation and selection. The perturbation operation is generally responsible for explorations of different regions of the search space by random alteration of the existing candidate solutions, while the fitness based selection operation of ABC performs the exploitation of the search regions explored so far.

There exist a number of recent works (e.g. [5-8]) that attempt to alter the explorative and/or exploitative properties of the standard ABC algorithm to avoid premature convergence and fitness stagnation. However, most of them focus on altering the perturbation operation only. In the literature, not much has been reported to improve the greedy fitness-based selection procedure of ABC. Our proposed algorithm – ABC with Improved Explorations (ABC-IX) alters both the selection and perturbation operations of the standard ABC algorithm to increase the explorative capacity of both the operations. A detailed description of both the improvements by ABC-IX is presented in section 3. Increased explorations can spread the population across more search regions allowing more possible trial solutions to be produced, which make ABC-IX robust against fitness stagnation and premature convergence.

The rest of this paper is organized as follows. Section 2 describes the standard ABC algorithm with its pseudocode. Section 3 presents the proposed algorithm – ABC-IX and explains the improvements on the selection and perturbation operations along with their pseudocode. Section 4 provides details of the benchmark problems, parameter settings of the algorithms, comparison of their results and experiments on ABC-IX to investigate its improvements. Finally, section 5 concludes with a few new directions for further study with ABC-IX.

2 Artificial bee colony (ABC) algorithm

The ABC algorithm mimics the food foraging behavior of honey bees with three groups of bees that are found in nature, i.e. employed, onlooker and scout bees. A bee working to forage a particular food source (i.e., candidate solution) previously visited by itself and searching only around its vicinity is called an employed bee. Onlooker bees randomly pick and follow any of the employed bees. The probability of picking an employed bee is proportional to the quality of its food source. Scout bees perform random explorations of the search space to find new food sources. Suppose $\boldsymbol{x_i}$ is a food source currently being visited by an employed bee. The bee employs (1) to search around the neighborhood of $\boldsymbol{x_i}$ in order to produce a new, trial food source $\boldsymbol{v_i}$.

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

Here, $j \in \{1, 2, \ldots, D\}$ and $k \in \{1, 2, \ldots, SN\}$ are randomly picked indices, D is the dimensionality of the problem, SN is the number of food positions, φ_{ij} is a uniform random value produced from [-1, 1]. If $\boldsymbol{v_i}$ has better 'fitness' than the old food source position $\boldsymbol{x_i}$, then $\boldsymbol{x_i}$ is replaced by $\boldsymbol{v_i}$. Else, $\boldsymbol{v_i}$ is discarded. With f being the function to be minimized, ABC computes the 'fitness' of a candidate solution, say $\boldsymbol{x_i}$, using (2).

$$fitness(\boldsymbol{x_i}) = \begin{cases} 1/\big(1 + f(\boldsymbol{x_i})\big), & \text{if } f(\boldsymbol{x_i}) \geq 0 \\ 1 + |f(\boldsymbol{x_i})| & \text{otherwise} \end{cases} \quad (2)$$

An onlooker bee randomly picks an employed bee $\boldsymbol{x_i}$ to follow and forages around the vicinity of its food source. The probability p_i that a food source $\boldsymbol{x_i}$ would be picked by an onlooker bee is computed using (3), which makes the probability p_i proportional to $fitness(\boldsymbol{x_i})$. This ensures that a more promising solution $\boldsymbol{x_i}$ with high fitness value is often foraged and exploited more intensively by (1) than relatively less fit solutions.

$$p_i = fitness(\boldsymbol{x_i}) / \sum_{n=1}^{SN} fitness(\boldsymbol{x_n}) \quad (3)$$

If a particular food source position $\boldsymbol{x_i}$ has not been improved over the last *limit* cycles, then it is abandoned and the bee employed to $\boldsymbol{x_i}$ now becomes a scout bee that is placed at random across the search space using (4), where $j = 1, 2, \ldots, D$ and $[min_j, max_j]$ is the search space along the j-th dimension.

