Particle Swarm Optimization – Artificial Bee Colony Chain (PSOABCC): A Hybrid Meteahuristic Algorithm

Oğuz Altun
Department of Computer Engineering
Yildiz Technical University
Istanbul, Turkey
oaltun@yildiz.edu.tr

Tarık Korkmaz
Department of Computer Engineering
Epoka University
Tiran, Albania
tarikkorkmaz5@hotmail.com

***Abstract* — A metaheuristic is a search strategy for finding optimal solutions in blackbox optimization problems. Artificial Bee Colony Algorithm (ABC) is a metaheuristics optimization algorithm mimicking the mobility of bees in a bee hive. Particle Swarm Optimization (PSO) is another metaheuristics optimization algorithm inspired by the behaviour of swarms searching for food sources.**

In this work we build a hybrid algorithm that we call 'Particle Swarm Optimization – Artificial Bee Colony Chain' (PSOABCC) that repeatedly apply ideas from PSO and ABC algorithms in a loop.

To ensure that all algorithms themselves have optimal parameters in the experiments, we optimize parameters of PSO, ABC, and PSOABC algorithms with Cuckoo Search optimization algorithm.

The hybrid algorithm shows better convergence performance than the other two algorithms on the two dimensional sphere, rosenbrock, and rastrigin functions.

Keywords— Metaheuristics; optimization; artificial bee colony; particle swarm optimization; Particle Swarm Optimization – Artificial Bee Colony Chain.

I. INTRODUCTION

Metaheuristics are algorithms for finding an optimal value in black box optimization problems. Swarm based metaheuristic algorithms are class of metaheuristics inspired by intelligent behavior of swarms. Amongst swarm based metaheuristic algorithms, Particle Swarm Optimization (PSO) [4] inspires from the naturalistic behavior of birds living in flocks as they gather information on the landscape and transmit the needed information to the other spread flock of birds as in groups' for reaching the best. Artificial Bee Colony (ABC) [6] inspires from the intelligent behavior of a bee colony while searching for better food sources. Cuckoo Search (CS) [5] inspires from the behavior of cuckoos while they look for ideal host nests to lay their eggs.

In this work we build a hybrid algorithm chaining PSO and ABC algorithms. We call the hybrid algorithm we build 'Particle Swarm Optimization - Artificial Bee Colony Chain' and we abbreviate that name as PSOABCC. We compare the success of PSO, ABC, and PSOABCC on three numerical function optimization problems. As each of the PSO, ABC, and PSOABCC have their own parameters that affect their behavior, before doing the comparison, we optimize those parameters of the PSO, ABC, and PSOABC themselves. We prefer to use a metaheuristics different from the algorithms compared (PSO, ABC, and PSOABCC) for this purpose, and decide on using Cuckoo Search (CS), a new metaheuristics algorithm with promising results [5].

The rest of the paper is organized as follows: In Section II we introduce the terminology used in the paper to prevent possible confusion for the reader that is accustomed to different terms used in the literature. In Section III we discuss the Particle Swarm Optimization algorithm. In Section IV we give the pseudo code of the Artificial Bee Colony algorithm. In Section V we summarize the Cuckoo Search algorithm. In Section VI we introduce the hybrid Particle Swarm Optimization – Artificial Bee Colony Chain. In Section VII we review the test functions used. In Section VIII we discuss a method for optimizing parameters of the algorithms themselves. We conclude by the comparison and discussions in Section IX and the summary in Section X.

II. TERMINOLOGY

The algorithms we discuss in this paper find the maximum value of a given function f in a search space (e.g. food source with maximum amount of food). In accordance, we use the word 'fitness' (and also sometimes 'height' or 'f value', interchangeably) for the value of the function, and design the problems as maximization of these functions (explained in detail in section VII).

We follow [1] and use the term 'function assessment' for obtaining the value (or fitness, or height) of the function f on a given coordinate in the search space. Some other works (e.g. [2]) use the term 'function evaluation' for the same purpose. All the algorithms discussed in this work keep track of the number of function assessments done so far during its running time in a variable called $assessmentcount$, and

break out of its main loop when maximum allowed number of function assessments ($maxassessments$) is reached (e.g. when the condition $assessmentcount \leq maxassessments$ holds).

