An Artificial Bee Colony Algorithm with History-Driven Scout Bees Phase

Xin Zhang[1] and Zhou Wu[2(✉)]

[1] College of Electronic and Communication Engineering,
Tianjin Normal University, Tianjin, China
xinzhang9-c@my.cityu.edu.hk
[2] Department of Electrical Electronic and Computer Engineering,
University of Pretoria, Hatfield, Pretoria, South Africa
wuzhsky@gmail.com

Abstract. The scout bees phase of artificial bee colony (ABC) algorithm emulates a random restart and cannot make sure the quality of the solution generated. Thus, we propose to use the entire search history to improve the quality of regenerated solutions, called history-driven scout bee ABC (HdABC). The proposed algorithm has been tested on a set of 28 test functions. Experimental results show that ABC cannot significantly outperforms HdABC on all functions; while HdABC significantly outperforms ABC in most test cases. Moreover, when the number of restarts increases, the performance of HdABC improves.

Keywords: Artificial bee colony · Search history · Binary space partitioning tree

1 Introduction

Artificial bee colony (ABC) is a simple and powerful metaheuristic for solving global optimization problems [1]. It is based on the intelligent behavior of honey bees. Many researchers have tried to improve its performance and make it better. For example, researchers propose to use the global best solution found so far to generate candidate solutions [2], [3]. Inspired by particle swarm optimization, Zhu et al. propose Gbest-guided ABC (GABC) algorithm [4]. Kang et al. propose Rosenbrock ABC which combines Rosenbrock's rotational direction method with the ABC algorithm [5]. Zhang et al. propose one-position inheritance and opposite directional search methods respectively for the employed bees phase and onlooker bees phase [6]. Karaboga et al. create a quick ABC algorithm which imitates the behavior of onlooker bees in a better way than standard ABC [7]. Kiran et al. modify ABC with a directed method [8]. Applications of discrete variants of ABC include [9], [10], [11], [12].

None of the above mentioned algorithms focus on the scout bees phase. Actually, the scout bees phase can be seen as a random search; the quality of the regenerated solution is unpredictable and low quality solution causes a waste of resources. Thus, we will concentrate on modifying the scout bees phase to improve the performance of the ABC algorithm. In this paper, we propose a novel ABC algorithm which uses the entire search history to improve the quality of solutions, called History-driven scout

© Springer International Publishing Switzerland 2015
Y. Tan et al. (Eds.): ICSI-CCI 2015, Part I, LNCS 9140, pp. 239–246, 2015.
DOI: 10.1007/978-3-319-20466-6_26

bee Artificial Bee Colony (HdABC). It has been noticed that history is a good reference to help the search. There are already some works on applying the entire search history in the EA field [13], [14], [15]. In our proposed algorithm, we apply BSP tree to improve the performance of the ABC algorithm.

The paper is organized as follows. Section 2 describes standard ABC algorithm. Section 3 explains the proposed HdABC algorithm in detail. Experimental results are shown in Section 4 and Section 5 gives the conclusion.

2 Artificial Bee Colony Algorithm

Standard ABC algorithm can be divided into four phases as follows:

1. Initialization Phase
   In the initialization phase, a population NP of solutions (food sources), i.e., $\mathbf{x}_i = \{x_{i,1}, x_{i,2}, x_{i,3}, \dots, x_{i,D}\}$, is initialized randomly in the search space.
2. Employed Bees phase
   Each employed bee randomly communicates with another employed bee to search a new location, i.e., $\mathbf{v}_i = \{v_{i,1}, v_{i,2}, v_{i,3}, \dots, v_{i,D}\}$. The equation to generate new location is shown in (1).

$$v_{i,j} = x_{i,j} + \phi_{i,j}\{x_{i,j} - x_{k,j}\} \ . \tag{1}$$

   where the indices $j \in \{1,2, \dots, D\}$ and $k \in \{1,2, \dots, NP\}, k \neq i$ are randomly generated. A coefficient $\phi_{i,j}$ is a random number between [-1, 1].
   The employed bees evaluate the new food source $\mathbf{v}_i$, and compared with their current food source $\mathbf{x}_i$ by the fitness of solutions. The equation to calculate the fitness is shown in (2), where $f(\mathbf{x}_i)$ represents the objective value of the solution $\mathbf{x}_i$.

