International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 9 – No.1, June 2015 – www.ijais.org

Balaning Explorations with Exploitations in the Artificial Bee Colony Algorithm for Numerical Function Optimization

Syeda Shabnam Hasan
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka-1208, Bangladesh

Fareal Ahmed
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka-1208, Bangladesh

ABSTRACT
This paper introduces a variant of Artificial Bee Colony algorithm and compares its results with a number of swarm intelligence and population based optimization algorithms. The Artificial Bee Colony (ABC) is an optimization algorithm based on the intelligent food foraging behavior of honey bees. The proposed variant, Artificial Bee Colony Algorithm with Balanced Explorations and Exploitations (ABC-BEE) makes attempts to dynamically balance the mutation step size with which the artificial bees explore the search space. Mutation with small step size produces small variations of existing solutions which is better for exploitations, while large mutation steps are likely to produce large variations that facilitate better explorations of the search space. ABC-BEE fosters both large and small mutation steps as well as adaptively controls the step lengths based on their effectiveness to produce better solutions. ABC-BEE has been evaluated and compared on a number of benchmark functions with the original ABC algorithm, Genetic Algorithm (GA), Particle Swarm Optimization (PSO) and Particle Swarm Inspired Evolutionary Algorithm (PS-EA). Results indicate that the proposed scheme facilitates more effective mutations and performs better optimization outperforming all the other algorithms in comparison.

Keywords
Artificial bee colony algorithm; Mutation; Exploration and exploitation; Function optimization.

1. INTRODUCTION
Swarm-based optimization algorithms (SOAs) employ a population of decentralized, self-organized agents and model some means of communication and information sharing among them to materialize a co-operative distributed search towards some optimal solution. Such an approach often mimics nature's methods to drive a distributed search to achieve some objective. Utilizing a population of agents who travel through the search space in parallel, SOAs exhibit remarkable robustness against being trapped in local optima, even in multimodal, high dimensional search space. This is a key advantage of SOAs over direct search algorithms such as hill climbing or random walk. SOAs can find reasonably good quality solutions within relatively short computation time. SOAs include Genetic Algorithm (GA) [14], Ant Colony Optimization (ACO) [15], Particle Swarm Optimization (PSO) [16], Bee Colony Optimization [1]-[5], [6] and so on. All of these population-based search algorithms employ some strategy to generate variations of the existing population of solutions to obtain a new offspring population. For example, GA [14], [17] applies genetic operators, like crossover and mutation on the existing individuals to alter them with the intention of obtaining a new population with better fitness. The individuals with better fitness enjoy greater chance to be selected for mating (i.e., recombination or crossover) and re-insertion to the offspring population. Thus GA tries to mimic the natural process of 'survival of the fittest' by providing privilege to the fitter individuals for selection and mating. Another swarm intelligence approach, Particle swarm optimization (PSO) is a stochastic optimization technique based on the social behavior and interactions in bird flocking or fish schooling. In PSO, individuals of the population move across the solution space like a group of interacting 'particles' and search for improved solutions. Each particle has a velocity and it changes its position and velocity based on the experience of itself and its neighbors. There exist several variants of the PSO algorithm based on various aspects of the base algorithm. The enormous adaptability of PSO to hybridizations and variations is considered its strength over many other algorithms. A hybrid approach combining PSO and Evolutionary Algorithm (EA) is the Particle swarm inspired Evolutionary Algorithm (PS-EA) [21]. PS-EA employs PSO, but tries to avoid generating infeasible solutions from the improper updates of PSO by heuristics with selection and mutation operations.

The intelligent food foraging behavior of honey bees has inspired several models to solve both combinatorial and continuous optimization problems [2-13]. A novel routing algorithm, BeeHive, is introduced by Wedde et al. [13] based on the communicative and dynamically evaluative procedures of honey bees. Teodorović proposed Bee Colony Optimization metaheuristic (BCO) [8] which can be employed to solve deterministic combinatorial problems, as well as combinatorial problems with uncertainty [5]. Lucic and Teodorović also demonstrated how the intelligent bee swarm behavior can be used to solve complex problems in traffic and transportation [6]. Tereshko and Loengarov considered bees as identical autonomous robots and presented experiments exhibiting robust and successful bee algorithms in complex robotic tasks [22]. Drias et al. introduced a metaheuristic, called Bees Swarm Optimization (BSO) and applied it to solve the maximum weighted satisfiability (max-sat) problem [9]. Benatchba et al. introduced a metaheuristic derived from the reproduction process of honey bees to solve a 3-sat problem [23]. Algorithms inspired by bee behavior have also been employed for solving Traveling Salesman Problem [24], Generalized Assignment Problem [25], Job shop scheduling [26], Training neural networks [12], dynamic allocation of internet servers [7] and so on. Relatively fewer works have

