SWARM OPTIMIZATION ALGORITHM BASED ON THE ANT COLONY LIFE CYCLE

Jiraporn Kiatwuthiamorn[1], Arit Thammano[2]

[1,2]Faculty of Information Technology,
King Mongkut's Institute of Technology Ladkrabang,
Ladkrabang, Bangkok 10520
Thailand

E-mail: jiraporn.kia@rmutr.ac.th[1], arit@it.kmitl.ac.th[2]

DOI: https://doi.org/10.22452/mjcs.sp2019no2.1

ABSTRACT

Optimization is very important to the success of any business. One technique for solving optimization is swarm intelligence; it has been successfully applied to solve a wide range of optimization problems. We devised a new swarm intelligence optimization algorithm based on the cooperative behavior of three different kinds of ants in a colony. Our algorithm consists of both exploration and exploitation processes to achieve better search performance. A new local search, inspired by the foraging of desert ants, was introduced to help the search move away from the local optima. Performance was evaluated on 23 standard benchmark functions of varying complexity. Our algorithm was able to find the global optima in more than 80 percent of the test functions, whereas the second-place algorithm only found around 10 percent of the functions tested.

Keywords: Biologically inspired algorithm, Ant colony life cycle, Swarm intelligence, Optimization algorithm.

1.0 INTRODUCTION

Many researchers have described new algorithms for solving optimization problems. The goal of an optimization is to find the best solution to a given problem under some constraints [1]. At present, many optimization algorithms are based on natural behaviours. Some of these, the swarm intelligence algorithms, mimic the behavior of living creatures. Swarm intelligence algorithms are inspired by animal behavior, such as food-seeking of flocking birds, swarming bees, walking ants and swimming bacteria. Two widely known algorithms of this type are Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). PSO [2] described by Kennedy and Eberhart in 1995, was based on the food-seeking behaviors of swarming creatures, such as birds and fish, while Dorigo's ACO [3] was based on the food-seeking behavior of ants. These algorithms have been successfully applied to various kinds of optimization problems such as the traveling salesman, scheduling, vehicle routing and classification problems. Inspired by these early successes, more algorithms based on some creature behaviours were developed. For example, Simon [4] described a Biogeography-Based Optimization (BBO), based on the migration behavior and the geography of their habitat. Gong et al. [5] developed BBO further, by applying three mutation operators—Gaussian, Cauchy, and Levy—to diversify search solutions, and used their algorithm, called real-coded biogeography-based optimization with mutation, to solve a continuous optimization problem more effectively. In 2012, Gandomi and Alavi [6] defined a Krill Herd (KH) algorithm based on the behavior of krill, that gather into a large herd to deter an attack by a predator. Any movement of the herd must preserve its density and any movement of an individual krill depends on the presence of other krill, their foraging activity and random motion. In 2012, Niu and Wang [7] presented an adapted Bacterial Colony Optimization, originally based only on individual E. coli behavior throughout its life cycle, to include a combination of a chemotaxis strategy and bacterial communication, which is used between individuals and between groups to enhance search effectiveness. A widely used Fruit Fly Optimization Algorithm (FFA), based on fruit fly food-seeking behavior by visual and olfactory senses, was proposed by Pan [8] in 2012. In 2014, Pan et al. [9] improved it to better solve high-dimensional continuous function optimization problems, by introducing an automatically adjusted food-seeking distance, which was originally fixed. In 2012, Rao et al. [10] described a Teaching-Learning-Based Optimization (TLBO), based on influences of teacher and peers on a student. It had two operation phases: teacher phase and learner phase.

In this paragraph, some optimization algorithms that are not based on living creatures are explored. In 2012, Lam et al. [11] presented a real-coded chemical reaction optimization algorithm inspired by the energy minimization principle. In 2015, a Water Cycle Algorithm (WCA), including an evaporation rate, was developed by Sadollah et al. for solving both constrained and unconstrained optimization problems [12]. Eskandar et al. [13] adapted WCA, which is based on the nature of the water cycle and its downward flow from the source to the sea.

Sadollah et al. added a feature, that took into account different evaporation rates in different water sources to make WCA more effective. Ghaemi and Feizi-Derakhshi [14] introduced a Forest Optimization Algorithm (FOA), that was inspired by the longevity of trees in a forest. Its search is based on the germination of seeds, that fell under a tree and were carried afar by animals or other natural processes, so that a global search can cover more space. The search was also based on the tree age and the survival likelihood of neighboring trees.

In this paper, we describe a new swarm intelligence algorithm, that was inspired by survival behaviors of ants in their life cycle. Imitating these behaviors, the algorithm was developed for solving continuous optimization problems and its performance was shown to be favorable.

The remainder of this paper consists of the following sections: section 2 describes the life cycle of an ant colony and several ant behaviors; section 3 describes our algorithm; section 4 describes the results; and the last section is the conclusion.

2.0 BIOLOGICAL BACKGROUND

This section presents the biological background for our algorithm, which includes the life cycle of an ant colony and several ant behaviors.

2.1 Life Cycle of an Ant Colony

Ants are social animals, that live and work together, for the survival of the colony as a whole. An ant colony consists of a queen, workers and drones (male ants). Each kind of ant has different functions. The queen and drones are reproductive; they mostly stay in the nest. When the time and the conditions are suitable for mating, they will fly out to mate. Workers are non-reproductive. Workers perform several functions, such as foraging, caring for the queen and larvae, cleaning up the nest and defending the nest from attackers. The life cycle of an ant colony can be divided into three stages – see Fig 1 [15].

2.1.1 Founding Stage

The founding stage starts when the time and environmental conditions are suitable for mating. Reproductive ants from many colonies fly out to mate in a suitable place, such as a high mound or a large tree. The ants mate while they are in the air. Each queen mates with one or more drones. After mating, the drones die and the queen searches for a suitable place to start a new colony and to lay eggs.

2.1.2 Ergonomic Stage

This stage occurs while the colony is young and underpopulated. At the beginning of this stage, the queen lays eggs that mostly mature to be workers. The workers perform various functions for the survival of the colony. The number of ants in the colony grows rapidly, in this stage, and stabilizes to a fairly constant level at the end of the stage.