$$fit(\mathbf{x}_i) = \begin{cases} \frac{1}{1+f(\mathbf{x}_i)}, & if\ f(\mathbf{x}_i) \geq 0 \\ 1 + abs(f(\mathbf{x}_i)), & if\ f(\mathbf{x}_i) < 0 \end{cases} \ . \tag{2}$$

3. Onlooker Bees phase
   Onlooker bees receive the information from the employed bees, and make decision on selecting some food sources for further search. By using the equation shown in (3), the probability p_i is calculated by the fitness of the food sources. The onlooker bees go to the better food sources with higher probability.

$$p_i = \frac{fit_i}{\sum_{n=1}^{NP} fit_n} \ . \tag{3}$$

4. Scout bees phase
   During the search, some food sources will be abandoned. A user defined parameter *limit* is introduced to control when to abandon a food source. The employed bee of the abandoned food sources will become scout bee. A scout bee searches a new food source randomly to replace the abandoned food source.

3 The Proposed Algorithm

3.1 Idea of the Proposed Algorithm

In most of optimization problems, except that with illness condition, the landscape of function is generally smooth. Thus starting at any point, if we move towards the nearest optimal point, we may get a better solution. In this paper, all the evaluated solutions and their fitness are memorized. By these search history, we estimate the fitness landscape of the function. By that, the estimated local optimal point of any solution can be found. Then we can find the better restarted solutions by moving them towards the local optimal point. Therefore the efficiency of ABC algorithm can be improved.

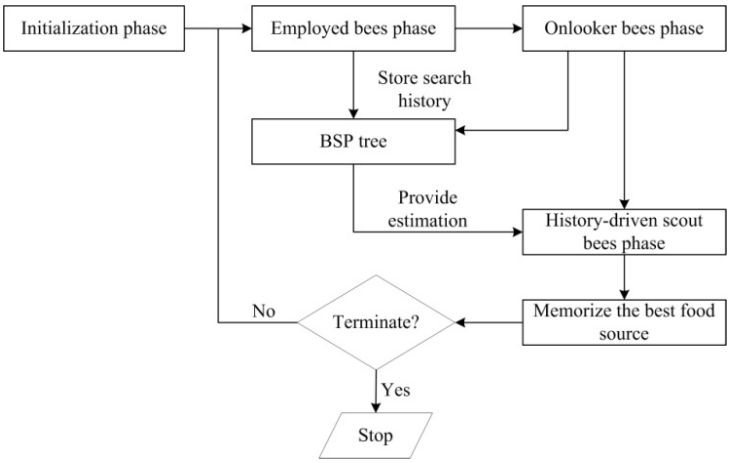

Fig. 1. Block diagram of HdABC

3.1 Structure of the Proposed Algorithm

The proposed algorithm has two main parts: 1) memory, and 2) history-driven scout bees algorithm. The function of the memory is to store the entire search history. The entire search history is used to build the estimated fitness landscape of the objective function (surrogate model), which can be used to estimate the fitness of solutions. Fig. 1 shows the block diagram of our proposed algorithm. In the beginning, the location of food sources (solutions) are randomly generated and evaluated by the objective function. Once a solution is evaluated, it will be stored with its fitness value in BSP tree. Then, starts the cycle of employed bees phase, onlooker bees phase and our proposed history-driven scout bees phase. The differences between standard ABC algorithm and our HdABC are that: HdABC stores all the evaluated solutions in BSP tree and substitutes history-driven scout bees phase for the scout bees phase.

3.2 History-Driven Scout Bees Algorithm

Binary space partitioning (BSP) tree is built as a surrogate model for providing fitness estimation. In [14], Chow and Yuen apply the idea of the nearest-neighbor search into BSP tree. They stored both the evaluated solution and their fitness into BSP tree. In this paper, we use the same tree as that in [14], [15].