been carried out on continuous and numerical optimization problems. Yang developed a virtual bee algorithm (VBA) to solve the numeric function optimizations [10]. To minimize functions with two-parameters, Yang employed a swarm of virtual bees that started by randomly moving around the search space and interacting when they find any good quality nectar represented by the value of the objective function better than some predefined threshold. The optimal solution is obtained from the intensity of bee interactions. VBA is tested on two functions with two parameters, one is unimodal and the other is multimodal. Results show that VBA is much efficient than the genetic algorithm. Pham et al. [11] proposed a bee inspired optimization algorithm, the Bees Algorithm, which is applicable to both combinatorial and functional optimization problems. It is applied on eight benchmark functions and results show that it outperforms deterministic simplex method, stochastic simulated annealing optimization procedure, genetic algorithm and ant colony system, in terms of both final solution quality (i.e., accuracy) and convergence speed (number of iterations). Karaboga has developed an artificial bee colony (ABC) algorithm [4] which has fewer parameters and employed it for optimizing multivariable functions. Basturk and Karaboga presented another variant of ABC algorithm, extended the experimental results of the original ABC algorithm on five multi-dimensional benchmark functions and compared the results with genetic algorithms [2], [3]. In this paper, we have further improved the basic ABC algorithm by introducing ABC with Balanced Explorations and Exploitations (ABC-BEE) which incorporates a mutation step size adjustment scheme within the basic ABC algorithm. From the experiments, we found that ABC-BEE provides better results on most of the benchmark functions, which indicates the effectiveness of the proposed adjustment scheme.

The rest of this paper is organized as follows. Section 2 describes the basic ABC algorithm. The proposed variant ABC-BEE is presented in the following section 3. Section 4 provides details of the benchmarking problems, parameter settings of the different algorithms and compares the results. Section 5 draws conclusion of this study and leaves a few suggestions as future research directions.

2. THE ABC ALGORITHM

Honey bees in a colony show remarkable self-organization and co-ordination skills in their food foraging behavior. Bees have to forage over a vast area in search of good sources of food. After an initial exploration stage, more bees are employed to collect honey from the more profitable food sources whereas fewer bees are assigned to the less worthy food sources. Some scout bees are also assigned for exploration to find newer food sources. If the quality of a food source declines after some exploitation, this information is also shared with other bees so that fewer bees are now attracted to this source. After the quality of a food source falls below some threshold, the bees assigned to it abandon it. The foraging process is initiated by scout bees that start searching for flower patches suitable as food sources. Quality is measured as a combination of some values, such as quantity and density of sugar, ease of access, distance from the colony etc. After they return to the hive, those scout bees that found a patch with quality above some threshold, deposit their nectar and then go to the 'dance floor' to perform a dance known as the 'waggle dance'. This dance plays the key role to communicate information among the bees about the food sources. The waggle dance contains three pieces of information: i) the quality of the flower patch of this dancing bee, ii) the distance of the flower patch from the hive, iii) the direction from the hive that you have to travel in order to reach the flower. The 'onlooker' bees, waiting around the dance floor, observe the waggle dances of these 'employed' bees that have found good food sources and pick any one of them to become its 'follower' and collect nectar from its flower patch. The better a flower patch as a food source, the bigger is the number of follower bees along with its employed bee. However, if the patch is no longer good enough, it will not be advertised in the next waggle dance and the bees recruited for it as employed or follower bees will choose either to follow some other employed bee or start working as a scout bee to randomly explore the search space for finding new food source.