2.1.3 Reproductive Stage

When the workers have gathered sufficient food and the time and environmental conditions are suitable (which are different for different ant species), the queen will lay eggs that will grow up to be reproductive ants and the workers will take care of the larvae. The mature reproductive ants wait for the right conditions to fly out of the nest to mate and start their own nest.

As described above, each group of ants has a distinct duty, which they perform for the survival of the colony. The position of their nest can be thought of as the location of a rich food source. Their queen is like the heart of the colony. In our algorithm, the position, that the queen eventually takes, represents the best solution for the problem, and the activity, that each ant from each group in the colony undertakes during a period of time in their lives, causes a step-change or move toward this best solution.

2.2 Mating Behavior

When the weather, temperature, humidity and the time of the year are right for reproduction, as in the founding stage, reproductive ants from many colonies fly out to a suitable place to find their mates. Females mate with one or more drones for a short period of time while flying. After mating, the drones die and the females find a suitable

place to start a new colony (which can be different for different ant species) and shed their wings. Then, the new queen starts laying eggs.

Ant mating behavior can be thought of as a means to discover a new and better solution because, after a female ant has mated, it will find a place to build its new nest, that is rich with food and far away from predators or other competing colonies.

2.3 Foraging Behavior

Foraging is one of the most important worker tasks, since it directly affects the survival of the colony. Different species of ants can have different foraging behaviors, mainly depending on the location of their habitat. Some species of ants follow a pheromone trace to move along a path toward a food source, while some desert species follow their leader who navigates a path, based on landmarks [16-19]. In our algorithm, foraging behavior, based on landmarks, is the local search for the best solution.

2.4 Migration Behavior

A colony of ants may migrate to build a new nest in a new place, when the colony is threatened by predators or when the ants have found a richer food source. Workers carry the queen and larvae to the new place. In our algorithm, ant migration to move their queen and build a nest in a more suitable place represents diverse searches, converging to the best solution.

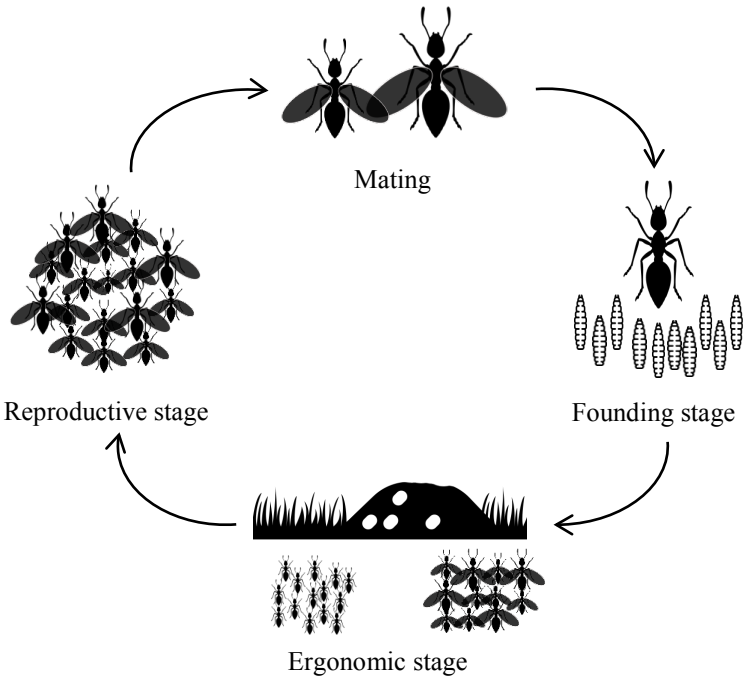

Fig. 1: Life cycle of an ant colony

3.0 OPTIMIZATION ALGORITHM INSPIRED BY THE LIFE CYCLE OF AN ANT COLONY

From fossil records, dated over millions of years old, ants have been shown to survive and prosper through natural selection. Hence, the way they live provides us with an idea for finding the best solution in a search space. We designed our algorithm steps to mimic the behaviors of ants of different groups throughout all phases of their life cycle. The flowchart of our algorithm is shown in Fig 2. Each step in the flowchart is explained below.

3.1 Initialization

First, initial values of the parameters are set by the user. Those parameters include the number of nests (N), the total number of ants in each nest (NP), the number of male ants (NM), the number of offspring created in each iteration

(NO), the lifespan of a worker ant and the maximum number of iterations. The algorithm starts by randomly picking G positions in the search space, where G >> N. Then the fitness of each position is evaluated. The one with the highest fitness is chosen to be the position of the first nest. Next, the Euclidean distances between the first nest and the remaining G-1 positions are calculated; the one located farthest from the first nest is assigned to be the second nest. This process continues, assigning locations for the j^{th} nest, where j = {3, 4, …, N}. It should be reiterated that the j^{th} nest must be located farthest away from the first j-1 nests.

When all N nests are determined, NP initial ants of each nest are randomly placed at a location inside the nest, where the boundary of the nest is set to be half the distance to the nearest nest. The initial queen in each nest is the one with the best fitness among all ants in the nest; hereafter, her location represents the location of the nest itself.

Next, NM initial male ants are selected by roulette wheel selection method in which the size of the slot corresponds to the distance between the ant and the queen, as in:

$$slot_i = \frac{d_i}{\sum_{k=1}^{NP-1} d_k} \quad (1)$$

where d_i is the Euclidean distance between the queen and ant i. The rest of the initial ants which have not been selected are assigned as workers.

