Indonesian Journal of Electrical Engineering and Computer Science
Vol. 3, No. 1, July 2016, pp. 164 ~ 173
DOI: 10.11591/ijeecs.v3.i1.pp164-173

An Algorithm for Continuous Optimization Problems using Hybrid Particle Updating Method

PB Shola, LB Asaju
Department of Computer Science, University of Ilorin, Ilorin, Nigeria
Corresponding author, e-mail: shola.bp@unilorin.edu.ng, lbasaju@unilorin.edu.ng

Abstract

Optimization problem is one such problem commonly encountered in many area of endeavor, obviously due to the need to economize the use of the available resources in many problems. This paper presents a population-based meta-heuristic algorithm for solving optimization problems in a continous space. The algorithm, combines a form of cross-over technique with a position updating formula based on the instantaneous global best position to update each particle position .The algorithm was tested and compared with the standard particle swarm optimization (PSO) on many benchmark functions. The result suggests a better performance of the algorithm over the later in terms of reaching (attaining) the global optimum value (at least for those benchmark functions considered) and the rate of convergence in terms of the number of iterations required reaching the optimum values.

Keywords*: population, continous, optimization, meta-heuristics, search.*

Copyright © 2016 Institute of Advanced Engineering and Science. All rights reserved.

1. Introduction

Optimization problems are common problems that arise in many areas of life, from science and engineering, to medicine and business among others. These problems arise from the need to find the best way to use the available resources (time, store, money etc.,) to meet certain life needs. Many of these problems occur naturally. Examples of this include finding a way to minimize the production cost of items, identifying the shortest path connecting two or more places and locating the 'highest' or 'lowest' point in an area. Some others arise from problems that are non naturally optimization problems being transformed or cast into such problems. An example is the problem of finding the root of an equation or a system of equations.

Though common and age-long, optimization problems have defiled all attempt to produce a single method that could solve them all. In fact it has been shown in [1] that no such algorithm could be found. The result of this is a 'daily' increase in the number of methods being devised for the problem, (each being able to solve only some kinds of the problem but unable to handle others) resulting in many optimization methods available from which a user could (and indeed responsible to) make a choice of a right (optimization) method for the right optimization problem. The directions of attack of optimization problems are usually to develop a new method that could solve some optimization problems that the others are unable to solve or at enhancing the effectiveness of the existing ones. The other way is to produce a method that has a wide (or wider) area of effectiveness or applicability.

We propose here a method for solving optimization problems in a continuous domain capable of locating the global optimum point of a wide varieties of functions. It is a population based hybrid method where each agent in the population employs both an evolutionary operation and a swarm-based updating method to determine its next move. An hybrid approach involves the appropriate merging of two or more methods (or concepts) in a way that they strengthen each other: a stronger one possibly taking position where the others are weaker. By this device the area of applicability or effectiveness of the hybrid is wider than those of its constituent.

The method proposed is meta-heuristic based. "A meta-heuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a

Received February 2, 2016; Revised May 25, 2016; Accepted June 10, 2016

collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures or a simple local search or just a construction method" [2].

Meta–heuristic approach is preferred because of its beautiful properties:

1. It does not impose any condition on the objective function (and could be tried on any objective function). In order words it i general purpose.
2. It does not require (the user to supply) starting values unlike the traditional methods such as gradient method, or Newton method.
3. It does not require the gradient of the objective function at any point. This is unlike the traditional methods such as the Newton and gradient methods.
4. It converges faster than these other methods especially on large-scale problems. The other methods (deterministic or exact) could hardly solve such problem in a reasonable amount of time.
5. It can escape from a local optimum into which it might fall.
6. It accommodates nesting or creation of levels of abstraction: a higher level heuristics could be devised to control a lower level one. For instance we could have an heuristics to start the iterative process and carries the iteration to a level where it could then hand over to another heuristics to carry on improvement process on the result.

Application of meta-heuristic to optimization problems has been of old and seems to have come to stay (being currently an active area of research) even though in most cases meta-heuristic-based methods lack mathematical proof of convergence, (their performances are usually confirmed or investigated through experimental observations) and provide near-optimal solutions in many cases.