The ABC algorithm mimics the food foraging behavior of the honey bees with these three groups of bees: employed bees, onlookers and scouts. A bee working to forage a food source (i.e. solution) previously visited by itself and searching only around its vicinity is called an employed bee. Employed bees perform waggle dance to propagate information of its food source to other bees. A bee waiting around the dance floor to choose any of the employed bees to follow is called an onlooker. A bee randomly searching a search space for finding a food source is called a scout. For every food source, there is only one employed bee and a number of follower bees. The scout bee, after finding a good food source also becomes an employed bee. In ABC algorithm implementation, half of the colony is employed bees and the other half is the onlookers. Number of food sources (i.e., solutions) is equal to the number of employed bees. An employed bee whose food source is exhausted (i.e. solution has not improved after several attempts) becomes a scout. The detailed pseudocode is given below.

Step 1) Generate an initial population of N individuals. Each individual is a food source (i.e. solution) and has D attributes, where D is the dimensionality of the problem.

Step 2) Evaluate the fitness of each individual.

Step 3) Each employed bee searches in the neighborhood of its current position to find a better food source. For each employed bee, generate a new solution, $\boldsymbol{v_i}$ around its current position, $\boldsymbol{x_i}$ using (1).

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

Here, $k \in \{1, 2, \ldots, N_{emp}\}$ and $j \in \{1, 2, \ldots, D\}$ are randomly chosen indices. N_{emp} is the number of employed bees. Φ_{ij} is a uniform random number generated from the range [-1, 1].

Step 4) Compute the fitness of both $\boldsymbol{x_i}$ and $\boldsymbol{v_i}$. Apply greedy selection scheme to choose the better one.

Step 5) Calculate the selection probability, P_i for each solution, $\boldsymbol{x_i}$ and normalize by

$$P_i = fit_i \Big/ \sum_{k=1}^{N} fit_k \qquad (2)$$

Step 6) Assign each onlooker bee to a solution, $\boldsymbol{x_i}$ at random with probability proportional to P_i

Step 7) Produce new food positions (i.e. solutions), $\boldsymbol{v_i}$ for each onlooker bee using the corresponding employed bee $\boldsymbol{x_i}$ by using (1).

Step 8) Evaluate the fitness of each employed bee, $\boldsymbol{x_i}$ and its produced onlooker bee, $\boldsymbol{v_i}$. Apply greedy selection scheme to keep the one with better fitness and discard the other.

Step 9) If a particular solution has not been improved over a number of cycles, then select it for abandonment. Replace it by placing a scout bee at a food source placed uniformly at random over the search space using (3), i.e., for $j = 1, 2, ..., D$

$$x_{ij} = min_j + \text{rand}(0,1) * (\max_j - \min_j) \qquad (3)$$

Step 10) Keep track of the best food source position (solution) found so far.

Step 11) Check for termination. If the best solution found is acceptable or maximum number of iterations has elapsed, stop and return the best solution found so far. Otherwise go back to step 2 and repeat.

3. ADJUSTMENT OF MUTATION STEP SIZE WITH ABC-BEE

The proposed variant, ABC-BEE is different from the original ABC algorithm in three aspects. First, ABC-BEE alters (1) which is used by steps 3 and 7 of the original ABC algorithm for reproducing new solutions from the existing ones. Second, ABC-BEE executes an 'adjustment phase' periodically after every t generations to automatically adapt the magnification factors, i.e., the M_{ij} values that it maintains for every dimension, j of every bee, $\boldsymbol{x_i}$. Third, ABC-BEE employs a larger interval of [–3, 3] instead of the narrower [–1, 1] interval used by the original ABC algorithm. All these three schemes work together to facilitate more effective mutations to produce better offspring. The magnification factors that ABC-BEE maintains and adapts periodically help the mutation process perform better exploitations and explorations across the search space. Thus (1), which conducts the mutation process, now gets tuned by the M_{ij} values that try either to enlarge or to shrink the $\Phi_{ij} \in [-3, 3]$ values to facilitate exploration or exploitation respectively, as illustrated in (4).

$$v_{ij} = x_{ij} + M_{ij} * \varphi_{ij} * (x_{ij} - x_{kj}) \qquad (4)$$

All M_{ij} values are initiated to 2.0 during the beginning of the search process. As the search progresses across several peaks and plateaus or flat regions of the fitness landscape, the M_{ij} values are automatically adjusted by the periodically invoked adjustment phases in order to take care of the current situation. Small values (less than unity) for M_{ij} would shrink the product $M_{ij} * \Phi_{ij}$ in (4) to facilitate small mutation steps and thus ensure exploitation around the existing solution, $\boldsymbol{x_i}$. On the contrary, large enough values for M_{ij} would expand the product $M_{ij} * \Phi_{ij}$ in order to promote large mutation steps which would help the search process quickly get rid of local optima and perform more explorations of the search space. Whether exploitation or exploration is better at current search stage might not be apparent or could not be predicted beforehand. So the adjustment phase executes periodically after each t generations, performs mutations with different range of M_{ij} values and promotes only those M_{ij} values that produce more successful mutations.