To improve the quality of restarted solutions, the guided anisotropic search (GAS) module [14] is used in this paper. The GAS module is a novel parameter-less adaptive mutation operator, which applies a randomized gradient descent-like method to mutate solutions towards the better solutions. Equation (4) shows the equation used to generate the new solution **v** by using solution **x**, and the corresponding local optimal point **p**.

$$\mathbf{v} = \mathbf{x} + \alpha(\mathbf{p} - \mathbf{x}) \quad . \tag{4}$$

In the equation, **x** is guided by the direction (**p**-**x**), it moves towards **p** with mutation step sizeα. To balance the exploitive effect of the gradient descent-like direction assignment, the value of α is a random number within the range [0,1] with uniform distribution. Therefore, the solution **v** lies on the line determined by **p** and **x**.

In the proposed history-driven scout bees algorithm, the GAS is used in two different ways: 1) local restart and 2) global restart. The usage of local restart is the same as the GAS, which mutates the abandoned solutions towards the nearest local optimal region. Different from the equation (3) that is used to generate solutions in ABC, GAS provides a multi-dimensional mutation to the ABC algorithm. Comparing to the local search method used in standard ABC, GAS allows solutions moving in a relatively wider area and helps solutions in ABC converge faster to a better area. However, when the ABC continuous running, the number of solutions stored in BSP tree increases, the sizes of sub-regions in BSP tree decrease. In this case, the area for the local restart will be decreased. Therefore, we introduce another usage of the GAS, which is global restart, to override this problem. The global restart works like an enhanced version of global search. It firstly random regenerates a new solution in the whole search space, and then applies the GAS to mutate it towards to a better area, which helps the search to escape from the local optimal and ensures the new solution has a relatively better quality. Nevertheless, the function of global restart is mainly focus on preventing the premature convergence. It is expected that the improvement is not significant when there is only global restart applied into the ABC algorithm. Therefore, in the proposed history-driven scout bees algorithm, we introduce a parameter *r* to control the ratio of using local restart and global restart, where *r* is increasing with the number of generations:

$$r = \frac{number\ of\ generation}{Total\ number\ of\ generation} \quad . \tag{5}$$

In each time a food source is abandoned, *r* will be calculated by (5). The history-driven restarted algorithm generates a random number$d \in [0, 1]$ to make decision on performing local restart or global restart, i.e.,

$$\text{operation} = \begin{cases} local\ restart, & \text{if } d \geq r \\ global\ restart, & \text{if } d < r \end{cases} \quad . \tag{6}$$

In the beginning of the search, the value is relatively smaller. The history-driven scout bees algorithm concentrates more in local restart, which helps the search converge faster. Then, at the middle of the search, the ratio of performing local restart and global restart is 1:1, which balances the exploration and exportation in the search. At the later stage of the search, the search is converged. Therefore the history-driven scout bees algorithm focuses on the global restart to prevent the premature convergence.

4 Experiment

In this paper, 28 real valued benchmark functions f_1-f_{28} are used to test the performance of test algorithms as shown in Table 1. All the functions are tested in $D = 30$. Each test algorithm tries to solve each of the test function 30 times. For the maximum number of evaluation in each trial, we run experiment with two different values: one is $MFEs$=5000D, and the other is $MFEs$=10000D.

To verify the performance of the proposed HdABC algorithm, it is compared with standard ABC. To compare the performance of test functions, the Mann-Whitney Utest (U-test) is used. The detailed setup of the test algorithms is NP=25 and $limit$=100.The values of NP and $limit$ follow [2], [3], [6].

