Intelligent Production Machines and Systems
D.T. Pham, E.E. Eldukhri and A.J. Soroka (eds)

The Bees Algorithm – A Novel Tool for Complex Optimisation Problems

D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri , S. Rahim , M. Zaidi

Manufacturing Engineering Centre, Cardiff University, Cardiff CF24 3AA, UK

Abstract

A new population-based search algorithm called the *Bees Algorithm (BA)* is presented. The algorithm mimics the food foraging behaviour of swarms of honey bees. In its basic version, the algorithm performs a kind of neighbourhood search combined with random search and can be used for both combinatorial optimisation and functional optimisation. This paper focuses on the latter. Following a description of the algorithm, the paper gives the results obtained for a number of benchmark problems demonstrating the efficiency and robustness of the new algorithm.

Keywords: Bees Algorithm, Function Optimisation, Swarm Intelligence

1. Introduction

Many complex multi-variable optimisation problems cannot be solved exactly within polynomially bounded computation times. This generates much interest in search algorithms that find near-optimal solutions in reasonable running times. The swarm-based algorithm described in this paper is a search algorithm capable of locating good solutions efficiently. The algorithm is inspired by the food foraging behaviour of honey bees and could be regarded as belonging to the category of "intelligent" optimisation tools [1].

Section 2 reviews related work in the area of intelligent optimisation. Section 3 describes the foraging behaviour of natural bees and the core ideas of the proposed Bees Algorithm. Section 4 details the benchmarking problems used to test the efficiency and robustness of the algorithm and presents the results obtained. These show that the algorithm can reliably handle complex multi-model optimisation problems without being trapped at local solutions.

2. Intelligent swarm-based optimisation

Swarm-based optimisation algorithms (SOAs) mimic nature's methods to drive a search towards the optimal solution. A key difference between SOAs and direct search algorithms such as hill climbing and random walk is that SOAs use a population of solutions for every iteration instead of a single solution. As a population of solutions is processed in an iteration, the outcome of each iteration is also a population of solutions. If an optimisation problem has a single optimum, SOA population members can be expected to converge to that optimum solution. However, if an optimisation problem has multiple optimal solutions, an SOA can be used to capture them in its final population. SOAs include the Ant Colony Optimisation (ACO) algorithm [2], the Genetic Algorithm (GA) [3] and the Particle Swarm Optimisation (PSO) algorithm [4].

Common to all population-based search methods is a strategy that generates variations of the solution being sought. Some search methods use a greedy criterion to decide which generated solution to retain. Such a criterion would mean accepting a new solution if and only if it increases the value of the

objective function (assuming the given optimisation problem is one of optimisation). A very successful non-greedy population-based algorithm is the ACO algorithm which emulates the behaviour of real ants. Ants are capable of finding the shortest path from the food source to their nest using a chemical substance called pheromone to guide their search. The pheromone is deposited on the ground as the ants move and the probability that a passing stray ant will follow this trail depends on the quantity of pheromone laid. ACO was first used for functional optimisation by Bilchev [5] and further attempts were reported in [5, 6].

The Genetic Algorithm is based on natural selection and genetic recombination. The algorithm works by choosing solutions from the current population and then applying genetic operators – such as mutation and crossover – to create a new population. The algorithm efficiently exploits historical information to speculate on new search areas with improved performance [3]. When applied to optimisation problems, the GA has the advantage of performing global search. The GA may be hybridised with domain-dependent heuristics for improved results. For example, Mathur et al [6] describe a hybrid of the ACO algorithm and the GA for continuous function optimisation.

Particle Swarm Optimisation (PSO) is an optimisation procedure based on the social behaviour of groups of organisations, for example the flocking of birds or the schooling of fish [4]. Individual solutions in a population are viewed as "particles" that evolve or change their positions with time. Each particle modifies its position in search space according to its own experience and also that of a neighbouring particle by remembering the best position visited by itself and its neighbours, thus combining local and global search methods [4].

There are other SOAs with names suggestive of possibly bee-inspired operations [7-10]. However, as far as the authors are aware, those algorithms do not closely follow the behaviour of bees. In particular, they do not seem to implement the techniques that bees employ when foraging for food.