Fig. 1 presents the pseudocode of the mutation step size adjustment cycle that is invoked periodically after every t generations. In this cycle, ABC-BEE adapts the magnification factor values of all the bees involved in the optimization process. The adjustment phase generates two uniform random values v_1 and v_2 for each employed and onlooker bee from the two Gaussian distributions with (mean, std. dev.) set to $(0, \alpha_1)$ and $(0, \alpha_2)$ respectively. Now, for every individual bee $\boldsymbol{x_i}$, three different offspring solutions (i.e., new food sources) are generated by using (4): one by employing the existing value of M_{ij} and two more offspring by using $M_{ij} = 2^{-v1}$ and 2^{v2} respectively. Since α_1 is much smaller than α_2, hence the magnitude of v_1 is likely to be smaller than its counterpart v_2. Thus the magnification factor, $M_{ij} = 2^{-v1}$ would generate small steps for better exploitations, while $M_{ij} = 2^{v2}$ would produce large steps for more search space explorations. Now, ABC-BEE evaluates the fitness of the three offspring and accepts the best one for the next generation. Also, it updates the M_{ij} to the weighted average of its current value and the magnification factor value that has produced the best offspring (say, $M_{ij}[k]$) using the following formula.

$$M_{ij} = \beta * M_{ij} + (1-\beta) * M_{ij}[k] \qquad (5)$$

However, the adjustment phase, as presented in Fig. 1, often causes the M_{ij} values to gradually decrease with generations, because exploitative small mutation steps usually have better chance to succeed than explorative larger steps. The continuous decrement of M_{ij} values shrinks the mutation steps which eventually might make the search process get trapped at locally optimal points. To avoid this, ABC-BEE adopts a simple, yet effective scheme. If a particular M_{ij} drops continuously over the last s_1 generations, and is never increased over these last s_1 generations without any fitness improvement of the solution $\boldsymbol{x_i}$, then ABC-BEE manually resets M_{ij} to 1.0 and keeps it constant at 1.0 for the next s_2 generations (otherwise, M_{ij} may quickly revert back to its smaller values again because of the weighted averaging). This resetting of magnification factors to higher values promotes larger mutation steps and helps get rid of any local optimum whenever any solution gets trapped there.

```
Procedure Adjustment Phase
for each bee i
  for j = 1 to D
  {
    v1 = absolute (Gaussian_Random (0, α1))
    v2 = absolute (Gaussian_Random (0, α2))
    Mij [1]= 2^-v1
    Mij [2]= 2^v2
    Mij [3]= Mij
    for count = 1 to 3
    {
      vij [count] = vij computed from (4), but
                    with Mij =Mij [k]
      fitness[count] = Fitness (vij [count])
    }
    Find k such that fitness[k] is the minimum
      over the array, fitness[1...3]
    Mij = β * Mij + (1- β) * Mij [k]
  }
```

Fig. 1: Pseudo code of the mutation step size adjustment phase, which is invoked periodically after every t generations

4. EXPERIMENTS

4.1 Benchmark functions

A function is called multimodal if it has multiple local optima. In order to minimize such a function, the search process must be able to avoid being trapped around the locally optimal points. The difficulty increases with the dimensionality of the problem, because the number of local minima increases exponentially with the number of dimensions. To compare the performance of the proposed ABC-BEE with the original ABC, PSO, PS-EA and GA, we employ five multimodal benchmark functions [21], [27] each with dimensionality set to 10, 20, and 30.

Table 1 shows the benchmark functions used for comparison. The first function, f_1 is the multimodal Griewank function. It reaches the global minimum value of 0 when the variables are (0, 0, …, 0). Initialization range of the variables is [−600, 600]. The product term present in f_1 introduces strong interdependence among the variables. So the techniques which try to optimize each variable separately without considering the others will fail with this function. The curse of high-dimensionality, combined with its regularly distributed multimodality makes the minimization process quite difficult for any algorithm.