Table 1. Test function set

f_1	Sphereical	f_{15}	Pathological
f_2	Schwefel's Problem 2.22	f_{16}	InvertedCosineWave
f_3	Schwefel's Problem 1.02	f_{17}	InvertedCosineMixture
f_4	Schwefel's Problem 2.21	f_{18}	EpistaticMichalewicz
f_5	Rosenbrock	f_{19}	LevyMontalvo2
f_6	Quartic	f_{20}	Neumaier3
f_7	Generalized Rastrigin	f_{21}	OddSquare
f_8	Generalized Griewank	f_{22}	Paviani
f_9	Schwefel's Problem 2.26	f_{23}	Periodic
f_{10}	Ackley	f_{24}	Salomon
f_{11}	High Conditioned Elliptic	f_{25}	Shubert
f_{12}	Levy	f_{26}	Sinusoidal
f_{13}	Zakharov	f_{27}	Michalewicz
f_{14}	Alpine	f_{28}	Whitely

Tables 2 shows the average values of the best fitness, standard deviation and p-values of the test algorithms found in 30 independent runs. To increase the readability, the best results are typed in **boldface**. The performance of standard ABC is compared with HdABC by using U-test to determine the significance. The p-value shows the result of the U-test. By definition, the result is said to be significant if p<0.05. A marker "*" is added to the p-values which shows the result is significant. Table 8 and Table 9 illustrates the mean values of number of scout bees done by the test algorithms in 30 independent runs with $MFEs$ =150000 and $MFEs$ = 300000, respectively.

From Table 2, compared with ABC, the performance of HdABC is clearly better than ABC. HdABC obtains the best results in total 23 out of 28 functions. Among these 23 functions, the result is significant in 14 functions. Besides, HdABC performs

worse than original ABC in only 5 functions, but all of them are not significant. It shows that HdABC outperforms standard ABC in most cases.

In fact, the difference between standard ABC and HdABC is that HdABC replaces the original scout bees phase to the proposed algorithm. Therefore, the results shown in Table 2 are directly reflecting the contribution made by the proposed algorithm. The proposed algorithm gives a significant improvement to original ABC, it makes the HdABC greatly outperforms standard ABC. To further prove that the proposed algorithm improves the original ABC, we have run another experiment with *MFEs* = 300000. In general, with the larger *MFEs*, the maximum number of generations will be larger. Thus more employed bees will be abandoned during the search, and then the contribution of our proposed algorithm should be shown clearly.

Table 2. The mean standard deviation (Std. D) and *p*-value (*p*) of the best function values found by ABC and HdABC in f_1–f_{28}, $MFES = 5000D$

	f_1	f_2	f_3	f_4	f_5	f_6
ABC	6.99E-16	3.55E-16	5.28E+03	3.89E+01	3.52E-01	1.20E+01
Std.D	*1.08E-16*	*8.46E-16*	*1.38E+03*	*5.86E+00*	*6.06E-01*	*5.17E-01*
HdABC	**2.38E-16**	**2.96E-16**	**5.09E+03**	**7.02E+00**	**1.99E-01**	**9.74E+00**
Std.D	*8.40E-17*	*6.62E-16*	*1.47E+03*	*1.50E+00*	*2.71E-01*	*6.17E-01*
p	*3.69E-11	9.64E-01	5.59E-01	*3.02E-11	7.39E-01	*3.69E-11
	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}
ABC	1.07E-14	**5.59E-14**	**-1.38E+04**	5.81E-14	7.51E-16	6.71E-16
Std.D	*2.29E-14*	*1.96E-13*	*1.63E-02*	*7.15E-15*	*1.61E-16*	*1.25E-16*
HdABC	**2.61E-15**	3.87E-13	-1.38E+04	**5.22E-14**	**7.02E-16**	**2.41E-16**
Std.D	*8.02E-15*	*1.31E-12*	*2.37E-02*	*6.40E-15*	*1.61E-16*	*1.05E-16*
p	*1.60E-03	7.73E-01	6.20E-01	*2.00E-03	1.41E-01	*3.69E-11
	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}
ABC	2.19E+02	**1.09E-08**	4.45E+00	-2.54E+01	-8.73E-16	-2.41E+01
Std.D	*3.02E+01*	*2.75E-08*	*3.41E-01*	*8.63E-01*	*2.69E-16*	*6.63E-01*
HdABC	**1.54E+02**	1.74E-08	**3.81E+00**	**-2.63E+01**	**-9.92E-16**	**-2.57E+01**
Std.D	*3.49E+01*	*5.68E-08*	*4.63E-01*	*1.04E+00*	*2.96E-16*	*6.03E-01*
p	*8.35E-08	8.53E-01	*1.16E-07	*5.56E-04	1.10E-01	*4.20E-10
	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}	f_{24}
ABC	**4.20E-03**	-2.63E+03	-2.80E-03	-9.98E+05	1.00E+00	1.73E+00
Std.D	*2.17E-02*	*8.55E+02*	*1.17E-02*	*5.18E-10*	*1.26E-05*	*2.41E-01*
HdABC	3.35E-02	**-3.78E+03**	**-5.90E-03**	**-9.98E+05**	**1.00E+00**	**1.40E+00**
Std.D	*1.75E-01*	*5.42E+02*	*1.30E-02*	*5.37E-10*	*6.04E-06*	*1.49E-01*
p	2.28E-01	*7.04E-07	2.06E-01	2.49E-01	2.06E-01	*7.60E-07
	f_{25}	f_{26}	f_{27}	f_{28}		
ABC	**-2.32E+34**	-1.74E+00	-2.96E+01	1.33E+02		
Std.D	*5.05E+33*	*8.23E-01*	*2.17E-02*	*1.04E+02*		
HdABC	-2.29E+34	**-3.50E+00**	**-2.96E+01**	**1.30E+02**		
Std.D	*4.01E+33*	*1.70E-03*	*2.72E-02*	*9.89E+01*		
p	8.42E-01	*6.52E-09	*4.51E-02	9.59E-01		