3. The bees algorithm

3.1. Bees in nature

A colony of honey bees can extend itself over long distances (more than 10 km) and in multiple directions simultaneously to exploit a large number of food sources [7,8]. A colony prospers by deploying its foragers to good fields. In principle, flower patches with plentiful amounts of nectar or pollen that can be collected with less effort should be visited by more bees, whereas patches with less nectar or pollen should receive fewer bees [9,10].

The foraging process begins in a colony by scout bees being sent to search for promising flower patches. Scout bees move randomly from one patch to another. During the harvesting season, a colony continues its exploration, keeping a percentage of the population as scout bees [8].

When they return to the hive, those scout bees that found a patch which is rated above a certain quality threshold (measured as a combination of some constituents, such as sugar content) deposit their nectar or pollen and go to the "dance floor" to perform a dance known as the "waggle dance" [7].

This mysterious dance is essential for colony communication, and contains three pieces of information regarding a flower patch: the direction in which it will be found, its distance from the hive and its quality rating (or fitness) [7,10]. This information helps the colony to send its bees to flower patches precisely, without using guides or maps. Each individual's knowledge of the outside environment is gleaned solely from the waggle dance. This dance enables the colony to evaluate the relative merit of different patches according to both the quality of the food they provide and the amount of energy needed to harvest it [10]. After waggle dancing on the dance floor, the dancer (i.e. the scout bee) goes back to the flower patch with follower bees that were waiting inside the hive. More follower bees are sent to more promising patches. This allows the colony to gather food quickly and efficiently.

While harvesting from a patch, the bees monitor its food level. This is necessary to decide upon the next waggle dance when they return to the hive [10]. If the patch is still good enough as a food source, then it will be advertised in the waggle dance and more bees will be recruited to that source.

3.2. *Proposed bees algorithm*

As mentioned , the Bees Algorithm is an optimisation algorithm inspired by the natural foraging behaviour of honey bees to find the optimal solution [4]. Figure 1 shows the pseudo code for the algorithm in its simplest form. The algorithm requires a number of parameters to be set, namely: number of scout bees (n), number of sites selected out of n visited sites (m), number of best sites out of

m selected sites (*e*), number of bees recruited for best e sites (*nep*), number of bees recruited for the other (*m-e*) selected sites (*nsp*), initial size of patches (*ngh*) which includes site and its neighbourhood and stopping criterion. The algorithm starts with the n scout bees being placed randomly in the search space. The fitnesses of the sites visited by the scout bees are evaluated in step 2.

1. Initialise population with random solutions.
2. Evaluate fitness of the population.
3. While (stopping criterion not met)
   //Forming new population.
4. Select sites for neighbourhood search.
5. Recruit bees for selected sites (more bees for best e sites) and evaluate fitnesses.
6. Select the fittest bee from each patch.
7. Assign remaining bees to search randomly and evaluate their fitnesses.
8. End While.

Fig.1 Pseudo code of the basic bees algorithm

In step 4, bees that have the highest fitnesses are chosen as "selected bees" and sites visited by them are chosen for neighbourhood search. Then, in steps 5 and 6, the algorithm conducts searches in the neighbourhood of the selected sites, assigning more bees to search near to the best e sites. The bees can be chosen directly according to the fitnesses associated with the sites they are visiting. Alternatively, the fitness values are used to determine the probability of the bees being selected. Searches in the neighbourhood of the best e sites which represent more promising solutions are made more detailed by recruiting more bees to follow them than the other selected bees. Together with scouting, this differential recruitment is a key operation of the Bees Algorithm.

However, in step 6, for each patch only the bee with the highest fitness will be selected to form the next bee population. In nature, there is no such a restriction. This restriction is introduced here to reduce the number of points to be explored. In step 7, the remaining bees in the population are assigned randomly around the search space scouting for new potential solutions. These steps are repeated until a stopping criterion is met. At the end of each iteration, the colony will have two parts to its new population – representatives from each selected patch and other scout bees assigned to conduct random searches.

4. Experiments

Clearly, the Bees Algorithm as described above is applicable to both combinatorial and functional optimisation problems. In this paper, functional optimisation will be demonstrated. The solution of combinatorial optimisation problems differs only in the way neighbourhoods are defined.

Two standard functional optimisation problems were used to test the Bees Algorithm and establish the correct values of its parameters and another eight for benchmarking the algorithm. As the Bees Algorithm searches for the maximum, functions to be minimised were inverted before the algorithm was applied.

