Turkish Journal of Engineering

https://dergipark.org.tr/en/pub/tuje

e-ISSN 2587-1366

Honey formation optimization: HFO

Zeki Yetgin *[1], **Mustafa Şamdan** [2]

[1]*Mersin University, Faculty of Engineering, Department of Computer Engineering, Mersin, Turkey*
[2]*Mersin University, Faculty of Engineering, Information Technologies Research and Application Center, Mersin, Turkey*

Keywords
ABC
Bee colony algorithm
Honey formation
Function decomposition

ABSTRACT

In this paper, a new optimization framework, namely Honey Formation Optimization (HFO), is proposed. In contrary to the Artificial Bee Colony Optimization (ABC) variants in literature, the HFO considers food sources consisting of many components and model the honey formation inside bees as a process of mixing the components with their special enzymes during chewing up the food source. We believe that bees analyze the amounts of components inside the food source and attempt more to collect weaker (less amount) components to improve the honey formation process. Thus, each time a worker exploits a food source it selects a component in such a way that weaker components are more frequently selected. The approach requires decomposing the solution into components where each component is evaluated by a component fitness function. The honey formula maps the component fitness to honey amount and considered as the equivalence of the fitness function. The worker bee uses the fitness of the selected component to evaluate the food source and does local search only around the selected component. The HFO and ABC Frameworks are compared on the basis of 9 benchmark functions. The result shows that HFO performs better than the ABC.

1. INTRODUCTION

Artificial Bee Colony (Karaboga 2005) is inspired by the intelligent behavior of honey bees. Scout, worker and onlooker bees form a colony and cooperatively search for food source positions. In ABC algorithm, scouts find initial positions of the food sources and then they are converted to workers. Workers exploit these sources and announce the information about them to onlooker bees in hive. Onlooker bees pay more visits to the better food sources and exploit them in the same way as workers do. Exploiting a food source means local search around the source and keeping track of the better food source. ABC algorithm has an increasing popularity in scientific community. It has been applied in solving many problems, such as image enhancement (Chen et al. 2017), compression (Ismail and Baskaran 2014), motion estimation (Cuevas et al. 2013), network attacks (Lozano et al. 2017), intrusion detection (Aldwairi et al. 2015), training neural networks (Karaboga et al. 2007), feature selection (Keles and Kilic 2018), clustering (Karaboga and Ozturk 2011), and among many others(Akay and Karaboga 2015; Liu et al. 2017; Apalak et al. 2014; Abro and Mohamad-Saleh 2014; Kang et al. 2013). In literature, ABC algorithm is initially proposed for optimization of numeric functions (Karaboga 2005). Since then many ABC variants (Karaboga et al. 2014; Jia et al. 2015; Huang et al. 2016) have been proposed for various type of optimizations such as constrained, multi objective, continuous and combinatorial design problems. Karaboga and his friends provided a comprehensive survey (Karaboga et al. 2014) that analyses these problems with the focus on ABC drawbacks. According to the survey studies, the great potential of ABC seems very clear with its good exploration capability but also a strong need to alleviate the weakness in exploitation capability (local search).

Majority of the ABC versions in literature focus on the exploit phase (Gao and Liu 2012; Wang et al. 2019; Han et al. 2015; Shah et al. 2014; Cheng and Jiang 2012; He et al. 2015; Chen et al. 2012; Kang et al. 2011),which improve the local search capability of bees (workers or onlookers). The original bees search around the current solution for a random neighbor towards to one of the existing solutions. Some articles (Gao and Liu 2012) (Shah et al. 2014) allow bees to search around the best

* **Corresponding Author**

*(zyetgin@mersin.edu.tr) ORCID ID 0000 – 0001 – 5918 – 6565
(msamdan@mersin.edu.tr) ORCID ID 0000 – 0003 – 4079 – 4565

Cite this article

Yetgin Z & Şamdan M (2021). Honey formation optimization: HFO. Turkish Journal of Engineering, 5(2), 81-88

Research Article / DOI: 10.31127/tuje.693103

Received: 23/02/2020; Accepted: 21/04/2020

solution of the current population. Although such approaches enable bees to converge the optimal solution very soon, they increases possibility of local stuck around the near optimal solution. Thus, (Gao and Liu 2012) also improves exploration ability of the scout bees by using chaotic and opposition based initialization (Sun et al. 2018) allows bees to search around a random existing solution for random neighbor towards to current solution, which is quite opposite of the original approach. Authors claim that this approach can expand the search range of new solution and further improve the exploration ability of ABC algorithm.

All the aforementioned ABC algorithms assume single component inside a food source and no honey formation inside bees. We considered that the honey formation starts when the bees chew up the food source, e.g. nectar, with their special enzymes and meanwhile they can analyze which components are needed to improve the quality of honey. The HFO Algorithm is actually a framework that can be applied to any ABC algorithms. It requires four major changes from existing ABC versions: i) solution decomposition into components where the component fitness function is composed from the cost function ii) honey formula mapping the component fitness to the solution fitness iii) a selection strategy for worker bees to select the weaker component randomly iv) applying local update only to the selected component. No change in onlooker phase is required. However optionally, instead of using fitness function, honey formula could still be used in onlooker phase. Also when the number of component is one, HFO becomes equal to original ABC algorithm.