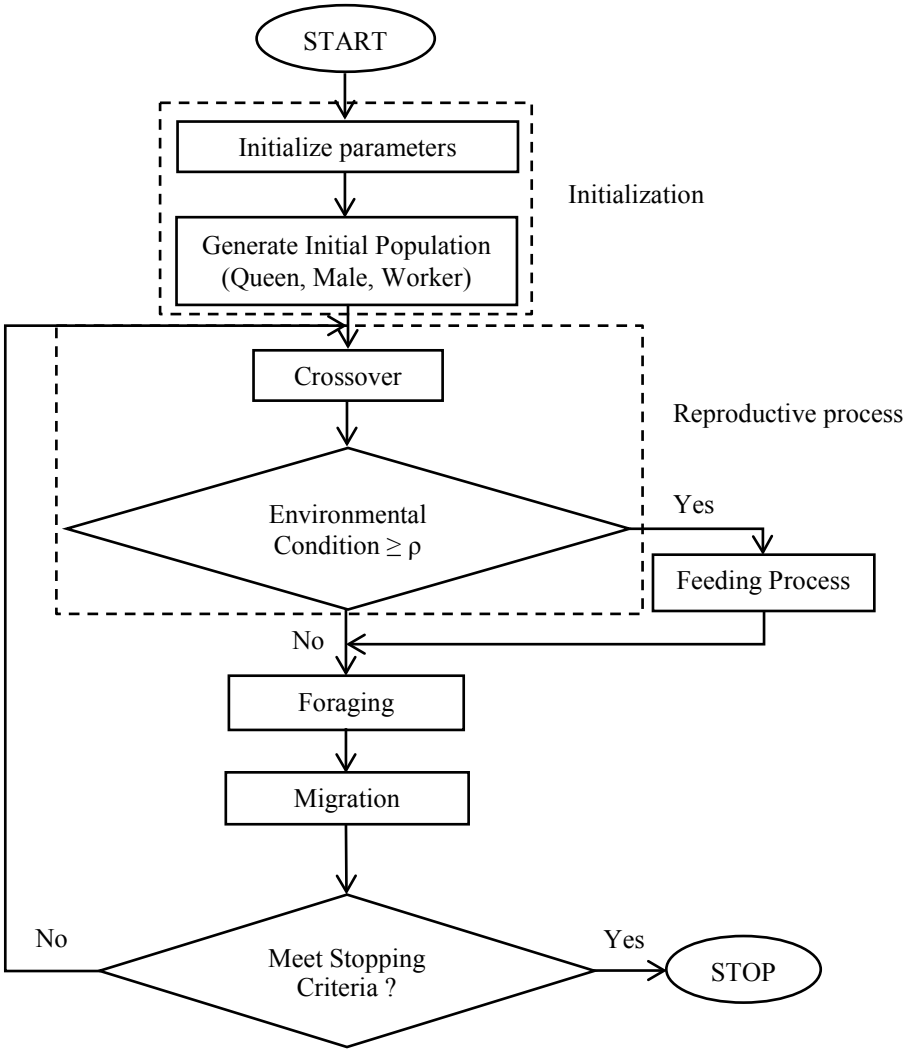

Fig. 2: Flowchart of the proposed algorithm

3.2 Reproductive Process

This step begins with multiple matings between the queen and several males. In this algorithm, each mating creates only one offspring. Therefore, a group of NO males are randomly selected from the set of NM males. Then an offspring is generated by arithmetically crossing [20] the genotype of the queen with that of the selected male:

$$off = \alpha q + (1 - \alpha)m \tag{2}$$

where q is the genotype of the queen, m is the genotype of each selected male and α is a random number in [0, 1]. An example of the arithmetic crossover operation is shown in Fig 3.

The set of offspring generated either develops into non-reproductive workers, or reproductive males and females, depending on the environmental conditions and the needs of the colony. When a queen has just started a new nest, workers are most needed to gather food and maintain the nest. However, when the environment is suitable, females and males are needed to expand the population. Our algorithm mimics this behavior to determine the role of newly generated offspring. First, the environmental condition of the colony is determined by calculating the ratio of the average fitness of workers to the fitness of the queen. It is very important to note that, in this paper, the fitness function (f(•)) is defined as the reciprocal of the objective function. If the ratio is less than an input threshold, ρ, the full set of offspring will develop into workers; otherwise, they will develop into reproductive ants. Offspring in the reproductive group will be improved by workers, as described in the feeding process in Step 3.3, while non-reproductive offspring will be assigned to forage for food as described in the foraging step 3.4.

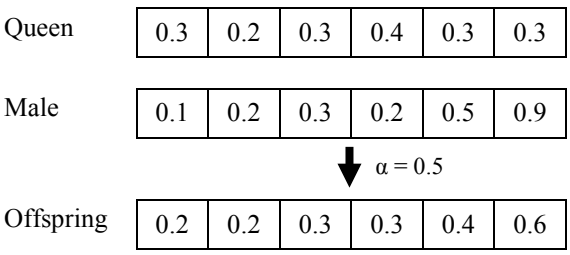

Fig. 3: Example of the arithmetic crossover operation

3.3 Feeding Process

This process starts by pairing each of the reproductive offspring generated in step 3.2 with a worker ant. For each reproductive offspring, a worker is randomly chosen from a roulette wheel, in which the size of the slot, $slot_i$, for a particular worker, is proportional to the its fitness:

$$slot_i = \frac{f(w_i)}{\sum_{k=1}^{NW} f(w_k)} \tag{3}$$

where $f(w_i)$ is the fitness of worker *i* and NW is the number of workers in the current iteration. As the result of the roulette wheel selection method, the fitter workers are likely to take care of more offspring than the less fit ones. After pairing, the worker that has a better fitness value than the offspring modifies the genotype of the offspring according to Eq. (4).

$$off = off + \beta(w - off) \tag{4}$$

where β is a random real number in [0, 1]. However, if the offspring has a better fitness than the worker, the offspring will not be modified. Next, each reproductive offspring, which passes through the feeding process, is checked to determine whether it resides within any of the existing nests. If yes, the offspring fitness is compared with that of the queen; if it is better than the queen, it replaces the queen, whereas the offspring has lower fitness value, it is killed. If the offspring does not reside within any of the existing nests, this means this area has not been covered and searched by any of the existing nests. In order to start exploring this area, a new nest is built and the offspring is assigned to be the queen of a new nest.