One of the early meta-heuristic devised for optimization problems is the simulated annealing (SA) [3]. It is a probabilistic method that uses a move strategy that imitates the annealing process of a crystalline solid, where a solid is slowly cooled so that when eventually its structure is 'frozen' this happens at a minimum energy configuration. Another is a tabu search [4, 5] that allows a worsening move to be taken where no improving move is available and either keeps a record of some solutions termed as tabu, (being solutions that are forbidden to be revisited in subsequent moves) or define them through a set of rules.

These methods are local search methods that improve on a single solution in each round of their iterations by replacing the single solution with its 'better' neighbour. Local search methods, in general, have the tendency of being trapped in local–optimum areas or on plateaus. Recent research activity is consequently focused on the population-based meta-heuristics resulting into the development of several population-based meta-heuristics, many of which have been used to solve real-life optimization problems: timetabling problem [6], vehicle routing problem [7] data mining [8], rostering [9], Resource scheduling [10], office space allocation problem [11], Circuit Design [12], engineering problems [13, 14]. These meta-heuristics come in different flavours based on what inspired them. Nature-inspired meta-heuristics are inspired by natural processes or phenomena, (laws of nature) such as those from biology (i.e principle of evolution), physics and chemistry (chemical laws). Genetic algorithm [15] and differential evolution [16] are optimization search methods modeled after the principle of evolution of organisms in biology where the best survive in a population. Gravitational search algorithm [17] was fashioned after the law of gravitation in physics and intelligent water drops [18] was proposed to reflect the swarming of water drops flowing with soil along a bed. In [19] is devised the harmony search technique which mimics the improvisation process of a musician.

A swarm-based meta heuristic optimization methods are inspired by the collective behavior of some social-living objects such as birds, fishes. The Particle swarm optimization technique (PSO) invented by [20] was devised to mimic the social behavior of flocking of birds or fish schooling. The Ant colony optimization methods [ACO] [21-23] was devised to imitate the behavior of ants in a colony trying to solve their food problems and Artificial bee colony (ABC) optimization algorithm [24] was presented to mimic the foraging behavior of honey bees in their colony. Symbiotic Organisms search [25] simulates the interactive behavior among organisms. Another meta-heuristic method is recorded in [26] regarding monkeys.

A survey of some of the meta-heuristic algorithms can be found in R.S. Parpinelli and H.S. Lopes [27]. Jorge A. and others [28] presented a review of meta-heuristics algorithms based on animal behaviour and classify them into classes (swarming, flocking, schooling, herd e.t.c) based on their social behavior as applied to traveling saleman problem. Bianchi, Leonora,

M Dorigo and others [29] presented a survey on metaheuristic for stochastic combinational optimization.

The next section presents the hybrid method while the section that follows it presents some the results obtained on applying the method on some benchmark functions.

2. The Hybrid Optimization Method

To be highly effective, optimization search algorithm in general should have two components: the explorative and the exploitative components. The explorative component enables the search algorithm to look beyond the neighbourhood of the current point in the search space for the optimum point and so enables the algorithm to avoid being trapped in a local optimum point. The exploitative component concentrates its search effort on the neighbourhood of the current point to avoid missing the global optimum point that might be hidden in the neighbourhood of the point.

The idea employed in this work is to use the evolution technique (precisely the cross-over and repair operations) together with a swarm based position updating method to produce the next position for each particle.

The evolution part is meant to serve the explorative component of the algorithm and is obtained by applying a cross over operation on two positions $\underline{GB}^k$ and $\underline{u}^k$, as:

$$x_j^{''} = \begin{cases} GB_j^k & if \;\; rand() > r \\ u_j^k & otherwise \end{cases}$$

To obtain $\underline{x}^{''} = (x_1^{''},\ x_2^{''}, \ldots\ldots, x_n^{''})$.

Here $\underline{GB}^k = (GB_1^k, GB_2^k, , \ldots., GB_{dim}^k)$, denotes the global best position (i.e the best position so far encountered during the movement of all the particles, up to the k[th] iteration) and $\underline{u}^k = c_1 * \underline{LB}_j^k + c_2 * \underline{GB}^k$ (i.e $\underline{u}^k$ is a linear combination of the local best position, $\underline{LB}_j^k$= $(LB_{i,1}^k, LB_{i,2}^k, \ldots., LB_{i,dim}^k)$ and the global best position , $\underline{GB}^k$ with weights c_1, c_2). The local best position, $\underline{LB}_j^k$ is the best position so far attained by particle i up to the k[th] iteration. The superscript denotes the iteration number while the subscript denotes the component number of the vector (position) to which it is attached. The dim denotes the dimension of the problem.