The article is organized as follows. Next section provides the original ABC. Third section provides the proposed HFO algorithm. The fourth section provides the experimental results and the last section gives concluding remarks and future directions.

2. ABC ALGORITHM

The basic ABC algorithm requires few parameters, such as the number of food sources denoted as NS, maximum iterations denoted as *MaxIter*, and the *trial* limit denoted as *limit*. As given in Algorithm 1, the ABC algorithm has three phases. In first phase, scout bees randomly explore the food space to find initial food sources, which are the initial solutions denoted as X=($\boldsymbol{x_1}$, $\boldsymbol{x_2}$,…,$\boldsymbol{x_{NS}}$) and formulated in Eq. (1).

$$x_i(j) = x_{\min(j)} + rand(0,1).\left(x_{\min(j)} + x_{\max(j)}\right) \quad (1)$$

where j is the updating dimension $\in \{1, 2,\ldots,D$, x_{min} and x_{max} are upper and lower bound solutions respectively. Any scout discovering the food source, source $x_i \in R^D$ becomes a worker bee with its associated food source in its memory. The bees measure the quality of the source x_i using the fitness function, formulated on basis of the cost function *f(x)* in Eq. (2).

$$fitx_i = fitness(x_i) = \begin{cases} 1/(1+f(x_i)), & f(x_i) > 0 \\ 1 + abs(f(x_i)), & f(x_i) \le 0 \end{cases} \quad (2)$$

In second phase, each worker *i* locally updates its food source x_i as a result of randomly searching its neighbourhood for a better solution. This phase is equivalent to local update procedure defined in Eq. (3-4), which forms a candidate $v_i \in R^D$ by updating a randomly selected dimension j of the solution x_i towards one of the existing solution $x_k \neq x_i$ formulated in Eq. (3). The workers replace the source x_i with the candidate v_i if later is better, formulated in Eq. (4). Local update procedure also updates a trial counter. If worker *i* cannot (update) improve its current solutionx_i, the trial counter c_i will be incremented, otherwise the counter is reset to zero.

$$v_i(j) = x_i(j) + rand(-1,1).\left(x_i(j) - x_k(j)\right) \quad (3)$$

$$\begin{cases} x_i = v_i, \quad c_i = 0, & fitv_i > fitx_i \\ c_i = c_i + 1, & fitv_i \le fitx_i \end{cases} \quad (4)$$

where $k \in \{1, 2, \ldots, NS\}$ is the randomly chosen indice and $k \neq i, j \in \{1, 2, \ldots, D\}$ indicates a random dimension selected to be updated. In third phase, workers announce the information about the food sources such as nectar amount and position by dancing in the hive. Onlooker bees watch the dances of these bees and select a random food source among the sources in such a way that better sources have more chance to be selected.

Algorithm 1. Basic ABC Algorithm (MaxIter, NS, limit): return Best

(1) Generate random NS solutions (Eq. 1)
(2) for iter =1 to MaxIter do
(3) for each worker:
(4) - apply local update procedure to the associated solution of the worker (Eq. 3-4)
(5) P←selection probabilities of solutions proportional to their fitness values (Eq.2, 5)
(6) for each onlookers:
(7) - select a random solution according to selection probability P
(8) - apply local update procedure to it (Eq. 3-4)
(9) for each scouts:
(10)- replace the associated solution with a random solution if the solution is not I updated for limit iteration (Eq. 1)
(10) keep track of Best solution so far

The selection probabilities, $P = \{p_1, p_2,\ldots,p_{NS}\}$, are formulated in Eq. (5) where p_i is the selection probability of the source x_i.

$$p_i = \frac{fitx_i}{\sum_{k=1}^{NS} fitx_k} \quad (5)$$

Then, they exploit their food sources in the same fashion as workers do (local update procedure). Thus,

onlookers mostly gather around globally better solutions to make global improvements. All bees keep track of the best food sources during searching for the sources. Any bee that cannot exploit its food source (improve its solution) within some trial limit becomes scout again and finds a random food source throughout the space. The algorithm repeats worker-onlooker-scout phases until the maximum cycles are completed. When algorithm terminates one of the bees in current population is expected to have the best food source in its memory.

3. HFO FRAMEWORK

The proposed HFO framework is given in Algorithm 2 below. HFO generalizes the ABC algorithm where single component assumption equalizes the both. The main difference is that the HFO assumes food sources each consisting of K components and worker bees attempting more to search for components that are needed according to current honey form inside bees. Every food source $\boldsymbol{x_i}$ has its own honey form produced from it. Thus, the food source and its honey form are associated.