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

Algorithm: Artificial Bee Colony (ABC) Algorithm

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

Fig. 1. Pseudocode for the standard Artificial Bee Colony (ABC) algorithm

3 ABC algorithm with improved explorations (ABC-IX)

The proposed algorithm – ABC-IX differs from the standard ABC algorithm in two important ways. First, ABC accepts a newly produced solution $\boldsymbol{v_i}$ only if $\boldsymbol{v_i}$ has higher fitness value than the original solution $\boldsymbol{x_i}$ (Fig. 1, steps 5,9). This exploitative selection

Algorithm: Simulated Annealing (SA) based Probabilistic Selection Scheme for ABC-IX
input: Two candidate solutions $\boldsymbol{x_i}$ and $\boldsymbol{v_i}$
output: Either of $\boldsymbol{x_i}$ and $\boldsymbol{v_i}$ selected to be included into the population
begin
  if $fitness(\boldsymbol{v_i}) \geq fitness\ (\boldsymbol{x_i})$ **then**
    return $\boldsymbol{v_i}$
  else
    $\Delta E = fitness(\boldsymbol{x_i}) - fitness\ (\boldsymbol{v_i})$
    if $rand(0,1) \leq \exp(-\Delta E/T)$ **then**
      return $\boldsymbol{v_i}$
    endif
  endif
  return $\boldsymbol{x_i}$
end

Algorithm: Perturbation by ABC-IX
input: An existing candidate solution $\boldsymbol{x_i}$
output: Perturbed candidate solution $\boldsymbol{v_i}$
begin
  $\boldsymbol{v_i} = \boldsymbol{x_i}$
  if $rand(0,1) \leq t$ **then**
    $q[\boldsymbol{v_i}] = q_{min} + (1.0 - q_{min}) \times rand(0,1)$
  else
    $q[\boldsymbol{v_i}] = q[\boldsymbol{x_i}]$
  endif
  for $j = 1$ to D **do**
    if $rand(0,1) \leq q[\boldsymbol{v_i}]$ **then**
      $k = rand\{1, 2, ..., SN\}$
      $\varphi_{ij} = rand(-1,1)$
      $v_{ij} = x_{ij} + \varphi_{ij} \times (x_{kj} - x_{ij})$
    endif
  enddo
  return $\boldsymbol{v_i}$
end

Fig. 2. Pseudocode for simulated annealing (SA) based probabilistic selection scheme (on the left) and perturbation with self-adaptive perturbation rate (on the right) for ABC-IX

Algorithm: Artificial Bee Colony Algorithm with Improved Explorations (ABC-IX)

1: Initialize the population of SN food source positions (i.e., candidate solutions) $\boldsymbol{x_i}$, for $i = 1, 2, ..., SN$. Each $\boldsymbol{x_i}$ is a vector of D parameters: $\boldsymbol{x_i} = [x_{i1}, x_{i2}, ..., x_{iD}]^T$
2: Evaluate the fitness of each food source positions using (2)
3: **repeat**
4*: Perturb each food source $\boldsymbol{x_i}$ to produce a new food $\boldsymbol{v_i}$ using the pseudocode for perturbation in Fig. 2
5*: Evaluate each new solution $\boldsymbol{v_i}$ by (2). Select either $\boldsymbol{x_i}$ or $\boldsymbol{v_i}$ by SA-based probabilistic selection (Fig. 2)
6: Calculate the probability p_i associated with each food source position $\boldsymbol{x_i}$ using (3)
7: For each onlooker bee, assign it to a food source $\boldsymbol{x_i}$, proportionally based on the probability p_i
8*: Produce $\boldsymbol{v_i}$ by perturbing the food position $\boldsymbol{x_i}$ of each onlooker bee with the perturbation code (Fig. 2)
9*: Evaluate each new solution $\boldsymbol{v_i}$ by (2). Select either $\boldsymbol{x_i}$ or $\boldsymbol{v_i}$ by SA-based probabilistic selection (Fig. 2)
10: If a food source has not improved during the last *limit* cycles, then abandon it and replace it with a new, randomly placed scout bee with its food source $\boldsymbol{x_i}$ produced by (4).
11: Memorize the best food source position found so far
12*: Set cycle counter $C = C + 1$ and system temperature T = α * T
13: **until** C = Maximum cycle number (MCN)
14: **return** the best food source position (i.e., candidate solution) found so far