The second function is the Rastrigin function, f_2. The first term in f_2 [27] comes from the sphere function, while the second term with cosine product introduces regularly distributed multimodality. The initialization range is set to [-15, 15]. The function reaches the value of 0 at its unique global minimum (0, 0, …, 0). The third function is the Rosenbrock function which reaches 0 at the global minimum (1, 1, …, 1). The global minimum is inside a narrow parabolic shaped flat valley. The variables are strongly dependent and gradients usually do not direct to the global minima. Both the functions are considered very challenging and have repeatedly been used with the optimization algorithms. The fourth function is the Ackley function which has the minimum value of 0 at its global minimum (0, 0, …, 0). Its initialization range is [−32.768, 32.768]. The exponential term with cosine sum introduces numerous local minima. Any algorithm using only gradient steepest descent will be trapped in a local minimum. So the algorithm has to combine both explorative and exploitative schemes to reach the global minimum avoiding numerous local minima during optimization. The fifth function is Schwefel function which has the minimum value of 0 at its global minimum (420.9867, 420.9867, …, 420.9867). Its initial range is [−500, 500]. The function has plenty of peaks and valleys. Unlike the previous functions, the global minimum is near the edge of the search space. Moreover, the function has a second best minimum which is far-away from the unique global minimum. This causes a great difficulty during the optimization process and many search algorithms get trapped in the second best minimum.

4.2 Parameter Settings for the algorithms

GA, PSO, PS-EA, ABC and ABC-BEE — all have two parameters in common: the population size and maximum number of generations. The population size has been set to 125 and the maximum number of generations is set to 500, 750 and 1,000 for the problems with dimensions of 10, 20 and 30 respectively, as specified in [21].

Settings for GA, PSO and PS-EA: The particular GA scheme we employed is specified in [21] with its parameter values. It includes single point uniform crossover, crossover rate of 0.95, uniform random selection, linear ranking fitness function, Gaussian mutation and mutation rate of 0.1. The particular version of PSO we used is based on two distinct equations with three parameters — w, φ_1 and φ_2, as in [21]. Here, φ_1 and φ_2 are learning rates and w is the inertia weight. As specified in [21], φ_1 and φ_2 are set to be 2.0 and w is varied linearly with iterations from 0.9 to 0.7. Another algorithm we employed is Particle Swarm Inspired Evolutionary Algorithm (PS-EA). PS-EA is a hybrid approach employing techniques from both the fields of PSO and EA. PS-EA performs an updating operation of each individual in the population using the Inheritance Probability Tree (PIT). This is part of the 'Self-updating Mechanism' (SUM) of PS−EA. SUM may dynamically adjust the inheritance probabilities in PIT. This Dynamic Inheritance Probability Adjustment (DIPA) is implemented by SUM considering the convergence rate of the algorithm during particular iterations. An initial infeasible set of inheritance probabilities, as suggested in [21], are used to test the performance of the DIPA module in PS-EA.

Settings for ABC and ABC-BEE: We used colony size of 125 with the percentage of both employed and onlooker bees set to 50% of the colony. The number of scout bees is set to 1. Maximum number of generations is set to 500, 750 and 1000 for the problems with dimensions of 10, 20 and 30 respectively. The number of fitness evaluations in an adjustment phase is three times than that of a typical generation, so each adjustment phase of the ABC-BEE algorithm is counted as three generations. This ensures a fair comparison between ABC and ABC-BEE by allowing equal number of fitness evaluations by both the algorithms. The adjustment phase is invoked after every t = 25 generations. With some preliminary experiments, other parameters are set as: s_1 = 25, s_2 = 10, α_1= 0.50, α_2=4.0 and β = 0.15.

4.3 Experimental Results

GA, PSO, PS-EA, ABC, and ABC-BEE — each is run 50 times on every benchmark function. The mean of the best function values over all the runs for each function is presented in Table 2. Results indicate that ABC-BEE outperforms other algorithms by several orders of magnitude for most instances. It is remarkable that ABC-BEE reaches very close proximity of the global minima for all the functions, while the basic ABC algorithm fails to reach sufficiently close to the global minima on a number of occasions, such as the high dimensional Schwefel and Rosenbrock functions. Although ABC outperforms ABC-BEE for 30D Ackley function (f_4), the performance difference is not statistically significant, as we have found in t-test with 95% degree of confidence.