3.4 Foraging

The foraging is a task performed by worker ants. In nature, most ants forage by following the pheromone trails of earlier successful foragers. However, desert ants follow the leader visually, while it is moving toward a landmark. In our algorithm, foraging mimics the desert ant behavior, by having the leader move toward a randomly selected landmark and the members follow the leader, where the position, w_i, for ant i is calculated:

$$w_i = \begin{cases} w_i + \left(\frac{f(w_i)}{f(w_{best})}\right)(landmark - w_i) + \left(1 - \frac{f(w_i)}{f(w_{best})}\right)(w_{best} - w_i), \ i\ is\ the\ leader. \\ w_i + \left(\frac{f(w_i)}{f(w_{best})}\right)\left(\frac{1}{i-1}\sum_{k=1}^{i-1}(w_k - w_i)\right) + \left(1 - \frac{f(w_i)}{f(w_{best})}\right)(w_{best} - w_i), \ i\ is\ the\ member. \end{cases} \quad (5)$$

where $f(w_i)$ is the fitness of worker i, $f(w_{best})$ is the fitness of the fittest worker among all foraging groups, and k is the worker that is in the same foraging group as the worker i and is in front of the worker i in the line. In this foraging, each nest sends out R foraging groups to gather food in the vicinity of the nest. Therefore, the workers in each nest are divided into R foraging groups using the following steps:

i) The workers are ranked by fitness.
ii) The workers are assigned in a circular order to the groups, starting from the fittest worker in the ranking – i.e., rank 1 is assigned to the first group, rank 2 is assigned to the second group, …, rank R+1 is assigned to the first group, rank R+2 is assigned to the second group, and so on, until all workers have been assigned. The first assigned worker of each group assigned as leader. All workers, as well as the leader, have a lifetime of L iterations. After that, they die and the successors in line replace them.

It is important to note that, if a foraging group moves toward a landmark for a length of time and does not find a good food source along the path, a new landmark will be randomly selected.

3.5 Migration of the Nest

In our algorithm, the location of the queen also represents the location of the nest. If a worker wanders into a position that makes its fitness value greater than that of the queen, the algorithm will move the queen to that position, which mimics migrating the nest to a better position.

3.6 Termination

The algorithm runs step 3.2 to step 3.5 repeatedly until one of the stopping conditions is satisfied. Then it stops and outputs the location of the best nest. The pseudocode is presented in Fig. 4.

4.0 EXPERIMENTAL RESULTS

A total of 23 standard benchmark functions [12][21], of various complexity, were used to assess the algorithm in terms of convergence rate and solution quality. The 23 benchmark functions can be classified into two groups according to their dimensional size.

- First group: 13 low-dimensional functions (11 two-dimensional functions and 2 four-dimensional functions).
- Second group: 10 high-dimensional (30 dimensions) functions that represent computationally complex tasks. The functions F1 – F6 are unimodal, whereas F7 – F10 are multimodal functions. The multimodal functions were used to test the ability of the algorithm to escape local minima.

The test functions are described in Tables 1 and 2, which provide the following information for each test function: Function name, Equation, Dimension (D), Search domain (S), and Optimum value (f_{min}). For each test function, five experimental repetitions were performed with different initial populations. Since the performance of the metaheuristic algorithms strongly depends on parameter settings, as most other intelligent systems, the parameters of our algorithm were tuned in order to optimize the output. After tuning, the best value for each parameter was chosen – see Table 3.

```
Initialize the control parameters;
Select the positions of N nests from the randomly generated G positions;
Randomly generate initial population of each nest;
Repeat
    For each nest
        Create the offspring by using the arithmetic crossover;
        Calculate the ratio of the average fitness of workers to the fitness of the queen, and then assign it as
        the environmental condition of the nest;
        If the environmental condition >= ρ then
            Pair each offspring with a worker ant by using the roulette wheel selection;
            For each newly created offspring
                If f(w) > f(off) then
                    off ← off + β(w – off);
                End if
                If the offspring resides within any of the existing nests then
                    If f(off) > f(q) then
                        q ← off;
                    Else
                        Kill the offspring;
                    End if
                Else
                    Create a new nest, and then assign the offspring to be the queen of this new nest;
                End if
            End for
        Else
            Append the newly created offspring to a list of worker ants;
        End if
        Divide the worker ants into R groups;
        For each worker ant
            Move the worker ant to its new position;
        End for
        If f(w_best) > f(q) then
            q ← w_best;
        End if
    End for
Until one of the stopping criteria is met.
Return the location of the best nest;
```

Fig. 4: Pseudocode of our algorithm

Table 1: Descriptions of the low-dimensional functions

	Function Name	Equation	D	S	f_{min}
F1	Six-hump camel-back	$4x_1^2 - 2.1x_1^4 + \frac{x_1^6}{3} + x_1x_2 - 4x_2^2 + 4x_2^4$	2	[-5, 5]	-1.0316285
F2	Branin	$\left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	2	$x_1 \in [-5, 10]$ $x_2 \in [0, 15]$	0.39788736
F3	Goldstein-Price	$\left(1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right)$ $\times\left(30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right)$	2	[-2,2]	3
F4	Beale	$[1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2$	2	[-4.5,4.5]	0
F5	Easom's Function	$-\cos(x_1)\cos(x_2)e^{[-(x_1-\pi)^2-(x_2-\pi)^2]}$	2	[-10,10]	-1
F6	Dekkers and Aarts	$10^5 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4$	2	[-20,20]	-24777
F7	Shubert	$\left(\sum_{i=1}^{5} i\cos((i+1)x_1 + i)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2 + i)\right)$	2	[-10,10]	-186.730908
F8	De Jong	$3905.93 - 100(x_1^2 - x_2)^2 - (1 - x_1)^2$	2	[-2.048, 2.048]	3905.93
F9	Martin and Gaddy	$(x_1 - x_2)^2 + [(x_1 + x_2 - 10)/3]^2$	2	[0, 10]	0
F10	Schaffer	$0.5 + \frac{\sin^2\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{\left(1 + 0.001(x_1^2 + x_2^2)\right)^2}$	2	[-100, 100]	0
F11	Easton and Fenton	$\left\{12 + x_1^2 + \frac{1 + x_2^2}{x_1^2} + \frac{x_1^2x_2^2 + 100}{(x_1x_2)^4}\right\}\left(\frac{1}{10}\right)$	2	[0, 10]	1.74
F12	Wood	$100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2$ $+10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1)$	4	[-5, 5]	0
F13	Powell Quartic	$(x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$	4	[-5, 5]	0