III. PARTICLE SWARM OPTIMIZATION (PSO) ALGORITHM

Particle Swarm Optimization [3] algorithm, whose pseudo code is given in Fig. 5, utilizes the idea of birds searching for food sources. Each bird has a velocity and a personal best position. On each step velocity of each bird is recalculated using current velocity, its personal best, and the global best as in (1). The new position of the bird on the step is calculated using this new velocity. Depending on the parameters , $psip$, and $psig$ the algorithm can behave more exploitative or explorative.

PSO(f, $maxassessments$, n):
1. // inputs are f, function to be maximized, $maxassessments$, the number of assessments algorithm allows before breaking out main loop, and n, number of particles
2. // output is g, the coordinate with the best fitness (f value, height) in the search space
3. Assign value 0 to the variable $assessmentcount$. Whenever the fitness is obtained (calling the function f), the $assessmentcount$ is incremented by 1.
4. For each particle
    a. Assign a random position x
    b. Accept that random position as the personal best p of this particle.
    c. Assign a random velocity v
5. Scan positions of all n particles and select position with the best f value as the global best g.
6. Repeat while $assessmentcount \leq maxassessments$
    a. For each particle
        i. Calculate new velocity v using (1).
        ii. Calculate new position x using $x = x + v$
        iii. If the new position x has a better fitness than personal best p, assign x to p
        iv. If the new position x has a better fitness than the global best g, assign x to g
7. Return g, the global best position

Fig. 1 the Particle Swarm Optimization (PSO) algorithm used in our comparisons

In the pseudo code in Fig. 1 new velocity of a particle is calculated by (1) where v is the velocity, w, $psip$, and $psig$ are weight scalars, and rp and rg are random values between 0 and 1.

$$v = w * v + rp * psip * (p - x) + rg * psig * (g - x) \quad (1)$$

IV. ARTIFICIAL BEE COLONY

Artificial Bee Colony (ABC), whose pseudo code is given in Fig. 2, is a metaheuristic that uses ideas from the behaviour of a bee hive. The algorithm starts with n food sources and corresponding n employed bees in the location of these food sources. The employed bees checkout neighbour positions and move there if those positions are better. Later, employed bees return to the hive and inform onlooker bees of the location and quality of the food sources they worked on. A food source may be chosen to be re-visited by an onlooker bee with probability proportional to its quality (fitness). When a food source is checked for better quality neighbours more than the allowed number ($limit$) and no better neighbour is found, the food source is abandoned, and a scout bee finds a random new food source instead of the abandoned one. The algorithm returns the best food source at the end.

ABC(f, $assessmentcount$, n):
1. // inputs are f, function to be maximized, $maxassessments$, the number of assessments algorithm allows before breaking out main loop, and n , number of food sources (equally, number of employed bees)
2. // output is g, the coordinate with the best fitness (f value, height) in the search space
3. // assume we have n food sources and n corresponding employed bees.
4. Assign value 0 to the variable $assessmentcount$. Whenever the fitness is obtained (calling the function f), the $assessmentcount$ is incremented by 1.
5. For each food source assign a random position x. Assume each employed bee is on one of these food sources, and measuring its quality (fitness).
6. Repeat while $assessmentcount \leq maxassessments$
    a. // Employed bee phase
    b. For each food source:
        i. Tweak the position x of the bee employed to get a new position
            1. If the new position has a better fitness accept the new position (move the bee to new position).
            2. Otherwise increase trial value of the food source

by one.

c. // Dance phase: Assume each of the employed bees are returned to the hive, and informed the quality of the corresponding food source (with a 'dance') to the onlooker bees.
d. Assign a probability r of being selected by onlooker bees to each food source. r should be such that the food source with higher fitness has more probability of being chosen.
e. // Onlooker bee phase:
f. For each employed bee (that returned to the hive and did a dance)
    i. With probability r, let its food source be visited by an onlooker bee. Onlooker bee checks fitness of a position in the neighborhood of employed bee position.
        1. If the new position is better accept the new position (move the bee to new position). Onlooker bee turns into an employed bee.
        2. Otherwise increase trial number of the food source by one.
g. // Scout bee phase.
h. For each food source position which trial number exceeds the $limit$ parameter:
    i. Let a scout bee randomly determine a new position and chose the neighborhood as the food source. Let the scout bee turn into an employed bee.