Fig. 3. Pseudocode for ABC-IX. Steps that differ from the ABC algorithm are marked with '*'

scheme denies any possible downhill movement and allows only uphill steps in the fitness landscape, which may lead to local optima. In contrast, ABC-IX promotes more search space explorations by probabilistically allowing some downhill steps using a simulated annealing-based selection scheme (Fig. 2). Second, ABC perturbs only a single parameter of each existing solution $\boldsymbol{x_i}$ which usually produces the new solution $\boldsymbol{v_i}$ in the neighborhood of $\boldsymbol{x_i}$, which is exploitative. In contrast, ABC-IX can perturb any number of parameters of $\boldsymbol{x_i}$ by introducing a self-adaptive perturbation rate for each individual solution $\boldsymbol{x_i}$, which is addressed as $q[\boldsymbol{x_i}]$ (Fig. 2). The value of $q[\boldsymbol{x_i}]$ is self-adapted gradually, cycle by cycle, separately for each $\boldsymbol{x_i}$. Fig. 3 presents

the pseudocode for ABC-IX, which differs from ABC in how the existing solutions are perturbed (steps 4, 8) and how the new solutions are selected into the population (steps 5, 9). They are further elaborated in the following paragraphs.

3.1 Simulated annealing (SA) based probabilistic selection

SA accepts both better and worse new solutions, but the probability of accepting a worse new solution is reduced over time, depending on a gradually decreasing control parameter T (i.e. temperature, from the analogy to the real annealing procedure in metallurgy). Given an initial candidate solution $\boldsymbol{x_i}$, SA randomly perturbs $\boldsymbol{x_i}$ to produce $\boldsymbol{y_i}$ in the neighborhood of $\boldsymbol{x_i}$. The change ΔE of the objective function value, $\Delta E = obj(\boldsymbol{x_i}) - obj(\boldsymbol{y_i})$. If $\boldsymbol{y_i}$ is better than $\boldsymbol{x_i}$ (i.e. $\Delta E < 0$), SA accepts $\boldsymbol{y_i}$ as its current state and discards $\boldsymbol{x_i}$. If $\boldsymbol{y_i}$ is worse (i.e. $\Delta E > 0$), SA may still accept $\boldsymbol{y_i}$, but only with probability = $e^{\frac{-\Delta E}{T}}$. SA usually starts with a high initial temperature T_0 to ensure high degree of initial explorations by frequently accepting worse solutions (larger T makes $\frac{\Delta E}{T} \rightarrow 0$, thus probability $e^{\frac{-\Delta E}{T}} \rightarrow 1$). As T gradually decreases, SA becomes increasingly exploitative, accepting better solutions only. The SA-based probabilistic selection scheme (Fig. 2) improves the explorative capacity of ABC-IX, because it can now accept both better and worse solutions to be more resilient against local optima. T is gradually decreased by the exponential cooling schedule: $T(t+1) = \alpha \cdot T(t)$. We used $\alpha = 0.99$. The initial temperature T_0 is set to 50 times of the difference of fitness values of the best and the worst solutions of the first generation.