Table 2: Descriptions of the high-dimensional functions

	Function Name	Equation	D	S	f_{min}
F1	Rosenbrock	$\sum_{i=1}^{D-1}\left[100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right]$	30	[-30, 30]	0
F2	Schwefel 2.21	$\max\{\lvert x_i\rvert, 1 \le i \le D\}$	30	[-100, 100]	0
F3	Step	$\sum_{i=1}^{D}(\lfloor x_i + 0.5\rfloor)^2$	30	[-100, 100]	0
F4	Sum of different power	$\sum_{i=1}^{D}\lvert x_i\rvert^{i+1}$	30	[-1, 1]	0
F5	Sphere	$\sum_{i=1}^{D} x_i^2$	30	[-100, 100]	0
F6	Schwefel 2.22	$\sum_{i=1}^{D}\lvert x_i\rvert + \prod_{i=1}^{D}\lvert x_i\rvert$	30	[-100, 100]	0
F7	Salomon	$1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{n} x_i^2}$	30	[-100, 100]	0
F8	Rastrigin	$\sum_{i=1}^{D}(x_i^2 - 10\cos(2\pi x_i) + 10)$	30	[-5.12, 5.12]	0
F9	Griewank	$\sum_{i=1}^{D}\frac{x_i^2}{4000} - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	[-600, 600]	0
F10	Ackley	$20 - 20\exp\left(-\frac{1}{5}\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + e$	30	[-32.768, 32.768]	0

Table 3: Values of parameters used in the proposed algorithm

Parameters	Values
Maximum number of iterations	10,000
Number of nests (N)	5
Number of ants in each nest (NP)	41
Number of male ants (NM)	20
Number of workers (NW)	20
Number of offspring created in each iteration (NO)	10
ρ	0.1 – 1.0
Number of foraging routes (R)	5
Lifetime of the worker (L)	10 iterations

The experimental results are reported in this and the following paragraphs. For convergence rate, the performance of our algorithm was compared with GA. For the low-dimensional functions, our algorithm converged to the optimum solution much faster than GA. As an example, figures 5 – 7 compare convergence of our algorithm and GA for the Six-hump camel-back, Branin, and Goldstein-price functions. From those figures, it is clear that our algorithm converged very much faster than GA, finding the global optima in a very small number of iterations, while GA was not able to find the global optimum. For all high-dimensional functions, except Rosenbrock, our algorithm converged slightly faster than the GA. As an example of this, the comparison of the convergence curves of our algorithm and GA for Salomon function is shown in figure 8. For Rosenbrock function, our algorithm converged slower than the GA, but eventually converged to a better solution. In figure 9, we see that for Rosenbrock, GA rapidly converged to its final value of 28.75187 while our algorithm converged steadily and slowly to its final value of 26.46650.