7. Scan the latest food source (employed bee) positions, and return the one with best fitness as the global best g.

Fig. 2 the Artificial Bee Colony (ABC) algorithm used in our comparisons

V. CUCKOO SEARCH (CS) ALGORITHM

Cuckoo Search (CS) [5] metaheuristic algorithm, inspires from the breeding behavior of cuckoos. Instead of building and maintaining their own nests, and raising their own babies, cuckoos lay eggs to nest of an unaware other bird (a.k.a. host nest). The host bird raises cuckoo baby as its own.

In the Cuckoo Search pseudo code given in Fig 3., each cuckoo looks for a new nest in each iteration of the main loop. If the new nest has better quality, it lays an egg there this time. The quality of host nest is assumed to be equivalent to the value of the function f to be maximized at that position in search space. Some of the host birds discover that there is an alien egg in their nest, and get rid of it. The cuckoo abandons this nest, and lays an egg to a new host it finds. The new nests are found using Levy Flights [5].

CS(f, $assessmentcount$, n):

1. // inputs are f, function to be maximized, $maxassessments$, the number of assessments algorithm allows before breaking out main loop, and n , number of nests (equally, number of cuckoos)
2. // output is g, the coordinate with the best fitness (f value, height) in the search space
3. Assign value 0 to the variable $assessmentcount$. Whenever the fitness is obtained (calling the function f), the $assessmentcount$ is incremented by 1.
4. For each cuckoo assign a random nest (position).
5. Repeat while $assessmentcount \leq maxassessments$
    a. // a new breeding season has begun
    b. For each cuckoo:
        i. Find a new nest, using Levy Flights [5].
        ii. If the new nest has a better fitness than the current nest, this year lay egg to new nest. Otherwise, lay egg to old nest.
    c. Abandon pa percent of the worst nests. The eggs on these nests are discovered by host bird.
    d. For each of the abandoned nests:
        i. Find a new nest, using Levy Flights [5].
        ii. Lay an egg to the new nest.
6. Scan the nests and return g, the nest with best fitness.

Fig. 3 Cuckoo Search (CS) algorithm used for optimizing parameters of PSO, ABC, and PSOABCC.

VI. PARTICLE SWARM OPTIMIZATION – ARTIFICIAL BEE COLONY CHAIN (PSOABCC) ALGORITHM

We build a hybrid algorithm that we call 'Particle Swarm Optimization – Artificial Bee Colony Chain' (PSOABCC) as given in Fig. 4. The algorithm starts by building random initial personal best positions for particles/bees. Then in the main

loop it improves the personal bests using PSO_PHASE (Fig. 5) and ABC_PHASE (Fig. 6) methods. When the main loop ends, the algorithm scans the latest personal best positions and returns the best of them as the global best.

The PSO_PHASE method (Fig. 5) takes a list of best positions of particles, updates them with better positions, and returns back. In contrast with the personal bests, a new random current position is assigned for each particle. The main loop of the method runs a fixed $niterpso$ times. The rest of the method behaves like the normal PSO algorithm in Fig. 1, except the output. PSO_PHASE outputs the modified set of personal bests back, whereas PSO algorithm returns a single global best position.

The ABC_PHASE method (Fig. 6) takes an initial list of employed bee (equivalently food source) positions as a parameter, improves them, and returns them back. In the PSOABCC this list of positions comes from the personal bests of the previous PSO_PHASE. This ensures that PSO_PHASE and ABC_PHASE incrementally improve the same list of personal best positions. The main loop of the method runs a fixed $niterabc$ times. The rest of the method behaves like the normal ABC algorithm in Fig. 2, except what it outputs. While the ABC_PHASE returns the list of current food source (employed bee) positions, the ABC algorithm returns the best of these food source positions as the global best.

PSOABCC(f, $assessmentcount$, n):

1. //inputs are f, function to be maximized, $maxassessments$, the number of assessments algorithm allows before breaking out main loop, and n, number of particles (equally, number of food sources, equally, number of employed bees)
2. //output is g, the coordinate with the best fitness (f value, height) in the search space
3. Assign value 0 to the variable $assessmentcount$. Whenever the fitness is obtained (calling the function f), the $assessmentcount$ is incremented by 1.
4. For each particle/employed bee assign a random initial position.
5. For each particle/employed bee assign initial position as the personal best. Let set of personal best values be P.
6. Repeat while $assessmentcount \leq maxassessments$
    a. Improve personal bests using PSO_PHASE as shown in Fig. 5: P <- PSO_PHASE(P)
    b. Improve personal bests using ABC_PHASE as shown in
    c. Fig. 6: P <- ABC_PHASE(P)
7. Scan the set of personal bests and return the one with the best fitness as the global best g.