3.2 Self-adaptive perturbation rate

ABC perturbs only a single, random parameter of the existing solutions using (1). This allows search along a single dimension at a time, which may be suitable for separable problems, but is inappropriate for non-separable problems. In contrast, ABC-IX can perturb any number of parameters allowing search along any possible direction. For every candidate solution $\boldsymbol{x_i}$, ABC-IX separately maintains and adapts a control parameter $q[\boldsymbol{x_i}]$, which denotes the probability of each parameter of $\boldsymbol{x_i}$ to be perturbed by the perturbation operation. Note that, in the pseudocode (Fig. 2), the value of $q[\boldsymbol{x_i}]$ is perturbed first, with probability t, using (5), before perturbing any parameter of $\boldsymbol{x_i}$ to produce $\boldsymbol{v_i}$. This perturbed value $q[\boldsymbol{v_i}]$ is then used as the perturbation rate (PR) when producing $\boldsymbol{v_i}$ from $\boldsymbol{x_i}$. A better value of PR is supposed to lead towards better offspring solution $\boldsymbol{v_i}$, which is more likely to survive than $\boldsymbol{x_i}$ and produce better, new trial solutions and hence, propagate the better values of the PR. Thus the gradual self-adaptation towards better, more effective PR values takes place across the population. We have used $t = 0.10$, $q_{max} = 1.0$ and $q_{min} = 1/D$.

$$q[\boldsymbol{v_i}] = \begin{cases} q_{min} + rand(0,1) * (q_{max} - q_{min}), & \text{if } rand(0,1) < t \\ q[\boldsymbol{x_i}] & \text{otherwise} \end{cases} \quad (5)$$

4 Experimental studies

ABC-IX is evaluated using a suite of 13 standard benchmark problems [9] for numerical optimization and compared with some recent variants of the ABC algorithm. For page limit, the benchmark suite functions are not presented here, but they can be readily found as f_1–f_{13} in [9]. These functions have been widely used in many recent studies on evolutionary and swarm intelligence algorithms. The suite contains both unimodal (f_1–f_7) and multimodal (f_8–f_{13}) functions. A function is called multimodal if it has multiple local optima. To optimize such a function, the search algorithm must possess both exploitative and explorative properties so that it can explore the locally optimal points without being trapped anywhere and thus keep exploring further for better unvisited search regions. Some of the multimodal functions can have hundreds of local minima, even when the dimensionality is just two or three (e.g., Griewank function f_{11}). Number of local optima usually increases exponentially with the number of dimensions, which often makes their minimization extremely difficult.

ABC-IX has been compared with the standard ABC algorithm and some other ABC variants [5-8] that try to alter the explorative/exploitative properties of ABC, such as ABC with self-adaptive mutation (ABC-SAM) [5], cooperative ABC (CABC) [6], ABC with diversity strategy (DABC) [7] and global best guided ABC [8]. On each function, ABC-IX has made 30 independent runs and the mean of the best results are presented in Tables 1–5. At first, ABC-IX is compared with the basic ABC [1] and ABC-SAM [5] in Table 1 with $SN = 50$, maximum no. of function evaluations FE = 100,000, no. of employed and onlooker bees = $SN/2$, no. of scout bees = 1 and $limit$ = 100. Table 1 shows that ABC-IX outperforms both ABC and ABC-SAM on most of the functions. In comparison to ABC, ABC-IX performs either better (on 11 out of the 13 functions) or at least equally well (f_6 and f_{13}). The other algorithm, ABC-SAM performs better than ABC-IX on one function only (f_8), while ABC-IX performs better on as many as 11 other functions. ABC-SAM performs better than the basic ABC, because it uses two different distributions, one exploitative and the other explorative, ensuring both explorations and exploitations. But it still uses an exploitative, fitness-based selection, which may be the reason that the more explorative ABC-IX often outperforms it. Fig. 4 shows that ABC-IX starts with more explorations and hence, reduced convergence speed than ABC, but gradually ABC-IX becomes more exploitative and achieves higher convergence speed. For f_9, the basic ABC is trapped during the last half of its execution, while ABC-IX shows no sign of stagnation.

In Table 2, ABC-IX is compared with two cooperative ABC variants – CABC_S and CABC_H [6]. Both the variants enforce more explorations by decomposing the search space and using multiple bee colonies to explore the multiple subspaces. For comparison, ABC-IX is implemented with the same settings [6]. Results show that ABC_IX outperforms CABC on four out of the six functions, while the CABC variants perform better on the remaining two functions only.