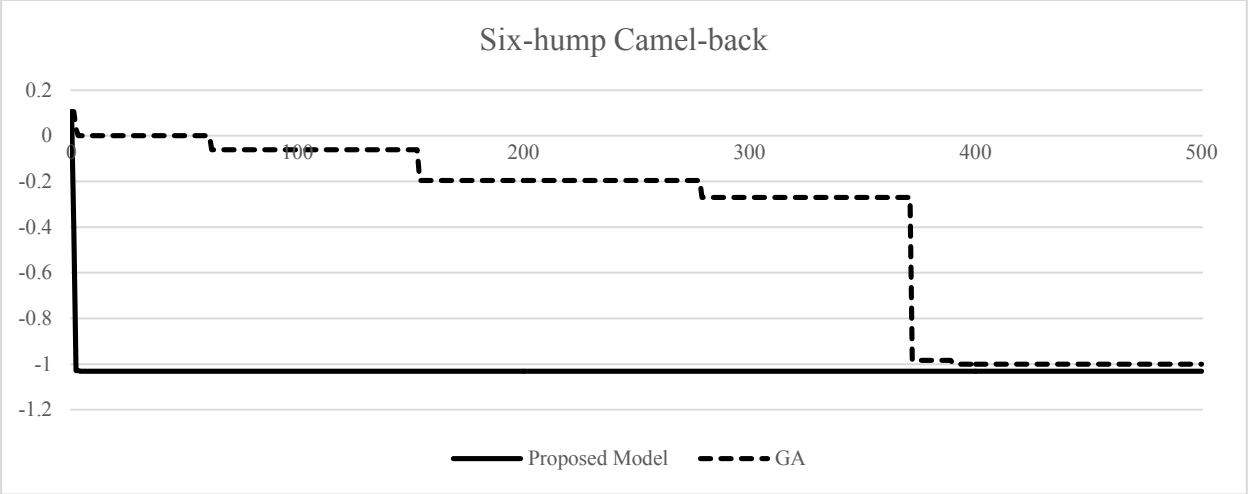

Fig. 5: Convergence curves for Six-hump camel-back function

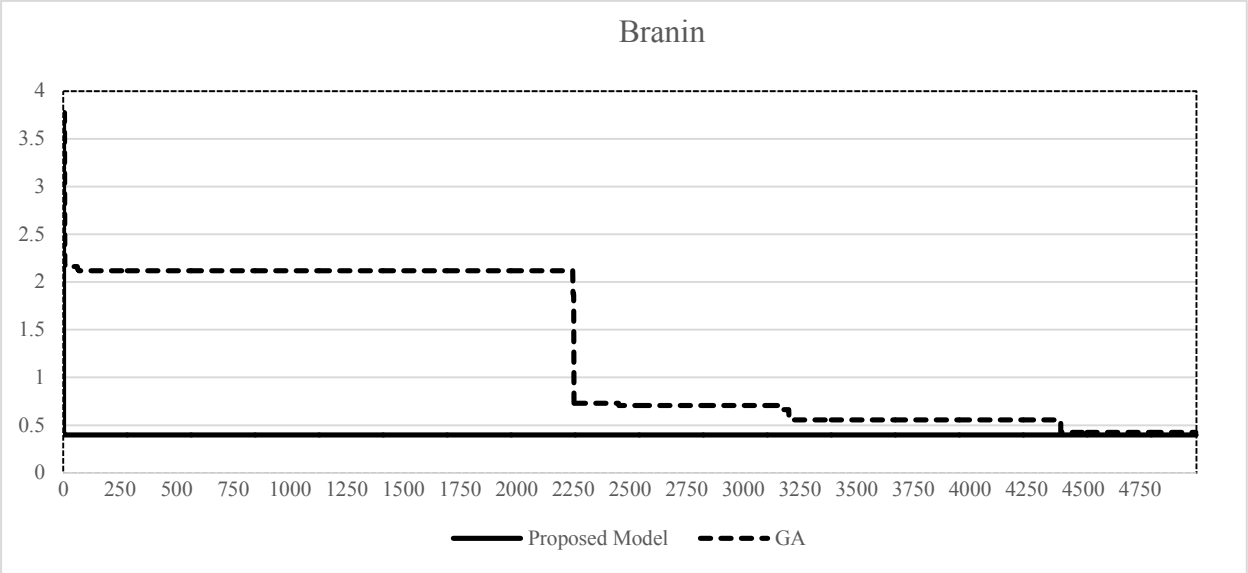

Fig. 6: Convergence curves for Branin function

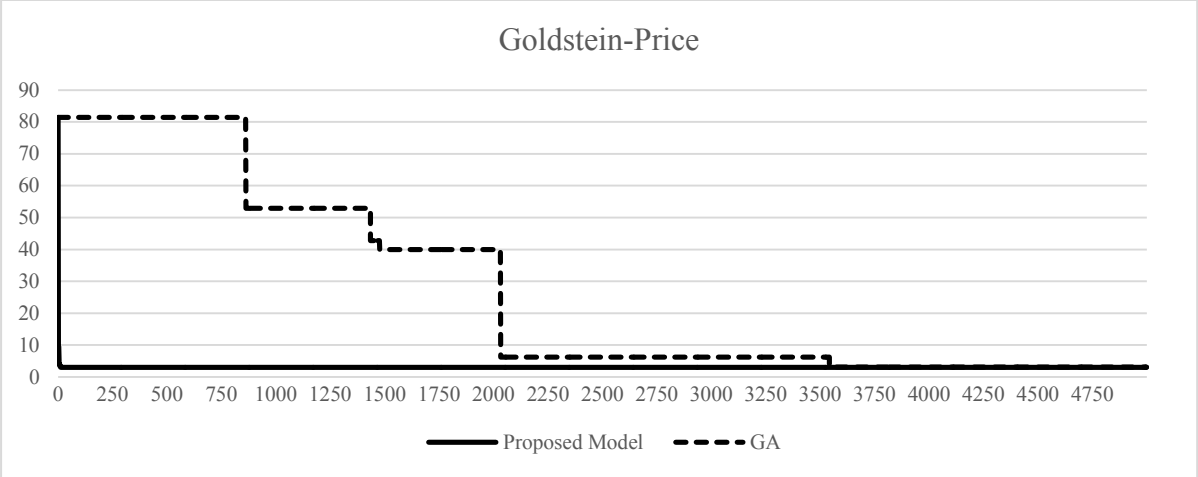

Fig. 7: Convergence curves for Goldstein-price function

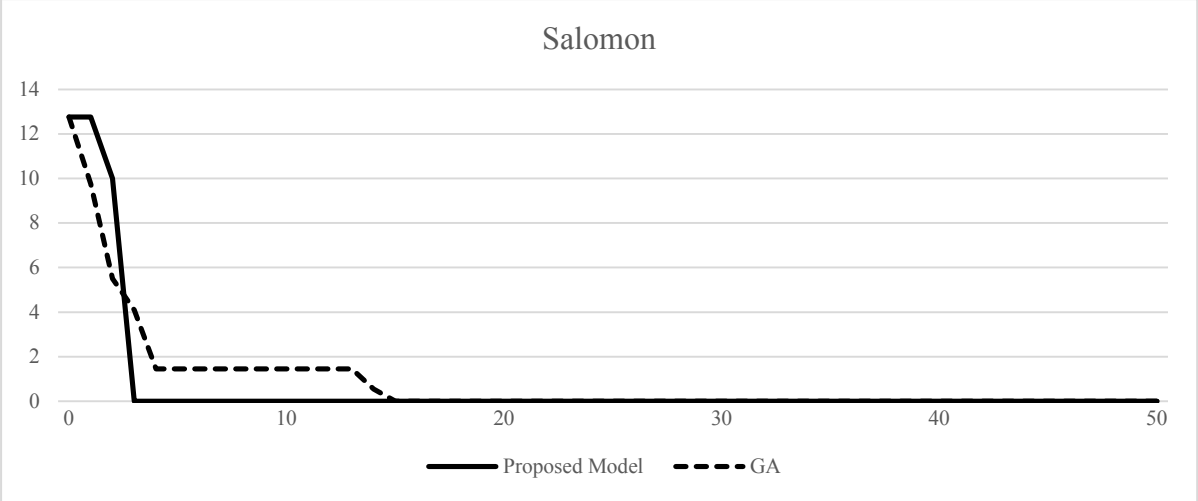

Fig. 8: Convergence curves for Salomon function

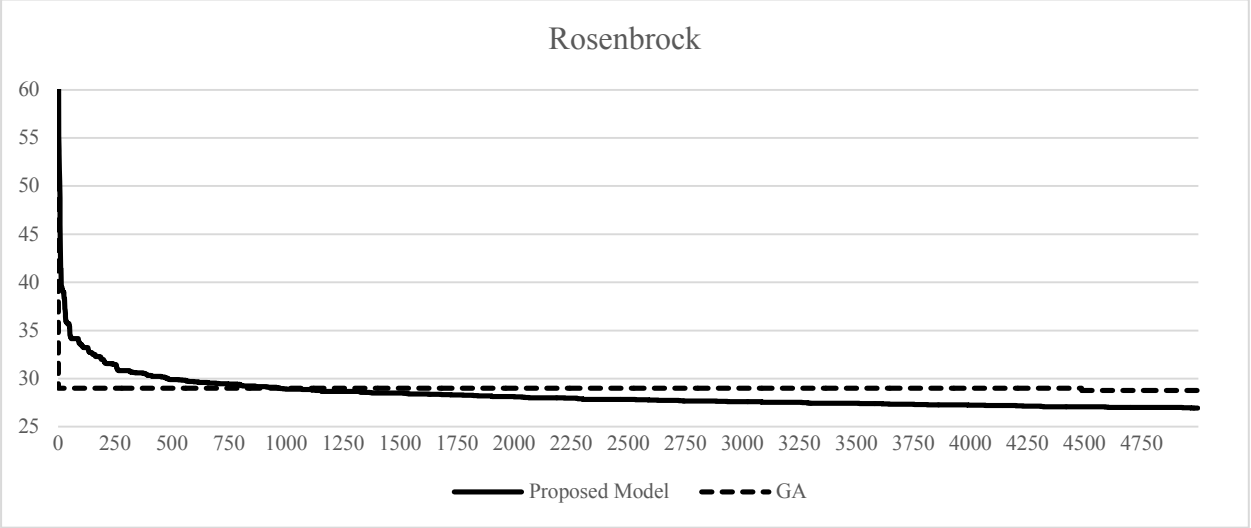

Fig. 9: Convergence curves for Rosenbrock function

For solution quality, we compared with several state-of-the-art algorithms in the literature. For the low-dimensional functions, results were compared with those of Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC), Spider Monkey Optimization (SMO), WCA, and Evaporation rate based WCA (ER-WCA). The results for PSO, ABC, and SMO were taken from Bansal et al. [22] while results for WCA and ER-WCA were taken from Sadollah et al. [12]. For the high-dimensional functions, results were compared against those from Real-Coded Chemical Reaction Optimization (RCCRO) [11], Social Spider Optimization (SSO), Artificial Bee Colony (ABC), Fruit Fly Optimization (FFO), Improved Fruit Fly Optimization (IFFO), and Harmony Search (HS). Results for SSO and ABC were obtained from Cuevas et al. [23], while those for FFO, IFFO and HS were taken from Pan et al. [9].

Tables 4 and 5 show the performance in terms of the average solution and standard deviation. The symbol "-" denotes that the algorithm was not tested on that function. The numbers in bold are the best solutions among all algorithms in each test function. For the low-dimensional functions, Table 4 shows that our algorithm outperformed other algorithms with respect to the percentage of global optima found. It was able to find the global optima in 10 out of 13 test functions, i.e. F1, F2, F3, F4, F5, F8, F9, F10, F11, and F13. ER-WCA, which came in second, successfully found the global optimum for 1 out of 9 functions tested while PSO, ABC, SMO, and WCA were not able to locate the global optimum in any of the functions tested. For functions F6 and F7 (for which no algorithm was able to find the global optimum), our algorithm exhibited the best performance for function F6 and the worst performance for function F7. However, for these 2 functions, differences between the best and the worst were very small. Moreover, it should be mentioned that our algorithm was very robust to the initial starting position, as its standard deviation was very small for all functions.