Fig. 4 Particle Swarm Optimization – Artificial Bee Colony Chain (PSOABCC) algorithm pseudocode

PSO_PHASE(P):

8. // input: $P = \{p_1, p_2, ...\}$ (initial best positions of the particles)
9. // output: $P = \{p_1, p_2, ...\}$ (final best positions of the particles)
10. For each particle
    a. Assign a random position x
    b. Update the personal best if x has better fitness than p
    c. Assign a random velocity v
11. Select position with the best fitness as global best g.
12. Repeat while we iterated less then $niterpso$:
    a. For each particle
        i. Calculate new velocity v using (1).
        ii. Calculate new position x using $x = x + v$
        iii. If the new position x has a better fitness than personal best p, assign x to p
        iv. If the new position x has a better fitness than the global best g, assign x to g
13. Return set of particle best positions P

Fig. 5. Particle swarm optimization phase (PSO_PHASE)

ABC_PHASE(X):

1. // input: $X = \{x_1, x_2, ...\}$ (initial employed bee positions)
2. // output: $X = \{x_1, x_2, ...\}$ (final employed bees positions)
3. While we iterated less than $niterabc$:
    a. // Employed bee phase
    b. For each food source:
        i. Tweak the position x of the bee employed to get a new position
            1. If the new position has a better fitness accept the new position (move the bee to new position).
            2. Otherwise increase trial value of the food source by one.
    c. // Dance phase:
    d. Assign a probability r of being selected by onlooker bees to each employed bee. r should be such that the employed bee with position that has better fitness has more probability of being chosen.
    e. // Onlooker bee phase:
    f. For each employed bee (that returned to the hive and did a dance)
        i. With probability r, let its food source be visited by an onlooker bee. Onlooker bee checks fitness of a position in the neighborhood of employed bee position.
            1. If the new position is better accept the new position (move the bee to new position). Onlooker bee turns into an employed bee.
            2. Otherwise increase trial number of the food source by one.
    g. // Scout bee phase.
    h. For each food source position which trial number exceeds the $limit$ parameter:
        i. Let a scout bee randomly determine a new position and chose the neighborhood as the food source. Let the scout bee turn into an employed bee.
4. Return X

Fig. 6 Artificial bee colony phase (ABC_PHASE)

VII. TEST FUNCTIONS

We have evaluated algorithms on following test functions.

A. Negative Rastrigin Function

Negative Rastrigin Function is the "negative" of Rastrigin Function [1] and can be defined by (2).

$$\text{Maximize } f(<x1,\dots,xn>) = -10n - \sum_{i=1}^{n} {x_i}^2 - 10\cos 2\pi x_i \quad (2)$$

In (2) n is number of dimensions, x_i is value of the solution vector in dimension i. Global maximum value is 0 and is on (0,0,..,0) . Initialization range of the function is $[-5.12, 5.12]$.

B. Negative Rosenbrock Function (Ros)

The Negative Rosenbrock (Ros) Function is the "negative" of the Rosenbrock Function [3] and can be defined by (3).

$$\text{Maximize } f(<x1,\dots,xn>) = -\sum_{i=1}^{n-1}(1 - x_i)^2 + 100(x_{i+1}x_i^2)^2 \quad (3)$$

In (3) n is number of dimensions, and x_i is value of the solution vector in dimension i. Global maximum value is 0 and is on (1,1,...,1). The initialization range of the function is $[-2.048, 2.048]$.

C. Negative Sphere Function (Sph)

Negative Sphere Function (Sph) is the negative of Sphere Function [1] and can be defined by (4).

$$\text{Maximize } f(<x1,\dots,xn>) = -\sum_{i=1}^{n} {x_i}^2 \quad (4)$$

In (4) n is number of dimensions, x_i is value of the solution vector in dimension i. Maximum height is 0 and is on (0,0,…,0). The initialization range is $[-5.12, 5.12]$.