Shekel's Foxholes (Fig. 2), a 2D function from De Jong's test suite, was chosen as the first function for testing the algorithm.

$$f(\vec{x}) = 119.998 - \sum_{i=1}^{25} \frac{1}{\mathrm{j} + \sum_{i=1}^{2} \left(x_i - a_{ij}\right)^6} \qquad (1)$$

$$a_{ij} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & \dots & 32 & 32 & 32 \end{pmatrix}$$

$$-65.536 \le x_i \le 65.536$$

For this function,

$$\vec{x}_{\max} = (-32, -32)$$

$$f(\vec{x}_{\max}) = 119.998$$

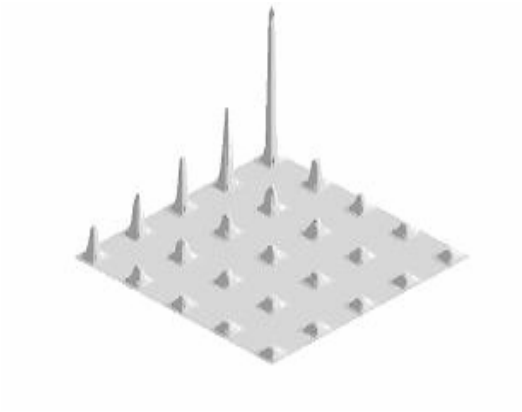

Fig 2. Inverted Shekel's Foxholes

The following parameter values were set for this test: population n= 45, number of selected sites m=3, number of elite sites e=1, initial patch size ngh=3, number bees around elite points nep=7, number of bees around other selected points nsp=2. Note that ngh defines the initial size of the neighbourhood in which follower bees are placed. For example, if x is the position of an elite bee in the i^{th} dimension, follower bees will be placed randomly in the interval $x_{ie} \pm ngh$ in that dimension at the beginning of the optimisation process. As the optimisation advances, the size of the search neighbourhood gradually decreases to facilitate fine tuning of the solution.

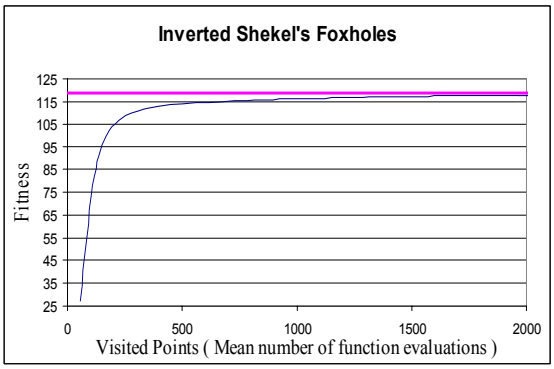

Fig 3. Evolution of fitness with the number of points visited (Inverted Shekel's Foxholes)

Fig. 3 shows the fitness values obtained as a function of the number of points visited. The results are averages for 100 independent runs. It can be seen that after approximately 1200 visits, the Bees Algorithm was able to find solutions close to the optimum.

To test the reliability of the algorithm, the inverted Schwefel's function with six dimensions (Eq. 2) was used. Fig. 4 shows a two-dimensional view of the function to highlight its multi modality.

The following parameter values were set for this test: population n=500, number of selected sites m=15, number of elite sites e=5, initial patch size ngh=20, number of bees around elite points nep=50, number of bees around other selected points nsp=30.