It is also observed that ABC-BEE has much better rate of convergence for most of the functions. To evaluate the convergence rates quantitatively, we now measure the number of generations that is required by both ABC and ABC-BEE to locate the global minima. Locating the global minimum $\boldsymbol{x}^*$is defined as reaching a function value $f(\boldsymbol{x})$ such that $|f(\boldsymbol{x}) - f(\boldsymbol{x}^*)| \leq 10^{-1}$. Table 3 shows that ABC-BEE locates the global minima with such accuracy faster than the basic ABC algorithm. For some functions, such as f_3 and f_5, ABC often fails to reach such proximity of the global minima within 1000 generations, which is marked as "*failed*" in Table 3. In contrast, ABC-BEE often locates the global minima within a few hundred generations. The only instance where ABC shows little bit faster convergence is the Ackley function, f_4, but this difference is not statistically significant, as we have found in t-test (not shown) with 95% degree of confidence.

How much the proposed adjustment phase can improve the mutation operation? To find out, we now compare the successful mutation rates achieved by ABC and ABC-BEE. Here 'success rate' of a mutation scheme is defined as the percentage of better offspring produced by that scheme. Results from Table 4 demonstrate that the percentage of successful mutations is often much higher with ABC-BEE in comparison to ABC. In some instances, such as f_2, f_3 and f_5, the success rate by ABC-BEE is almost double of ABC.

These success rates nicely coincides with the results in Table 2, because the optimization performance (presented in Table 2) of ABC-BEE is most noticeably better than ABC for these same three functions, i.e., f_2, f_3 and f_5. Such a beautiful correlation between better results and higher mutation success rates directly indicates that the proposed adjustment phase, as employed by ABC-BEE, yields relatively better offspring by mutation step size adjustments and thus contributes towards the improvement of the performance of the algorithm.

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

No	Function	*D*	*S*	*C*	f_{min}
f_1	Griewank	10, 20, 30	$[-600, 600]^D$	*MN*	0
f_2	Rastrigin	10, 20, 30	$[-5.12, 5.12]^D$	*MS*	0
f_3	Rosenbrock	10, 20, 30	$[-30, 30]^D$	*UN*	0
f_4	Ackley	10, 20, 30	$[-32, 32]^D$	*MN*	0
f_5	Schwefel	10, 20, 30	$[-500, 500]^D$	*MS*	0

Table 2. Mean of the best function values found over 50 independent runs for each function by GA, PSO, PS-EA, ABC and the proposed ABC-BEE algorithm. The best result for each function is shown in boldface font.

Function	*D*	GA	PSO	PS-EA	ABC	ABC-BEE	Best Results By
f_1	10	0.050228	0.079393	0.222366	8.7×10^{-4}	**9.45×10^{-14}**	ABC-BEE
	20	1.0139	0.030565	0.59036	2.10×10^{-8}	**1.08×10^{-14}**	
	30	1.2342	0.011151	0.8211	2.87×10^{-9}	**7.82×10^{-12}**	
f_2	10	1.3928	2.6559	0.43404	**0**	**0**	ABC ABC-BEE
	20	6.0309	12.059	1.8135	1.45×10^{-8}	**3.13×10^{-13}**	
	30	10.4388	32.476	3.0527	0.033874	**6.59×10^{-9}**	
f_3	10	46.3184	4.3713	25.303	0.034072	**1.32×10^{-7}**	ABC-BEE
	20	103.93	77.382	72.452	0.13614	**7.80×10^{-5}**	
	30	166.283	402.54	98.407	0.219636	**2.99×10^{-4}**	
f_4	10	0.59267	9.85×10^{-13}	0.19209	7.8×10^{-11}	**3.09×10^{-14}**	ABC ABC-BEE
	20	0.92413	1.178×10^{-9}	0.32321	1.6×10^{-11}	**1.79×10^{-13}**	
	30	1.0989	1.491×10^{-6}	0.3771	**3×10^{-12}**	3.16×10^{-12}	
f_5	10	1.9519	161.87	0.32037	1.27×10^{-9}	**6.63×10^{-18}**	ABC-BEE
	20	7.285	543.07	1.4984	19.83971	**9.10×10^{-10}**	
	30	13.5346	990.77	3.272	146.8568	**0.009081**	

Table 3. Number of generations required by ABC and ABC-BEE to reach the global minimum. Results are averaged over 50 independent runs.