For the high-dimensional functions, Table 5 shows that our algorithm performed better than other six algorithms we considered. It was able to find the global optima in 9 out of 10 test functions. RCCRO, FFO, and IFFO, which were tied for second place, managed to find the global optimum in only 1 function. For the function F1, none of the algorithms was able to converge to the global optimum. For the average solution, our algorithm achieved the best result, edging out RCCRO. However, our algorithm performed much more consistently than RCCRO as indicated by the lower standard deviation. Moreover, results on multimodal functions (F7 – F10) confirmed that our algorithm was able to escape from local optima and converged to global optima.

Table 4: Experimental results for low-dimensional functions

	Our Algorithm	**PSO**	**ABC**	**SMO**	**WCA**	**ER-WCA**
F1	**-1.0316285 (0)**	-1.0311485 (3.05E-04)	-1.0311205 (3.11E-04)	-1.0312265 (2.99E-04)	-1.0316 (1.38E-08)	-1.0316 (7.89E-10)
F2	**0.39788736 (0)**	0.39836836 (2.82E-04)	0.39836436 (2.79E-04)	0.39831236 (2.91E-04)	0.398272 (3.20E-04)	0.398193 (2.94E-04)
F3	**3.00 (0)**	3.000483 (2.70E-04)	3.000488 (3.08E-04)	3.000485 (2.96E-04)	3.000561 (2.55E-04)	3.000494 (3.08E-04)
F4	**0 (0)**	4.22E-06 (2.67E-06)	7.81E-06 (2.42E-06)	4.81E-06 (2.58E-06)	-	-
F5	**-1.00 (0)**	-0.9999999 (2.87E-14)	-0.9999999 (5.64E-07)	-0.9999999 (2.69E-14)	-	-
F6	**-24776.51834 (0)**	-24776.509 (5.64E-03)	-24776.510 (5.25E-03)	-24776.511 (4.98E-03)	-	-
F7	-186.71340 (0.02)	-186.730807 (4.01E-04)	-186.7309027 (5.95E-06)	**-186.7309030 (5.58E-06)**	-	-
F8	**3905.93 (0)**	-	-	-	3905.940137 (0.0382)	3905.934711 (9.83E-03)
F9	**0 (0)**	-	-	-	4.16E-04 (3.11E-04)	2.45E-04 (2.55E-04)
F10	**0 (0)**	-	-	-	1.16E-03 (2.58E-03)	2.46E-04 (2.62E-04)
F11	**1.74 (0)**	-	-	-	1.7441 (1.96E-06)	1.7444 (1.08E-03)
F12	9.19E-07 (4.77E-07)	-	-	-	1.58E-06 (7.60E-06)	**0 (0)**
F13	**0 (0)**	-	-	-	6.09E-10 (8.29E-10)	4.57E-29 (2.48E-28)

Table 5: Experimental results for high-dimensional functions

	Our Algorithm	**RCCRO**	**SSO**	**ABC**	**FFO**	**IFFO**	**HS**
F1	**26.96094 (0.388366)**	27.06 (34.27)	114 (39)	138 (155)	767 (881)	73.40 (272)	196 (619)
F2	**0 (0)**	9.32E-03 (3.66E-03)	-	-	2.59 (3.37)	2.87E-06 (3.73E-07)	7 (0.966)
F3	**0 (0)**	**0 (0)**	2.68E-03 (6.05E-04)	4.06E-03 (2.98E-03)	**0 (0)**	**0 (0)**	3 (2.07)
F4	**0 (0)**	-	-	-	1.04 (1.99)	4.90E-15 (2.15E-14)	6.12E-09 (1.08E-08)
F5	**0 (0)**	6.43E-07 (2.10E-07)	1.96E-03 (9.96E-04)	2.90E-03 (1.44E-03)	5.22 (0.599)	4.96E-13 (3.18E-13)	7.07 (3.24)
F6	**0 (0)**	2.20E-03 (4.34E-04)	0.0137 (3.11E-03)	0.135 (0.0801)	136 (425)	2.33E-06 (4.25E-07)	0.0859 (0.0539)
F7	**0 (0)**	-	0.274 (0.0517)	4.14 (0.469)	0.40 (0.0474)	1.60 (0.309)	1.80 (0.297)
F8	**0 (0)**	9.08E-04 (2.88E-04)	8.59 (1.11)	26.40 (10.60)	286 (34)	6.34E-11 (0.182)	1.03 (0.756)
F9	**0 (0)**	0.0112 (0.0162)	3.29E-03 (5.49E-04)	0.0522 (0.0342)	126 (51.9)	0.0123 (0.0162)	1.09 (0.0318)
F10	**0 (0)**	1.94E-03 (4.19E-04)	0.0136 (2.36E-03)	0.653 (0.309)	20.50 (0.145)	5.13E-07 (9.35E-08)	1.04 (0.341)