VIII. OPTIMIZING ALGORITHM PARAMETERS USING CUCKOO SEARCH

Metaheuristic algorithms PSO, ABC, and PSOABC themselves have parameters that affect their behavior. In order to compare each algorithm at its best success rate, we optimize the parameters of the algorithms before we compare them. For this purpose we use a fourth algorithm, Cuckoo Search (CS), given in Fig. 3, as the optimizer.

For this purpose we define the parameter optimization fitness function F in Fig. 7. In contrast to simpler fitness functions we defined in Section VII, this function has an $algorithm$ (e.g. PSO), and a simple function f (e.g. Negative Rastrigin Function discussed in Section VII.A) to be solved by $algorithm$. F takes the list of parameter values $params$ (e.g. $\{w = 0.3, psip = 0.5, psig = 0.4\}$) as coordinates, and returns a fitness value. For this, the $algorithm$ solves the simple function f m times with given parameter value set $params$, to get m different best positions. Each best position has a corresponding f value (fitness). The mean of these fitness values is returned as a means of how well the parameter value set $params$ did with $algorithm$ and f. Running m independent runs and taking the mean is necessary because all the algorithms we are interested in comparing (PSO, ABC, and PSOABCC) behave according to randomly generated values.

We give F as a parameter to CS as a function to be optimized. This way we actually optimize the parameter set of the $algorithm$. Any metaheuristic algorithm could be used for this purpose, we choose CS because a) it is a different algorithm than PSO, ABC, and PSOABCC, b) good results with CS are reported in [5].

$F(algorithm, f; params)$:

1. $ntrials$: <- 1
2. while $i \leq m$:
   a. g_i <- $algorithm(f, params)$
   b. v_i <- $f(g_i)$
3. s <- $v_1 + v_2 + \cdots + v_m$
4. $meanh$ <- $\frac{s}{m}$
5. Return $meanh$

Fig. 7 A fitness function for optimizing metaheuristics algorithms

For the PSO algorithm, we optimize the parameters w, $psip$, and $psig$ used in (5). For ABC algorithm and PSOABCC algorithm we optimize the $limit$ variable used in their scout bee phases. For PSOABCC algorithm we optimize parameters $niterpso$ and $niterabc$ that affect breaking out of main loops of PSO_PHASE and ABC_PHASE. Each algorithm has different optimal parameters for each fitness function f. Hence, we re-optimize each parameter set on each test function.

We prefer not giving the optimized parameter values here as we think the values we found may be local optima.

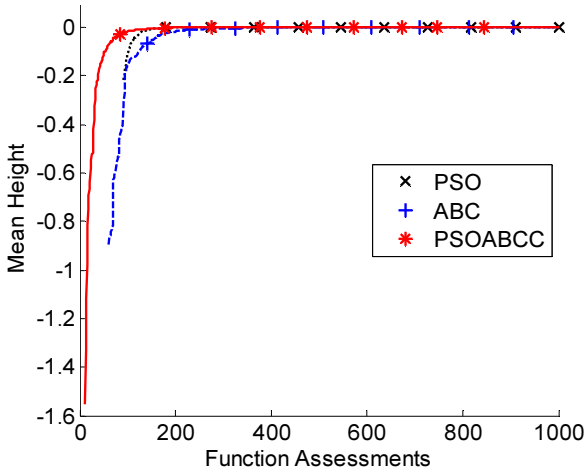
IX. OPTIMIZATION COMPARISON AND DISCUSSION

To compare optimization performances of PSO, ABC, and PSOABCC, we do 100 independent runs of each of these algorithms on each test function and plot the mean convergence graph. Each algorithm was terminated after reaching 1000 function assessments (function evaluations) to make a fair comparison. On each of the test functions we tested, the PSOABCC is clearly converges faster than the other two algorithms.

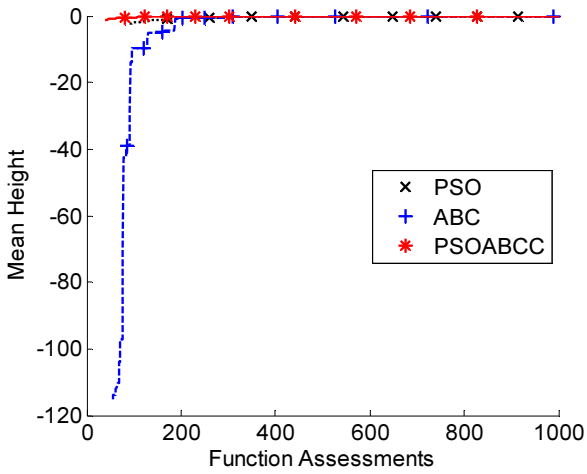
On Negative Sphere Function (Fig. 8a) which is a unimodal function, all the algorithms converge before 300 assessments, and the order of convergence is PSOABCC>PSO>ABC. Hence PSOABCC converges faster than the other two.