Function	Dimensionality	ABC	ABC-BEE	Better Convergence By
f_1	30	238.5	**181.8**	**ABC-BEE**
f_2	30	319.6	**210.7**	**ABC-BEE**
f_3	30	*Failed*	**491.0**	**ABC-BEE**
f_4	30	**183.3**	186.9	**ABC**
f_5	30	*Failed*	**835.5**	**ABC-BEE**

Table 4. Comparison of successful mutation rates achieved by ABC and ABC-BEE. Results are averaged over 50 independent runs.

Function	Dimensionality	Successful Mutation Rate (%)		Better Performance By
		ABC	ABC-BEE	
f_1	30	18.2	**23.8**	**ABC-BEE**
f_2	30	11.2	**19.5**	**ABC-BEE**
f_3	30	6.8	**12.6**	**ABC-BEE**
f_4	30	**25.1**	22.3	**ABC**
f_5	30	5.5	**10.9**	**ABC-BEE**

5. CONCLUSION AND FUTURE RESEARCH DIRECTIONS

This paper introduces ABC-BEE, an improvement over the basic ABC algorithm incorporating an automatic adjustment phase for the mutation step size. ABC-BEE is evaluated on a number of benchmark functions and is compared with a number of population-based and swarm intelligence algorithms. Experiments and empirical results from ABC-BEE clearly suggest the effectiveness of adopting the proposed mutation step size adjustment phase which employs Gaussian and Exponential distributions to produce both large and small steps and then picks the more successful step lengths. However, there are several future research directions suggested by this study. First, ABC-BEE mostly follows a greedy management of the magnification factors (i.e., the M_{ij} values), as it tends to travel towards the magnification factor values that lead to immediate fitness improvements. A more sophisticated and less greedy adjustment phase may further improve its performance. Second, ABC-BEE demonstrates excellent capacity to locate the global optima, so one interesting idea would be to hybridize ABC-BEE with other existing algorithms. ABC-BEE could be employed on a problem that is partially solved by another algorithm while the global optimum is still unknown. It would be interesting to find out whether ABC-BEE can improve the final solution quality. Third, ABC-BEE has been applied only to the continuous function optimization problems. It would be interesting to study how well ABC-BEE performs for other optimization problems, especially the discrete and real world optimization problems.

6. REFERENCES

[1] D. Karaboga, B. Basturk, "On the performance of artificial bee colony (ABC) algorithm", in *Applied Soft Computing*, vol. 8, no. 1, pp. 687-697, 2008.

[2] D. Karaboga, B. Basturk, "Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems", in *Proceedings of the 12th international Fuzzy Systems Association world congress on Foundations of Fuzzy Logic and Soft Computing*, June 18-21, 2007, Cancun, Mexico.

[3] B. Basturk, D. Karaboga, "An artificial bee colony (ABC) algorithm for numeric function optimization", in *Proceedings of the IEEE Swarm Intelligence Symposium 2006*, Indianapolis, Indiana, USA, 12–14 May 2006.

[4] D. Karaboga, "An idea based on honey bee swarm for numerical optimization", Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[5] P. Lucic, D. Teodorovic, "Vehicle Routing Problem with Uncertain Demand at Nodes: The Bee System and Fuzzy Logic Approach", in *Fuzzy Sets in Optimization*, Editor J.L. Verdegay, Springer-Verlag, Berlin Heidelbelg, pp. 67-82, 2003**.**

[6] P. Lucic, D. Teodorovic, "Bee system: Modeling Combinatorial Optimization Transportation Engineering Problems by Swarm Intelligence", in *Preprints of the TRISTAN IV Triennial Symposium on Transportation Analysis*, Sao Miguel, Azores Islands, pp. 441-445,2001.

[7] S. Nakrani, C. Tovey, "On Honey Bees and Dynamic Allocation in an Internet Server Colony", *Proceedings of 2nd International Workshop on the Mathematics and Algorithms of Social Insects*, Atlanta, Georgia, USA, 2003.

[8] D. Teodorovic, M. Dell'Orco, "Bee Colony Optimization - A Cooperative Learning Approach to Complex Transportation Problems", in *Advanced OR and AI Methods in Transportation*, pp. 51-60, 2005.