Algorithm 2. HFO Framework (MaxIter, NS, limit, K): return Best

(1)	Generate random NS solutions	(Eq.1)
(2)	for iter =1 to MaxIter do	
(3)	for each worker:	
(4)	- P ← selection probabilities of components in current solution, inversely proportional to their component fitness	(Eq. 8)
(5)	- select a random component according to selection probability P	
(6)	- apply local update procedure to the selected component	(Eq. 9-10)
(7)	P← selection probabilities of solutions, proportional to their fitness	(Eq. 5, 12)
(8)	for each onlookers:	
(9)	- select a random solution according to selection probability P	
(10)	- apply local update procedure	(Eq. 3, 11)
(11)	for each scouts:	
(12)	- replace the associated solution with a random solution if the solution is not updated for limit iteration	(Eq.1)
(13)	keep track of Best solution so far	

The honey form or equivalently the food source is considered as a solution and the components of the food source are sub solutions. The HFO finally finds the source that produce the best honey form. HFO uses cost-based approach: instead of using fitness, cost values are used when comparing solutions or components. The HFO defines three design concepts:

i) Component Design that require to decompose the solution into components where *c.* component of the solution x_i is denoted as $x_i{}^c$

ii) Component Fitness Design that deals with how to approximately measure each component fitness in a given solution. Normally, in HFO Framework, component fitness design is required for each component. However, the design of component and its fitness depends on the problem. Here, we assume original cost function f as component fitness functions where the fitness of the component $x_i{}^c$ is denoted as $fitx_i{}^c$ while its cost is denoted as $f(x_i{}^c)$

iii) Honey Formula Design (Optional): If the solution fitness can be expressed as function of component fitness such that this function shows equivalence / approximation to the original fitness function, then we call this function as Honey Function. If there is no way for Honey Function, one can use the original fitness function as Honey Function. One advantage of using honey function is that the solution cost is computed in terms of component cost values $f(x_i{}^c)$ that are already computed during local update procedure. This reduces the complexity of HFO.

One major form for component design is given in Eq. (6) where a solution $x_i \in R^D$ is decomposed into K non-overlapping sub solutions, causing a shift from the space R^D to $R^{D/K}$, with D/K as the dimension of components. Components are separated by pipe symbols in Eq. (6) just for visualization.

Among many forms of solution decomposition, following shows an overlapping form of components where half of each component is overlapped with the neighbor components.

$$x_i = [\underline{x_{i1}{}^1, x_{i2}{}^1, \ldots x_{i(D/K)}{}^1} \,/\, \underline{x_{i(D/K+1)}{}^2, x_{i(D/K+2)}{}^2 \ldots x_{i(2D/K)}{}^2} \,/\, \ldots .] = [\, x_i{}^1 \,/\, x_i{}^2 \,/ \ldots \,] \quad (6)$$

$$\text{Xi}= [\frac{X_i^2}{X_i^1 X_i^3}, \ldots \ldots] \quad (7)$$

The solution should be decomposed into components in such a way that the honey formula can bind the component fitness to the original fitness function.

According to HFO, the worker *i* evaluates the components of x_i and more probably modify (local update) the weaker component due to the fact that the component in less amount are more vital and more needed to improve the current honey form. Let $x_i{}^c \subseteq x_i$ be the *c*. component of x_i.The selection probability of component *c* for the worker $\boldsymbol{i}$, denoted as P_i^c, is inversely proportional to its fitness, formulated in Eq. (8) where the component cost $f(x_i{}^c) = costx_i{}^c$ naturally measures the inverse fitness of the component *c* of x_i.

$$P_i^c = \begin{Bmatrix} \dfrac{f(x_i{}^c) + 1}{\sum_{j=1}^{K}(f(x_i{}^j) + 1)} \text{ if f(x) is in positive domain} \\ \dfrac{f(x_i{}^c) - \min\limits_{k=1..K}(f(x_i{}^k)) + 1}{\sum_{j=1}^{K}\left(f(x_i{}^j) - \min\limits_{k=1..K}(f(x_i{}^k)) + 1\right)} \text{ otherwise} \end{Bmatrix} \quad (8)$$

The local update procedure for worker bees is applied on component basis, which is formulated in Eq. (9-10) where c is the selected component and j is the updating dimension of the component c during local search aroundx_i. Note that the proposed local update procedure can be applied to any ABC variant not limited to Eq. (9-10). The idea here is the workers modifies the selected component according to their local search strategies.

$$v_i^c(j) = x_i^c(j) + rand(-1,1).\left(x_i^c(j) - x_k^c(j)\right) \quad (9)$$

$$\begin{cases} x_i = v_i, c_i = 0, & f(v_i^c) < f(x_i^c) \\ c_i = c_i + 1, & otherwise \end{cases} \quad (10)$$

When comparing two solutions x_i and v_i in onlooker phase, we also prefer to use cost function rather than fitness function since the fitness definition in original ABC may cause implementation issue related to infinite precision requirement at the term 1/(1+cost). Thus, the local update procedure for onlooker is modified using Eq. (11) as follows

$$\begin{cases} x_i = v_i, c_i = 0, & F(v_i) < F(x_i) \\ c_i = c_i + 1, & otherwise \end{cases} \quad (11)$$

where the $F(x)$ is the cost form of honey formula, formulated in Eq.(12). Honey formula F(x) is an approximation to the original cost function f(x) or must have equivalence relation with the f(x). Here as honey function we adapt summation operator.

$$F(x_i) = \sum_{c=1}^{K} f(x_i^c) \cong f(x_i) \quad (12)$$

The cost function approximation using the summation of component cost values are one form of honey formula, among many others.