$$f(\vec{x}) = -\sum_{i=1}^{6} -x_i \sin\left(\sqrt{|x_i|}\right) \qquad (2)$$

$$-500 \le x_i \le 500$$

$$\vec{x}_{\max} = (420.9, 420.9, 420.9, 420.9, 420.9, 420.9)$$

$f(\vec{x}_{\max}) \approx 2513.9$ For this function,

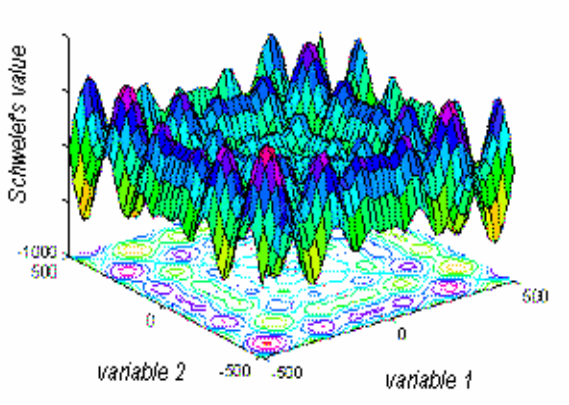

Fig 4. 2D Schwefel's function

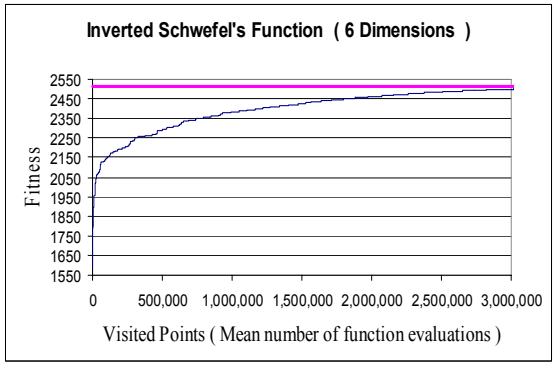

Fig 5. Evolution of fitness with the number of points visited (Inverted Schewefel's Fuction)

Fig. 5 shows how the fitness values evolve with the number of points visited. The results are averages for 100 independent runs. It can be seen that after approximately 3,000,000 visits, the Bees Algorithm was able to find solutions close to the optimum.

The Bees Algorithm was applied to eight benchmark functions [6] and the results compared with those obtained using other optimisation algorithms. The test functions and their optima are shown in Table 1. For additional results and a more detailed analysis of the Bees Algorithm, see [1].

Table 2 presents the results obtained by the Bees Algorithm and those by the deterministic Simplex method (SIMPSA) [6], the stochastic simulated annealing optimisation procedure (NE SIMPSA) [6], the Genetic Algorithm (GA) [6] and the Ant Colony System (ANTS) [6].

Again, the numbers of points visited shown are averages for 100 independent runs. Table 3 shows the empirically derived Bees Algorithm parameter values used with the different test functions.

Table 1. Test Functions

No	Function Name	Interval	Function	Global Optimum
1	De Jong	[-2.048, 2.048]	$\max F = (3905.93) - 100(x_1^2 - x_2)^2 - (1 - x_1)^2$	X(1,1) F=3905.93
2	Goldstein & Price	[-2, 2]	$\min F = [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $X[30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	X(0,-1) F=3
3	Branin	[-5, 10]	$\min F = a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1-f)\cos(x_1) + e$ $a = 1, b = \frac{5.1}{4}\left(\frac{7}{22}\right)^2, c = \frac{5}{22}X7, d = 6, e = 10, f = \frac{1}{8}X\frac{7}{22}$	X(-22/7,12.275) X(22/7,2.275) X(66/7,2.475) F=0.3977272
4	Martin & Gaddy	[0, 10]	$\min F = (x_1 - x_2)^2 + ((x_1 + x_2 - 10)/3)^2$	X(5,5) F=0
5	Rosenbrock	(a) [-1.2, 1.2] (b) [-10, 10]	$\min \quad F = 100\ (x_1^2 - x_2)^2 + (1 - x_1)^2$	X(1,1) F=0
6	Rosenbrock	[-1.2, 1.2]	$\min F = \sum_{i=1}^{3} \{100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2\}$	X(1,1,1,1) F=0
7	Hyper sphere	[-5.12, 5.12]	$\min F = \sum_{i=1}^{6} x_i^2$	X(0,0,0,0,0,0) F=0
8	Griewangk	[-512, 512]	$\max F = \frac{1}{0.1 + \left(\sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\right)}$	X(0,0,0,0,0,0,0,0,0,0) F=10

Table 2. Results

	SIMPSA		NE SIMPSA		GA		ANTS		Bees Algorithm	
func no	succ %	mean no. of evaluations	succ %	mean no. of evaluations	succ %	mean no. of evaluations	succ %	mean no. of evaluations	Succ %	mean no. of evaluations
1	****	****	****	****	100	10160	100	6000	100	**868**
2	****	****	****	****	100	5662	100	5330	100	**999**
3	****	****	****	****	100	7325	100	1936	100	**1657**
4	****	****	****	****	100	2844	100	1688	100	**526**
5a	100	10780	100	4508	100	10212	100	6842	100	**631**
5b	100	12500	100	5007	****	****	100	7505	100	**2306**
6	99	21177	94	3053	****	****	100	8471	100	**28529**
7	****	****	****	****	100	15468	100	22050	100	**7113**
8	****	****	****	****	100	200000	100	50000	100	**1847**

**** Data not available

The optimisation stopped when the difference between the maximum fitness obtained and the global optimum was less than 0.1% of the optimum value or less than .001, whichever was smaller. In case the optimum was 0, the solution was accepted if it differed from the optimum by less than 0.001.