The next two comparisons of ABC-IX are with DABC [7] and GABC [8]. DABC tries to ensure more search space explorations by preserving more population diversity, while GABC alters the basic perturbation formula (1) in an exploitative way by using the global best solution found so far. ABC-IX is re-implemented with the same

settings as suggested in [7] and [8]. Tables 3-4 show that ABC-IX often outperforms DABC (2 out of 4 functions) and GABC (4 out of 5 functions). The reason may be that GABC is too exploitative, while DABC uses a naïve strategy of repeated switching that may cause frequent oscillations between exploitations and explorations.

For more investigations, we have designed two more variants of ABC – ABC-SimAn and ABC-SAD. ABC-SimAn includes the simulated annealing based selection but excludes the self-adaptive perturbation. In contrast, ABC-SAD includes the self-adaptive perturbation, but does not use the explorative simulated annealing based selection. Both of them are tested on the six multimodal functions f_8–f_{13} with SN = 100 and MCN = 1000. Results (Table 5) show that both ABC-SimAn and ABC-SAD outperform the basic ABC on most (5 out of 6) of the functions. This indicates the necessity of more explorations. Furthermore, ABC-IX, which combines both the explorative techniques, outperforms both ABC-SimAn and ABC-SAD on all the functions, which indicates that the synergy and interaction between the explorative selection and perturbation operations can be further useful to improve the results.

5 Suggestions for further study

There may be several future research directions suggested by this study. Firstly, ABC-IX uses a simple exponential cooling schedule for the system temperature T. A more sophisticated cooling strategy (e.g., a strategy parameterized by the population diversity or current explorative/exploitative requirement) may be more effective. Secondly, ABC-IX concentrates only on the explorative capacity of the algorithm. Putting some emphasis to control the exploitations may further improve the results. Thirdly, the quality of the final solution might be improved further by using an efficient and exploitative local searcher. Finally, ABC-IX has been applied mainly on continuous optimization problems. It would be interesting to study how well ABC-IX performs on many other existing problems, especially the discrete and real world ones.

Table 1. Comparison of ABC [1], ABC-SAM [5] and ABC-IX. Boldface font marks the best performance for each function.

No	ABC	ABC-SAM	ABC-IX
f_1	3.58E-11	4.18E-14	**2.86E-38**
f_2	1.04E-14	2.47E-08	**6.52E-18**
f_3	2.75E-10	3.95E-12	**1.86E-36**
f_4	9.37E+00	1.69E+01	**1.17E-02**
f_5	2.75E+00	4.27E-02	**1.95E-01**
f_6	**0**	**0**	**0**
f_7	8.61E-13	3.66E-16	**1.64E-63**
f_8	3.49E+02	**1.53E+02**	1.56E+02
f_9	5.79E-15	1.26E-16	**6.14E-41**
f_{10}	3.08E-06	9.26E-08	**3.82E-15**
f_{11}	4.35E-08	8.36E-10	**9.70E-40**
f_{12}	5.82E-08	2.78E-12	**7.40E-14**
f_{13}	**2.64E-03**	2.96E-01	**2.61E-03**

Table 2. Comparison of CABC variants [6] and ABC-IX. Best results are in boldface font.

No	CABC_S	CABC_H	ABC-IX
f_1	3.30E-19	5.92E-18	**9.41E-48**
f_5	6.33E+00	4.80E-01	**3.21E-07**
f_8	**1.30E-04**	**1.27E-04**	1.86E-01
f_9	**0**	**0**	3.86E-52
f_{10}	1.83E-14	8.35E-15	**1.14E-16**
f_{11}	4.42E-02	7.96E-03	**3.85E-51**

Table 3. Comparison between DABC [7] and ABC-IX. Best results are in boldface font.

No	DABC	ABC-IX
f_1	1.08E-16	**2.82E-63**
f_5	**1.55E-05**	**1.59E-05**
f_9	**0.00E-306**	4.29E-73
f_{11}	1.11E-16	**7.60E-67**

Table 4. Comparison between GABC [8] and ABC-IX. Best results are shown in boldface font.