The fitness form of honey formula $F_{fit}(x_i)$ is given in Eq.(13) that is only used in Eq.(14) to compute the selection probabilities p_i of solutions for onlookers.

$$F_{fit}(x_i) = \begin{cases} \frac{1}{F(x_i)+1} & if\ f(x)\ in\ positive\ domain \\ \frac{1}{F(x_i) - \min\limits_{k=1..NS} F(x_k) + 1} & otherwise \end{cases} \quad (13)$$

$$p_i = \frac{F_{fit}(x_i)}{\sum_{k=1}^{NS} F_{fit}(x_k)} \quad (14)$$

HFO does not require any change in onlooker bees except using cost function f(x) when comparing two solutions. However, the approximated version of cost function F(x) could optionally be used to benefit from cost function decomposition. Cost functions may not be easily decomposed into component cost functions. Some cost functions are separable and easily expressed in terms of component costs. Thus, one can consider cost function approximation if it allows cost function decomposition on components.

3.1. Some Forms for Component Design

The component cost functions $f(x_i^c)$ and component itself x_i^c must be considered together in design. Component design is problem specific and must be done for each benchmark functions. Here we propose some design strategies for component and its cost functions. Let the cost function f(x) expressed as f(x) = g(x) + h(x), 3 forms of component design are defined as follows:

1. Form: x_i^c is non-overlapped and partition on f(x) as follows: $f(x_i^c) = g(x_i^c) + h(x_i^c)$
2. Form: x_i^c is non-overlapped and partition on g(x) as follows: $f(x_i^c) = g(x_i^c) + h(x)/K$
3. Form: x_i^c is overlapped and partition on f(x) as follows: $f(x_i^c) = g(x_i^c) + h(x_i^c)$

4. EXPERIMENTAL RESULTS

ABC and HFO algorithms are compared based on 9 benchmark functions given in Table 1. The benchmark functions have different characteristics such as multimodal and non-convex (Ackley, Qing, Egg-Crate, Xin-She Yan, Rosenbrock), multimodal and convex(Rastrigin), unimodal and non-convex(Griewank), unimodal and convex(Brown, Sphere). The functions are tested for the maximum number of iterations *MaxIter = 5000*, the number of food sources *NS=60*, the problem dimension *D=50*, and the parameter *limit=NS×D* and the number of components *K=10* for HFO.

However with this limit setting, the Rastrigin function is reached to global min zero for both ABC and HFO, thus the *limit = 0.1xNS x D* is considered only for Rastrigin function. The colony has equal number of worker bees and onlooker bees, considered equal to NS. Each experiment for the same parameter settings repeated 20 times and the average values are used to compute the performance metrics, such as min, max and mean of cost function values. The component design for each benchmark functions are provided in Table 3 where the details of component design is given in previous section.

The experimental results are provided in Table 4 and Figs. 1-9 where the table demonstrates the comparison of the objective performances with achieved min, max and mean cost values and the figures show the evolution curves of ABC and HFO. The Table 4 clearly shows that HFO is superior to original ABC for all functions. Particularly, for unimodal and convex problems the HFO performs best. However, for many difficult functions such as Rosenbrock, Egg-Crate, and Xin-She Yan that are multimodal and non-convex the HFO also performs well.

The evolution curves of mean cost values across the iterations are given in Figs. 1-9. The figures clearly show that the HFO converge speed is also better than ABC. The ABC sometimes early converges to a mean cost value around 1.0e-16 due to its weak local search strategy. With ABC local search, the solution is randomly updated and this causes improvement in less speed. HFO causes more correct solution update by updating only the worse component of the moment.

For each benchmark functions the behaviors of ABC and HFO varies. In general, ABC early maturates for all functions except Ackley where HFO performs better but converged earlier than ABC. Looking the Figure 1, around

the iteration 3800-4000 HFO becomes saturated, however, ABC continue to improve. Due to early maturation for Ackley, all solutions in the population have sufficient time to reach the best value. Thus, the best and worst values are converged to each other for Ackley.

Table 2. Typical benchmark functions

Name	Function	Range	Min
Ackley	$f_1 = -20exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e$	(−32.768, 32.768)	0
Brown	$f_2 = \sum_{i=1}^{n-1} (x_i^2)^{(x_{i+1}^2+1)} + (x_{i+1}^2)^{(x_i^2+1)}$	(−4, 4)	0
Griewank	$f_3 = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})$	(−600, 600)	0
Qing	$f_4 = \sum_{i=1}^{n} (x_i^2 - i)^2$	(−10,10)	0
Rastrigin*	$f_5 = 10n + \sum_{i=1}^{n} (x_i^2 - 10cos(2\pi x_i))$	(-5.12, 5.12)	0
Sphere	$f_6 = \sum_{i=1}^{n} x_i^2$	(−100, 100)	0
Egg Crate	$f_7 = \sum_{i=1}^{n} x_i^2 + 25 \sum_{i=1}^{n} sin^2(x_i)$	(−500, 500)	0
Xin-SheYan	$f_8 = \left(\sum_{i=1}^{n} sin^2(x_i) - e^{-\sum_{i=1}^{n} x_i^2}\right) e^{-\sum_{i=1}^{n} sin^2\sqrt{\lvert x_i \rvert}}$	(−10,10)	-1
Rosenbrock	$f_9 = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$	(−50, 50)	0