The performance of ABC and HdABC in the case of *MFEs* = 300000 are not shown in this paper for the saving of space. The results show that ABC gives the same

performance as HdABC in 3 functions. It illustrates that both of them obtain the global optimal solution. As it is no differences between ABC and HdABC in these 3 functions, we simply discard these results. Compared to ABC in the rest of 25 functions, HdABC obtains the best results in 22 out of 25 functions. Among these 22 functions, HdABC performs significantly better than ABC in 17 functions. In only 3 test functions, ABC outperforms HdABC, but the results of all of them are not significant. Compared to the results in Table 2, it is clearly shown that the number of the test functions that HdABC outperforms ABC is significantly increased, and the number of the test functions that HdABC performs worse than ABC is decreased. The experimental results prove that the proposed algorithm brings a positive effect to the original ABC.

Table 3. The mean of the number of scout bees done by ABC and HdABC in f_1–f_{28} ,*MFES* = 5000*D* and 10000*D*, respectively

5000*D*	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
ABC	50	25	2.87	559	4.87	312	25.1	30.3	2.43	25.1
HdABC	81.4	28	3.17	572	4.73	518	35.8	46.4	2	32.3
	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}
ABC	25.1	50	15.8	0.2	97.1	37.7	50	102	20.1	15.3
HdABC	36.1	80.7	14.2	0.17	103	40.7	75.1	103	24.3	18.7
	f_{21}	f_{22}	f_{23}	f_{24}	f_{25}	f_{26}	f_{27}	f_{28}		
ABC	76.6	50	26.7	85.3	32.7	26.9	6.17	57.8		
HdABC	95.4	63.7	21.7	137	26	34.6	4.83	67.8		
10000*D*	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
ABC	96	48	8.7	982	21.6	591	68.1	72.6	26.2	48.3
HdABC	184	56.3	8.2	1030	24.9	1003	102	132	38.9	75.2
	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}
ABC	72.1	96	31.1	24.2	193	78.5	96.1	198	48.7	34
HdABC	105	193	27.6	32.6	202	88.7	181	200	80.4	53.4
	f_{21}	f_{22}	f_{23}	f_{24}	f_{25}	f_{26}	f_{27}	f_{28}		
ABC	141	96	57.5	169	63.3	55.4	31	114		
HdABC	187	148	63.1	290	59.9	101	50.7	133		

Table 3 shows the mean of numbers of scout bees done by ABC and HdABC in the 28 test functions. Through comparison, it shows that with the larger value of *MFEs*, the number of scout bees is significantly increased.