[9] H. Drias, S. Sadeg, S. Yahi, "Cooperative Bees Swarm for Solving the Maximum Weighted Satisfiability Problem", *IWAAN International Work Conference on Artificial and Natural Neural Networks*, Barcelona, Spain, pp. 318-325, 2005.

[10] X. S. Yang, "Engineering Optimizations via Nature-Inspired Virtual Bee Algorithms", *IWINAC2005*, LNCS 3562, J. M. Yang and J.R. Alvarez (Eds.), Springer-Verlag, Berlin Heidelberg, pp. 317-323, 2005.

[11] D. T. Pham, E. Kog, A. Ghanbarzadeh, S. Otri, S. Rahim, M. Zaidi, "The Bees Algorithm – A Novel Tool for Complex Optimization Problems", *IPROMS 2006 Proceeding 2nd International Virtual Conference on Intelligent Production Machines and Systems*, Oxford, Elsevier, 2006.

[12] D. T. Pham, E. Koc, A. Ghanbarzadeh, S. Otri, "Optimization of the Weights of Multi-Layered Perceptions Using the Bees Algorithm", *in Proceedings of 5th International Symposium on Intelligent Manufacturing Systems*, Sakarya, Turkey, pp. 38-46, 2006.

[13] H. F. Wedde, M. Faruq, Y. Zhan, "BeeHive: An Efficient Fault-Tolerant Routing Algorithm Inspired by Honey Bee Behavior", *Ant Colony, Optimization and Swarm Intelligence*, Eds. M. Dorigo, Lecture Notes in Computer Science 3172, Springer Berlin, pp. 83-94, 2004.

[14] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading: Addison-Wesley Longman, 1989.

[15] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, 2004.

[16] R. Eberhart, Y. Shi and J. Kennedy. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, 2001.

[17] J. H. Holland. *Adaptation in Natural & Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[18] M. Mathur, S. B. Karale, S. Priye, V. K. Jayaraman, B. D. Kulkarni, "Ant Colony Approach to Continuous Function Optimization". *Ind. Eng. Chem. Res.* 39(10), 2000, 3814-3822.

[19] G. Bilchev and I. C. Parmee, "The Ant Colony Metaphor for Searching Continuous Design Spaces", in *Selected Papers from AISB Workshop on Evolutionary Computing*, 1995, pp. 25-39.

[20] M. Dorigo, G. D. Caro, L. M. Gambardella, "Ant algorithms for Discrete optimization", Artificial Life, vol. 5, no. 2, pp. 137-172, 1999.

[21] D. Srinivasan, T. H. Seow, "Evolutionary Computation", *CEC '03*, 8–12 Dec. 2003, 4(2003), Canberra, Australia, pp. 2292–2297.

[22] V. Tereshko, A. Loengarov, "Collective Decision-Making in Honey Bee Foraging Dynamics", *Comput. Inf. Sys. J.*, vol. 9, no. 3, pp. 1–7, 2005.

[23] K. Benatchba, L. Admane, M. Koudil, "Using bees to solve a data-mining problem expressed as a max-sat one, artificial intelligence and knowledge engineering applications: a bio-inspired approach". in *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005*, Las Palmas, Canary Islands, Spain, 15–18, June 2005.

[24] L. P. Wong , M. Y. H. Low , C. S. Chong, "A Bee Colony Optimization Algorithm for Traveling Salesman Problem", *Proceedings of the 2008 Second Asia International Conference on Modelling & Simulation (AMS)*, pp. 818-823, May 13-15, 2008.

[25] A. Baykasoglu, L. Ozbakor and P. Tapkan, "Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem", chapter 8 in *Swarm Intelligence, Focus on Ant and Particle Swarm Optimization*, I-Tech Education and Publishing, Vienna, Austria, 2007, ISBN 978-3-902613-09-7.

[26] C. S. Chong, A. I. Sivakumar, M. Y. H. Low , K. L. Gay, "A bee colony optimization algorithm to job shop scheduling", *Proceedings of the 38th conference on Winter simulation*, December 03-06, 2006, Monterey, California.

[27] X. Yao, Y. Liu, and G. Lin, "Evolutionary Programming Made Faster", *IEEE Transactions on Evolutionary Computation*, Vol-3, No. 2, 1999.