Table 3. Component design for benchmark functions

Objective Functions	Component Overlapped	Component Cost Function Form#	Partition on g(x) or f(x)
Ackley	No	1	f(x)
Brown	Yes(1 element)	3	f(x)
Griewank	No	1	g(x)= $\sum_{i=1}^{n} \frac{x_i^2}{4000}$
Qing	No	1	f(x)
Rastrigin*	No	1	f(x)
Sphere	No	1	f(x)
EggCrate	No	2	g(x)= $\sum_{i=1}^{n} x_i^2$
Xin-SheYang	No	1	f(x)
Rosenbrock	Yes(1 element)	3	f(x)

Table 4. Performance comparison : ABC versus HFO

Objective Functions	ABC			HFO		
	Best	Mean	Worst	Best	Mean	Worst
Ackley	5.1e-14	6.3e-14	6.8e-14	8.0e-15	8.0e-15	8.0-15
Brown	7.5e-16	9.6e-16	1.2e-15	9.4e-48	3.2e-47	1.1e-46
Griewank	0	9.4e-17	1.1e-16	0	0	0
Qing	9.6e-16	2.7e-15	1.3e-14	2.3e-17	7.1e-16	7.2e-15
Rastrigin*	0	2.3e-14	1.1e-13	0	0	0
Sphere	7.7e-16	9.4e-16	1.1e-15	2.8e-44	1.7e-43	4.1e-43
EggCrate	6.5e-16	7.9e-16	9.9e-16	1.1e-34	5.8e-34	2.7e-34
Xin-SheYang	7.7e-17	9.7e-17	1.1e-16	3.5e-37	5.6e-33	3.1e-32
Rosenbrock	0.5e-02	0.2	0.8	2.2e-06	1.2e-05	4.4e-05

The Figure 3 and Figure 5 shows similar behaviors where there is a limit around 1e-15 and 1e-17 for Griewank and Rastrigin respectively and exceeding the limit causes their converge to zero. Another reason for similar behavior for both function is their similar function definitions where $\boldsymbol{g(x)} = \sum_{i=1}^{n} (\boldsymbol{x_i^2}/4000)$ and $h(x) = 1 - \prod cos(xi/\sqrt{i})$ are used for Griewank function where h(x) is defined in terms of cosine function and g(x) and h(x) could be zero. When we look at the Rastrigin where $g(x) = \sum_{i=1}^{n} (x_i^2)$ and $\boldsymbol{h(x)} = 10n - \sum_{i=1}^{n} 10cos(2\pi x_i)$ could be considered, h(x) is again defined in terms of cosine function, and g(x) and h(x) could be zero. Rastrigin seems simple for both ABC and HFO. ABC is also reaching to zero with normal limit parameter. Thus, we reduced the limit L for only Rastrigin.

5. CONCLUSION

In this article, a new optimization framework namely Honey Formation Optimization (HFO) is introduced. HFO extends the Artificial Bee Colony Algorithm by considering multiple components in food sources and worker bees searching more frequently for the components in less amount due to fact that the component in less amount more limits the honey formation process and thus more vital for worker bees. For single component assumption, HFO and ABC become equal. The proposed optimization is a framework that could be applied to any ABC variant. The components are considered as sub solutions and honey formation process

mix up the components towards better honey fitness. The experimental results demonstrates that HFO can performs better and converge earlier than the ABC. However, HFO uses honey formula that requires design for both solution decomposition and cost function decomposition together. Thus, some functions are difficult or even impossible to decompose perfectly. In such cases new approaches are required to partition functions on the basis of component design.