On Negative Rosenbrock Function (Fig. 8b) which is a unimodal function, all three functions converge around 200 function assessments. Again PSOABCC is the fastest.

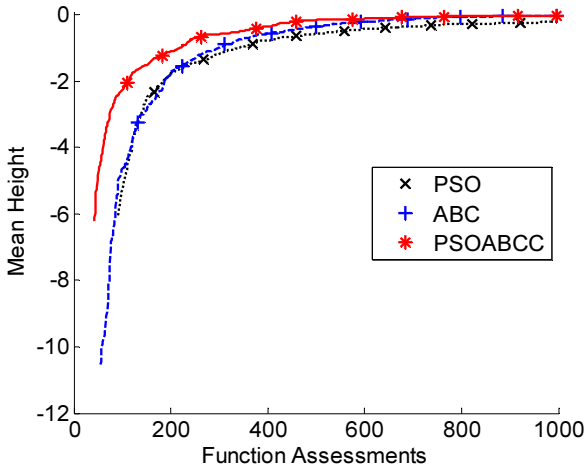
On Negative Rastrigin Function (Fig. 8c) which is a multimodal function, PSO keeps stuck in a local optima and can not reach to the global optimum even after 1000 assessments. After 800 assessments ABC catches on PSOABCC, but PSOABCC clearly converges faster.



(a) Negative Sphere Function



(b) Negative Rosenbrock Function



(c) Negative Rastrigin Function

Fig. 8 Convergence graphs of PSO, ABC and PSOABCC algorithms on (a) Negative Sphere Function, (b) Negative Rosenbrock Function, (c) Negative Rastrigin Function. Each convergence plot shows the mean height of 100 independent runs against number of function assessments (function evaluations).

X. SUMMARY

We chain the Particle Swarm Optimization (PSO) and Artificial Bee Colony (ABC) algorithms to obtain a hybrid we call Particle Swarm Optimization – Artificial Bee Colony Chain (PSOABC). We compare the three algorithms PSO, ABC, and PSOABC on three well known test functions: Negative Sphere Function, Negative Rosenbrock Function, and Negative Rastrigin Function.

We pre-optimized the parameters of the algorithms themselves on each function using another algorithm, Cuckoo Search, since each algorithm behave different according to these input parameter values.

We compare the convergence graphs of PSO, ABC, and PSOABCC on each function. The convergence graphs show that PSOABCC converges faster from the two other algorithms.

Testing the algorithm further on other test and real world problems and looking for algorithmic improvements remains as future work.

XI. REFERENCES

[1] S. Luke, Essentials of metaheuristics, 2012.

[2] X. Yan, Z. Yunlong and Z. Wenping, "A Hybrid Artificial Bee Colony Algorithm for Numerical Function Optimization," in *11th International Conference on Hybrid Intelligent Systems (HIS)*, 2011.

[3] E. James and K. Russell, "Particle Swarm Optimization," *1995 IEEE International Conference on Neural Networks,* vol. 4, p. 1942–1948, 1995.

[4] M. Melanie, An Introduction to Genetic Algorithms, MIT Press, 1998.

[5] R. Storn and K. Price, "Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces," In Technical Report TR-95-012, Berkley, 1995.

[6] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm," *Journal of Global Optimization,* vol. 3, p. 459–471, 2007.

[7] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," In Technical Report TR-06, Kayseri/Türkiye, 2005.

[8] A. Eiben and J. Smith, Introduction to evolutionary computing, Springer, 2003.

[9] X.-S. Yang and S. Deb, "Engineering optimisation by Cuckoo Search," *Int. J. Mathematical Modelling and Numerical Optimisation,* vol. 1, no. 4, pp. 330-343, 2010.

[10] R. Eberhart, Y. Shi and J. Kennedy, Swarm intelligence, Morgan Kaufmann, 2001.