No	*D*	GABC	ABC-IX
f_1	30	4.17E-16	**2.75E-107**
f_5	2	**1.68E-04**	2.93E-03
f_9	30	9.47E-15	**9.20E-107**
f_{10}	30	3.21E-14	**4.14E-15**
f_{11}	30	2.96E-17	**5.68E-105**

Table 5. Comparison of ABC [1], ABC-SimAn, ABC-SAD and ABC-IX. Best results in bold.

No	ABC	ABC-SimAn	ABC-SAD	ABC-IX
f_8	7.52E+02	5.29E+02	8.90E+01	**2.15E+01**
f_9	5.02E-14	2.53E-17	6.85E-18	**8.71E-20**
f_{10}	3.04E-07	8.50E-09	8.37E-10	**8.89E-11**
f_{11}	9.84E-10	1.91E-19	3.78E-19	**2.07E-23**
f_{12}	7.18E-10	7.55E-14	5.09E-14	**1.91E-14**
f_{13}	2.63E-03	2.64E-03	2.63E-03	**2.61E-03**

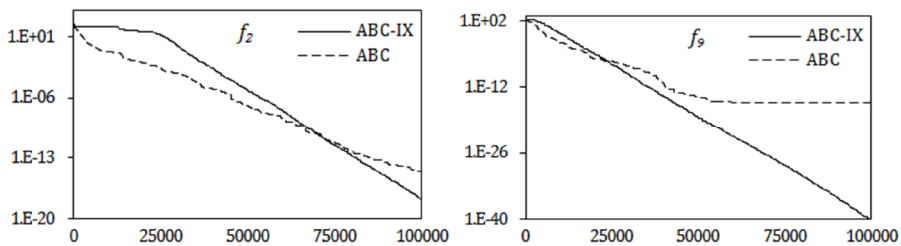

Fig. 4. Convergence of ABC and ABC-IX for the functions f_2 and f_9. The vertical axis shows the function value, while the horizontal axis is the number of function evaluations.

References

1. Karaboga, D., Basturk, B.: On the Performance of Artificial Bee Colony (ABC) Algorithm. Applied Soft Computing, vol. 8, pp. 687–697 (2008)
2. Karaboga, D., Akay, B.: A Comparative Study of Artificial Bee Colony Algorithm. Applied Mathematics and Computation, vol. 214, pp. 108–132 (2009)
3. Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N.: A Comprehensive Survey: Artificial Bee Colony (ABC) Algorithm and Applications. Artificial Intelligence Review, Online First™ (2012). DOI: 10.1007/s10462-012-9328-0.
4. Lampinen, J., Zelinka, I.: On Stagnation of the Differential Evolution Algorithm. In: Proc. 6th International Mendel Conf. on Soft Computing, Czech Republic, pp. 76-83 (2000)
5. Alam, M.S., Islam, M.M.: Artificial Bee Colony Algorithm with Self-Adaptive Mutation: A Novel Approach for Numeric Optimization. In: Proceedings of the 2011 Int. Conf. on Trends and Developments in Converging Technology, Bali, Indonesia, pp. 49–53 (2011)
6. Abd, M.E.: A cooperative Approach to the Artificial Bee Colony algorithm. In: IEEE Congress on Evolutionary Computation (CEC), Barcelona, Spain, pp. 1–5 (2010)
7. Lee, W.P., Cai, W.T.: A Novel Artificial Bee Colony Algorithm with Diversity Strategy. In: Proc. of the 7th Int. Conf. on Natural Computation, China, pp. 1441–1444 (2011)
8. Zhu, G., Kwong, S.: Gbest-guided Artificial Bee Colony Algorithm for Numerical Function Optimization. Applied Mathematics & Computation, vol. 217, 3166–3173 (2010)
9. Yao, X., Liu, Y., Lin, G.: Evolutionary Programming Made Faster. IEEE Transactions on Evolutionary Computation, vol. 3, pp. 82–102 (1999)