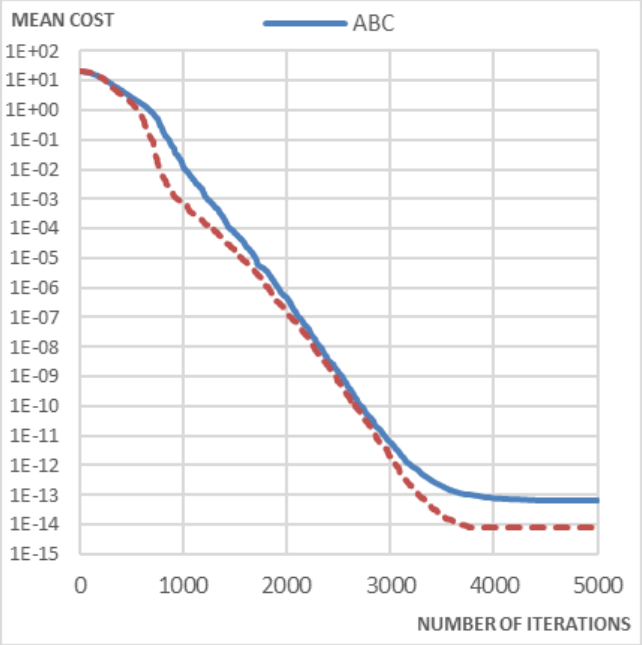

Figure 1. Evolution curves for Ackley

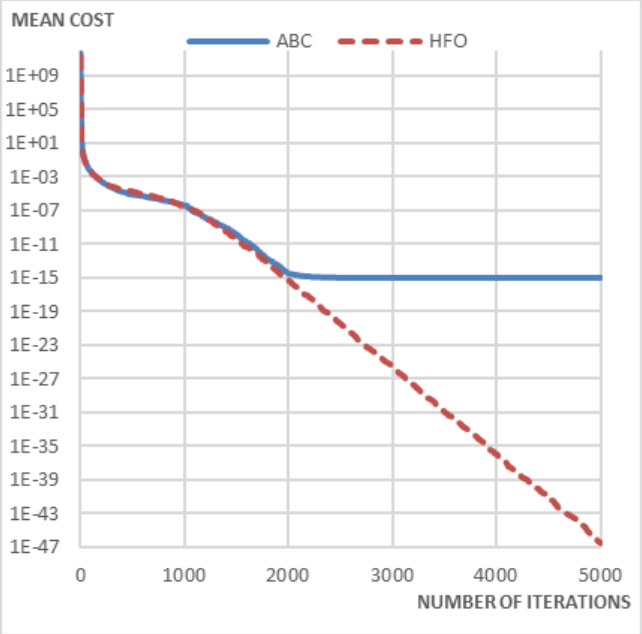

Figure 2. Evolution curves for Brown

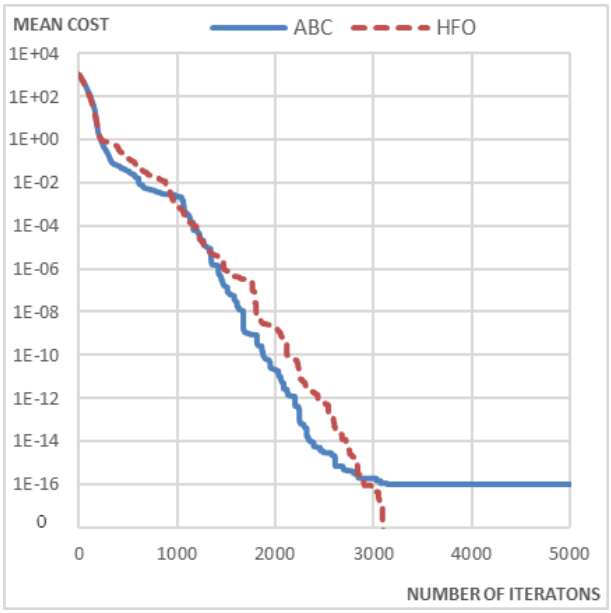

Figure 3. Evolution curves for Griewank

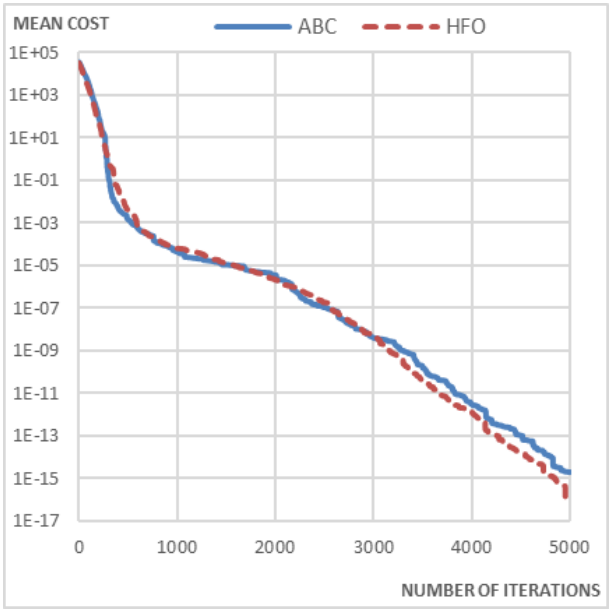

Figure 4. Evolution curves for Qing

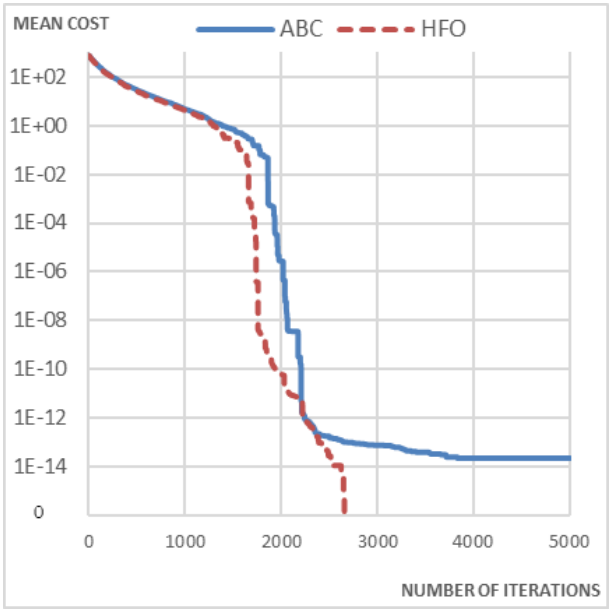

Figure 5. Evolution curves for Rastrigin

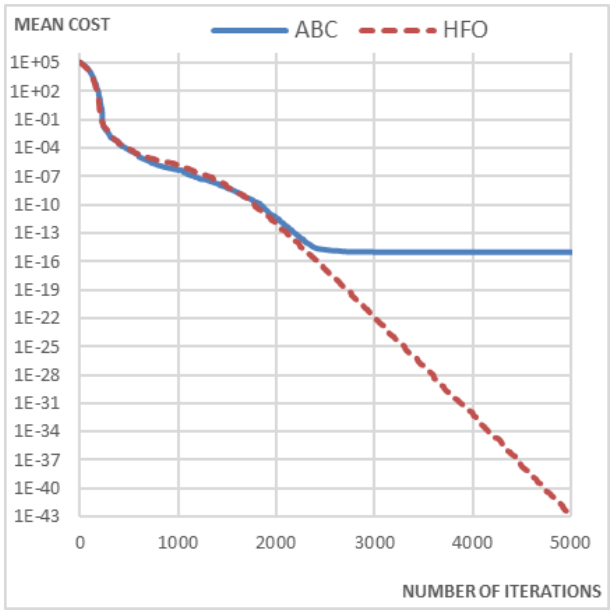

Figure 6. Evolution curves for Sphere

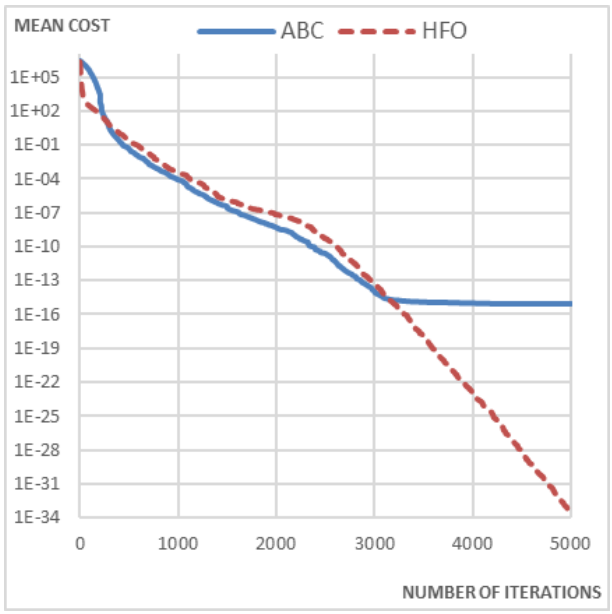

Figure 7. Evolution curves for Egg Crate

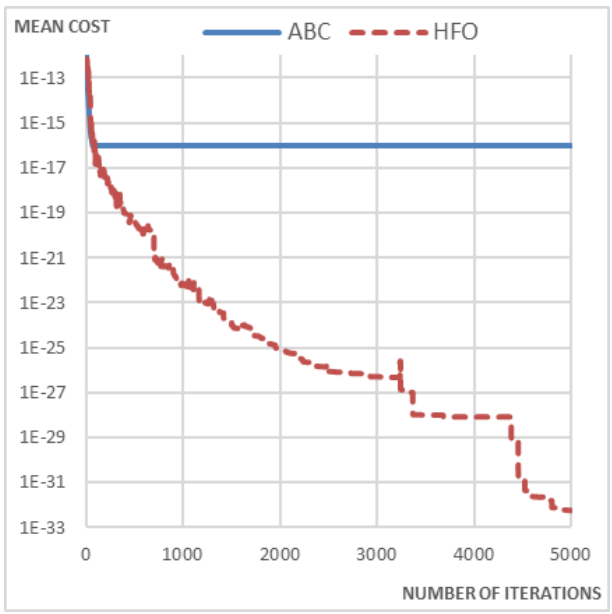

Figure 8. Evolution curves for Xin-SheYang

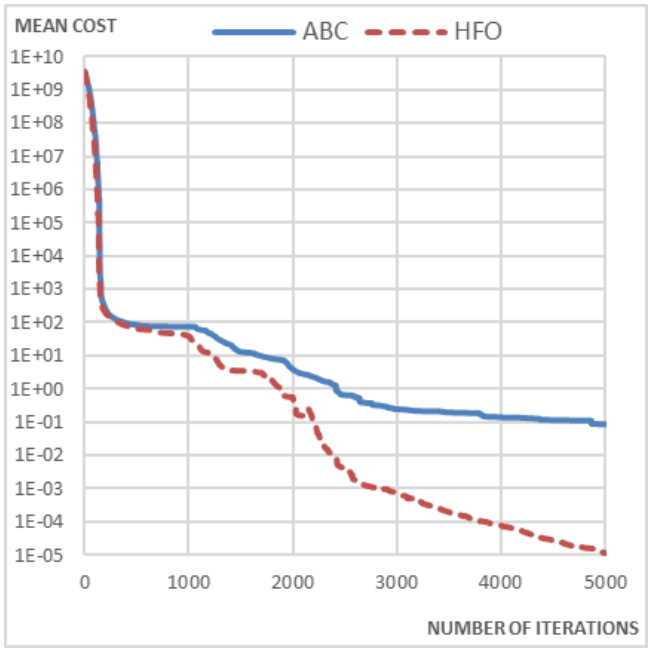

Figure 9. Evolution curves for Rosenbrock

REFERENCES

Abro A G & Mohamad-Saleh J (2014). Enhanced probability-selection artificial bee colony algorithm for economic load dispatch: A comprehensive analysis. Engineering Optimization, 46(10), 1315–1330. DOI: 10.1080/0305215X.2013.836639

Akay B & Karaboga D (2015). A survey on the applications of artificial bee colony in signal, image, and video processing. Signal, Image and Video Processing, 9, 967–990. DOI: 10.1007/s11760-015-0758-4

Aldwairi M, Khamayseh Y & Al-Masri M (2015). Application of artificial bee colony for intrusion detection systems. Security and Communication Networks, 8(16), 2730–2740. DOI: 10.1002/sec.588

Apalak M K, Karaboga D & Akay B (2014). The Artificial Bee Colony algorithm in layer optimization for the maximum fundamental frequency of symmetrical laminated composite plates. Engineering Optimization, 46(3), 420–437. DOI: 10.1080/0305215X.2013.776551

Chen J, Li C & Yu W (2017). Adaptive Image Enhancement Based on Artificial Bee Colony Algorithm. Advances in Engineering Research, 116, 689-695.