The first test function was De Jong's, for which the Bees Algorithm could find the optimum 120 times faster than ANTS and 207 times faster than GA, with a success rate of 100%. The second function was Goldstein and Price's, for which the Bees Algorithm reached the optimum almost 5 times faster than ANTS and GA, again with 100% success. With Branin's function, there was a 15% improvement compared with ANTS and 77% improvement compared with GA, also with 100% success.

Table 3. Bees Algorithm parameters

fun no	n	m	e	n1	n2	ngh (initial)
1	10	3	1	2	4	0.1
2	20	3	1	1	13	0.1
3	30	5	1	2	3	0.5
4	20	3	1	1	10	0.5
5a	10	3	1	2	4	0.1
5b	6	3	1	1	4	0.5
6	20	6	1	5	8	0.1
7	8	3	1	1	2	0.3
8	10	3	2	4	7	5

Functions 5 and 6 were Rosenbrock's functions in two and four dimensions respectively. In the two-dimensional function, the Bees Algorithm delivers 100% success and good improvement over the other methods (at least twice fewer evaluations than the other methods). In the four-dimensional case, the Bees Algorithm needed more function evaluations to reach the optimum with 100% success. NE SIMPSA could find the optimum with 10 times fewer function evaluations but the success rate was only 94% and ANTS found the optimum with 100% success and 3.5 times faster than the Bees Algorithm. Test function 7 was a Hyper Sphere model of six dimensions. The Bees Algorithm needed half of the number of function evaluations compared with GA and one third of that required for ANTS. The eighth test function was a ten-dimensional function. The Bees Algorithm could reach the optimum 10 times faster than GA and 25 times faster than ANTS and its success rate was 100%.

5. Conclusion

This paper has presented a new optimisation algorithm. Experimental results on multi-modal functions in n-dimensions show that the proposed algorithm has remarkable robustness, producing a 100% success rate in all cases. The algorithm converged to the maximum or minimum without becoming trapped at local optima. The algorithm generally outperformed other techniques that were compared with it in terms of speed of optimisation and accuracy of the results obtained. One of the drawbacks of the algorithm is the number of tunable parameters used. However, it is possible to set the parameter values by conducting a small number of trials.

Other optimisation algorithms usually employ gradient information. However, the proposed algorithm makes little use of this type of information and thus can readily escape from local optima. Further work should address the reduction of parameters and the incorporation of better learning mechanisms.

Acknowledgements

The research described in this paper was performed as part of the Objective 1 SUPERMAN project, the EPSRC Innovative Manufacturing Research Centre Project and the EC FP6 Innovative Production Machines and Systems (I*PROMS) Network of Excellence.

References

[1] Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S and Zaidi M. The Bees Algorithm. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005.

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

[3] Goldberg DE. Genetic Algorithms in Search, Optimization and Machine Learning. Reading: Addison-Wesley Longman, 1989.

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

[5] Bilchev G and Parmee IC. The Ant Colony Metaphor for Searching Continuous Design Spaces. in Selected Papers from AISB Workshop on Evolutionary Computing. (1995) 25-39.

[6] Mathur M, Karale SB, Priye S, Jayaraman VK and Kulkarni BD. Ant Colony Approach to Continuous Function Optimization. Ind. Eng. Chem. Res. 39(10) (2000) 3814-3822.

[7] Von Frisch K. Bees: Their Vision, Chemical Senses and Language. (Revised edn) Cornell University Press, N.Y., Ithaca, 1976.

[8] Seeley TD. The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies. Massachusetts: Harvard University Press, Cambridge, 1996.

[9] Bonabeau E, Dorigo M, and Theraulaz G. Swarm Intelligence: from Natural to Artificial Systems. Oxford University Press, New York, 1999.

[10] Camazine S, Deneubourg J, Franks NR, Sneyd J, Theraula G and Bonabeau E. Self-Organization in Biological Systems. Princeton: Princeton University Press, 2003.