5 Conclusion

History-driven scout bees Artificial Bee Colony (HdABC) algorithm is proposed. It uses a memory archive called Binary Space Partitioning tree to store the entire search history, and applies the Guided Anisotropic Search (GAS) module to find a better solution. The proposed algorithm contains two parts. One is for global search and the other is for local search. Experimental results show that the use of entire search history and the GAS module bring positive effects to ABC algorithm.

For the future direction, the GAS module can be applied to employed bees phase or onlooker bees phase to further improve the performance of ABC algorithm. Hybrid gravitational evolution [16], [17] or neighborhood field optimization [18] methods with ABC are also interesting.

References

1. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithms. Journal of Global Optimization. **39**, 459–471 (2007)
2. Diwold, K., Aderhold, A., Scheidler, A., Middendorf, M.: Performance evaluation of artificial bee colony optimization and new selection schemes. Memetic Computing. **3**, 149–162 (2011)
3. Zhang, X., Zhang, X., Ho, S.L., Fu, W.N.: A modification of artificial bee colony algorithm applied to loudspeaker design problem. IEEE Transactions on Magnetics. **50**, 737–740 (2014)
4. Zhu, G., Kwong, S.: Gbest-guided artificial bee colony algorithm for numerical function optimization. Applied Mathematics and Computation. **217**, 3166–3173 (2010)
5. Kang, F., Li, J., Ma, Z.: Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical function. Information Sciences. **181**, 3509–3531 (2011)
6. Zhang, X., Zhang, X., Yuen, S.Y., Ho, S.L., Fu, W.N.: An improved artificial bee colony algorithm for optimal design of electromagnetic devices. IEEE Transactions on Magnetics. **49**, 4811–4816 (2013)
7. Karaboga, D., Gorkemli, B.: A quick artificial bee colony (qABC) algorithm and its performance on optimization problems. Applied Soft Computing **23**, 227–238 (2014)
8. Kiran, M.S., Findik, O.: A directed artificial bee colony algorithm. Applied Soft Computing **26**, 454–462 (2015)
9. Ozturk, C., Hancer, E., Karaboga, D.: Dynamic clustering with improved binary artificial bee colony algorithm. Applied Soft Computing **28**, 69–80 (2015)
10. Ozturk, C., Hancer, E., Karaboga, D.: Improved clustering criterion for image clustering with artificial bee colony algorithm. Pattern Analysis and Applications, 1–13 (2014)
11. Pan, Q.K., Wang, L., Li, J.Q., et al.: A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan inimisation. Omega **45**, 42–56 (2014)
12. Cui, Z., Gu, X.: An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems. Neurocomputing **148**, 248–259 (2015)
13. Yuen, S.Y., Chow, C.K.: A genetic algorithm that adaptively mutates and never revisits. IEEE Transactions on Evolutionary Computation. **13**, 454–472 (2009)
14. Chow, C.K., Yuen, S.Y.: An Evolutionary Algorithm that Makes Decision Based on the Entire Previous Search History. IEEE Transactions on Evolutionary Computation **15**, 741–769 (2011)
15. Leung, S.W., Yuen, S.Y., Chow, C.K.: Parameter control system of evolutionary algorithm that is aided by the entire search history. Applied Soft Computing. **12**, 3063–3078 (2012)
16. Lou, Y., Li, J., Shi, Y., Jin, L.: Gravitational co-evolution and opposition-based optimization algorithm. International Journal of Computational Intelligence Systems **6**, 849–861 (2013)
17. Lou, Y., Li, J., Jin, L., Li, G.: A coEvolutionary algorithm based on elitism and gravitational evolution strategies. Journal of Computational Information Systems **8**, 2741–2750 (2012)
18. Wu, Z., Chow, T.W.S.: Neighborhood field for cooperative optimization. Soft Computing **17**, 819–834 (2013)