Chen S-M, Sarosh A & Dong Y-F (2012). Simulated annealing based artificial bee colony algorithm for global numerical optimization. Applied Mathematics and Computation, 219(8), 3575–3589. DOI: 10.1016/j.amc.2012.09.052

Cheng X & Jiang M (2012). An improved artificial bee colony algorithm based on Gaussian mutation and chaos disturbance. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 326–333. DOI: 10.1007/978-3-642-30976-2_39

Cuevas E, Zaldívar D, Pérez-Cisneros M, Sossa H & Osuna V (2013). Block matching algorithm for motion estimation based on Artificial Bee Colony (ABC). Applied Soft Computing, 13(6), pp. 3047–3059. DOI: 10.1016/j.asoc.2012.09.020

Gao W & Liu S (2012). A modified artificial bee colony algorithm. Computers & Operations Research, 39(3), 687–697. DOI: 10.1016/j.cor.2011.06.007

Han Y Y, Gong D & Sun X (2015). A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. Engineering Optimization, 47(7), 927–946. DOI: 10.1080/0305215X.2014.928817

He X, Wang W, Jiang J & Xu L (2015). An improved artificial bee colony algorithm and its application to multi-objective optimal power flow. Energies, 8(4), 2412–2437. DOI: 10.3390/en8042412

Huang F, Wang L & Yang C (2016). A new improved artificial bee colony algorithm for ship hull form optimization. Engineering Optimization, 48(4), 672–686. DOI: 10.1080/0305215X.2015.1031660

Ismail M M & Baskaran K (2014). Hybrid lifting based image compression scheme using particle swarm optimization algorithm and artifical bee colony algorithm. International Journal of Advanced Research in Computer and Communication Engineering, 3(1), 4899-4907.

Jia D, Duan X & Khan M K (2015). Modified artificial bee colony optimization with block perturbation strategy. Engineering Optimization, 47(5), 642–655. DOI: 10.1080/0305215X.2014.914189

Kang F, Li J & Ma Z (2011). Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions. Information Sciences, 181(16), 3508–3531. DOI: 10.1016/j.ins.2011.04.024

Kang F, Li J & Ma Z (2013). An artificial bee colony algorithm for locating the critical slip surface in slope stability analysis. Engineering Optimization, 45(2), 207–223. DOI: 10.1080/0305215X.2012.665451

Karaboga D (2005). An idea based on honey bee swarm for numerical optimization. Technical Report-tr06, Erciyes University, Engineering Faculty, Computer Engineering Department, 200, 1-10.

Karaboga D, Akay B & Ozturk C (2007). Artificial bee colony (ABC) optimization algorithm for training Feed-Forward neural networks. Modeling Decisions for Artificial Intelligence, Springer Berlin Heidelberg, pp. 318–329. DOI: 10.1007/978-3-540-73729-2_30

Karaboga D, Gorkemli B, Ozturk C & Karaboga N (2014). A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. Artificial Intelligence Review, 42, pp. 21–57. DOI: 10.1007/s10462-012-9328-0

Karaboga D & Ozturk C (2011). A novel clustering approach: Artificial Bee Colony (ABC) algorithm. Applied Soft Computing, 11(1), 652–657. DOI: 10.1016/j.asoc.2009.12.025

Keles M K & Kilic U (2018). Artificial Bee Colony Algorithm for feature selection on SCADI Dataset. 3rd International Conference on Computer Science and Engineering (UBMK), IEEE, 463–466. DOI: 10.1109/UBMK.2018.8566287

Liu Y, Ma L & Yang G (2017). A Survey of Artificial Bee Colony Algorithm. 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER), IEEE, 1510–1515. DOI: 10.1109/CYBER.2017.8446301

Lozano M, García-Martínez C, Rodríguez F J & Trujillo H M (2017). Optimizing network attacks by artificial bee colony. Information Sciences, 377, 30–50. DOI: 10.1016/j.ins.2016.10.014

Shah H, Herawan T, Naseem R & Ghazali R (2014). Hybrid guided artificial bee colony algorithm for numerical function optimization. Lecture Notes in Computer Science, 8794(7). DOI: 10.1007/978-3-319-11857-4_23

Sun L, Chen T & Zhang Q (2018). An artificial bee colony algorithm with random location updating. Scientific Programming.

Wang S, Guo X & Liu J (2019). An efficient hybrid artificial bee colony algorithm for disassembly line balancing problem with sequence-dependent part removal times. Engineering Optimization, 51(11), 1–18. DOI: 10.1080/0305215X.2018.1564918