5.0 CONCLUSION

We presented a new swarm optimization algorithm based on the cooperative behavior of three different kinds of ants in a colony. To obtain a good starting point, the initial nests were evenly distributed throughout the search space. For each nest, the offspring were created by using an arithmetic crossover operation. Then the foraging process was used to refine the quality of the search results. We test our algorithm on 23 well-known benchmark functions. Our algorithm outperformed all the other algorithms in terms of convergence rate and solution quality. Regarding the convergence rate, our algorithm converged to the global optima in less than 10 iterations for most functions, whereas GA used several hundred to several thousand iterations. Regarding the solution quality, our algorithm successfully found the global optima in more than 80 percent of the test functions, whereas the second-place algorithm only found around 10 percent of the functions tested.

REFERENCES

[1] A. D. Belegundu, T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering. 2nd ed*. Cambridge University Press, 2011.

[2] J. Kennedy, R. Eberhart, "Particle Swarm Optimization", in *Proceedings of the IEEE International Conference on Neural Networks*, 1995, pp. 1942–1948.

[3] M. Dorigo, V. Maniezzo, A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents", *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, Vol. 26 Issue 1, pp. 29-41.

[4] D. Simon, "Biogeography-based Optimization", *IEEE Transactions on Evolutionary Computation*, Vol. 12, pp. 702–713.

[5] W. Gong, Z. Cai, C. X. Ling, H. Li, "A Real-coded Biogeography-based Optimization with Mutation", *Applied Mathematics and Computation*, Vol. 216, pp. 2749–2758.

[6] A. H. Gandomi, A. H. Alavi, "Krill Herd: A New Bio-inspired Optimization Algorithm", *Communications in Nonlinear Science and Numerical Simulation*, Vol. 17, pp. 4831–4845.

[7] B. Niu, H. Wang, "Bacterial Colony Optimization", *Discrete Dynamics in Nature and Society*, Vol. 2012.

[8] W. T. Pan, "A New Fruit Fly Optimization Algorithm: Taking the Financial Distress Model as an Example", *Knowledge-Based Systems*, Vol. 26, pp. 69-74.

[9] Q. K. Pan, H. Y. Sang, J. H. Duan, L. Gao, "An Improved Fruit Fly Optimization Algorithm for Continuous Function Optimization Problems", *Knowledge-Based Systems*, Vol. 62, pp. 69–83.

[10] R. V. Rao, V. J. Savsani, D. P. Vakharia, "Teaching-Learning-Based Optimization: An Optimization Method for Continuous Non-linear Large Scale Problems", *Information Sciences*, Vol. 183, pp. 1–15.

[11] A. Y. S. Lam, V. O. K. Li, J. J. Q. Yu, "Real-coded Chemical Reaction Optimization", *IEEE Transactions on Evolutionary Computation*, Vol. 16, pp. 339–353.

[12] A. Sadollah, H. Eskandar, A. Bahreininejad, J. H. Kim, "Water Cycle Algorithm with Evaporation Rate for Solving Constrained and Unconstrained Optimization Problems", *Applied Soft Computing*, Vol. 30, pp. 58–71.

[13] H. Eskandar, A. Sadollah, A. Bahreininejad, M. Hamdi, "Water Cycle Algorithm - A Novel Metaheuristic Optimization Method for Solving Constrained Engineering Optimization Problems", *Computers and Structures*, Vol. 110–111, pp. 151–166.

[14] M. Ghaemi, M. R. Feizi-Derakhshi, "Forest Optimization Algorithm", *Expert Systems with Applications*, Vol. 41, pp. 6676–6687.

[15] B. Hölldobler, E. O. Wilson, *The Ants. 1st ed*. Harvard University Press, 1990.

[16] R. Wehner, C. Meier, C. Zollikofer, "The Ontogeny of Foraging Behaviour in Desert Ants, Cataglyphis bicolor", *Ecological Entomology*, Vol. 29, pp. 240–250.

[17] P. Schultheiss, A. Wystrach, E. L. G. Legge, K. Cheng, "Information Content of Visual Scenes Influences Systematic Search of Desert Ants", *The Journal of Experimental Biology*, Vol. 216, pp. 742–749.

[18] S. Bolek, H. Wolf, "Food Searches and Guiding Structures in North African Desert Ants, Cataglyphis", *Journal of Comparative Physiology A*, Vol. 201, pp. 631–644.

[19] N. J. R. Plowes, K. Ramsch, M. Middendorf, B. Hölldobler, "An Empirically Based Simulation of Group Foraging in the Harvesting Ant, Messor Pergandei", *Journal of Theoretical Biology*, Vol. 340, pp. 186–198.

[20] X. Yao, Y. Liu, G. Lin, "Evolutionary Programming Made Faster", *IEEE Transactions on Evolutionary Computation*, Vol. 3, pp. 82–102.

[21] M. Jamil, X. S. Yang, "A Literature Survey of Benchmark Functions for Global Optimization Problems", *International Journal of Mathematical Modelling and Numerical Optimisation*, Vol. 4, pp. 150–194.

[22] J. C. Bansal, H. Sharma, S. S. Jadon, M. Clerc, "Spider Monkey Optimization Algorithm for Numerical Optimization", *Memetic Computing*, Vol. 6, pp. 31–47.

[23] E. Cuevas, M. Cienfuegos, D. Zaldívar, M. Pérez-cisneros, "A Swarm Optimization Algorithm Inspired in the Behavior of the Social-spider", *Expert Systems with Applications*, Vol. 40, pp. 6374–6384.