The parameter r, is a real number in [0,1] meant to control the diversity of $\underline{x}^{''}$ from its constituents, $\underline{GB}^k$, $\underline{u}^k$. For instance setting r=1 would cause $\underline{x}^{''}$=$\underline{u}^k$ while r=0 would cause it about the same as $\underline{GB}^k$. However r is set to 0.5 for the results presented below. In this experiment c_1, c_2 are each set to 1.

The position $\underline{x}^{''}$ is repaired using the rule:

$$x_j^{'} = \begin{cases} x_j^{''} & if \;\; x_j^* \;\; is \;\; in \; the \; interval \;\; [minx_j, maxx_j] \\ minx_j + rand() * (maxx_j - minx_j) & otherwise \end{cases}$$

To cause the resulting position, $\underline{x}^{'} = (x_1^{'},\ x_2^{'}, \ldots\ldots, x_n^{'})$ lie within the search space. The $minx_j, maxx_j$ denote the lower and upper bounds on the search space along the j[th] dimension and rand() is a random number in the range [0,1].

The exploitative component of the algorithm is computed from the formular,

$$\underline{x}^* = \underline{x}_i^k + rand() * c_0 * (\underline{GB}^k - \underline{x}_i^k)$$

This is a swarm based updating method where each particle ignores its own personal experience but uses the global experience of the swarm (i.e the current global best position, $\underline{GB}^k$) to determine its next move. Geometrically the equation might be considered as expressing the action of a particle taking a step from its current position (or taking a neighbouring position) along the direction ($\underline{GB}^k - \underline{x}_i^k$).

Here $\underline{x_i}^k = (x_{i,1}^k, x_{i,2}^k, \ldots\ldots, x_{i,dim}^k)$ denotes the position of particle i at time k (i.e at k^{th} iteration) while rand() is a random number generator that returns a random number in the range [0,1]. The parameter c_0 is set to 3.5 in this experiment.

The position, $\underline{x_i}^{k+1}$, for the i^{th} particle at time k+1 is then taken as either $\underline{x}'$ or $\underline{x}^*$ depending on the one with better fitness value,

$$\underline{x_i}^{k+1} = \begin{cases} \underline{x}' & if \quad \text{fitValue}(\underline{x}') > fitValue(\underline{x}^*) \\ \underline{x}^* & otherwise \end{cases}$$

Diversification (i.e exploration) is further enhanced, by making a particle too close to the current global best position to take a new position which is generated randomly:

$$\underline{x_i}^{(k+1)} = \begin{cases} \underline{minx} + rand() * (\underline{maxx} - \underline{minx}) & if\ distance(\underline{x_i}^{k+1}, \underline{GB}^k) < \epsilon \\ \underline{x_i}^{k+1} & as\ calculated\ just\ above \end{cases}$$

Where $\underline{minx} = (minx_1, minx_2, \ldots\ldots, minx_{dim})$, $\underline{maxx} = (maxx_1, maxx_2, \ldots\ldots, maxx_{dim})$.

The parameter ϵ is used to control this and is set to $\epsilon = 10^{-10}$ in this experiment.

With the following parameters defined as below:

c_0, c_1, c_2**:** positive constants. In this experiment c_1, c_2 are each set to 1 while c_0 is set to 3.5.

dim: the dimension of the problem.

$distance(\underline{u}, \underline{v})$: geometric distance of $\underline{v}$ from $\underline{u}$

D=[minx1,maxx1]x[minx2,maxx2]x…x[minxdim, maxxdim]: the domain of the problem.

minx =(minx1, minx2, ….., minxdim), maxx =(maxx1, maxx2, ….., maxxdim)

fitValue(z) : the fitness value of its vector argument z.

cRate: a real value in the interval [0,1] controlling the cross-over and was set to 0.5..

ϵ: the minimum distance allowed between a particle and the current global position. A particle whose expected position violates this will become a launcher. A launcher is here defined as a particle that generate its next position randomly. In essence ϵ controls the number of launchers. All particle will become launchers if ϵ is very large while small number or non will be if ϵ is zero or less than it. The value $\epsilon = 10^{-10}$ is used in this experiment.

The above is precisely put in an algorithm as follows:

Initialization step:

(a) INITIALIZE randomly the positions $\underline{x_i}^{(0)}$ of all the particles in the population:

$\underline{x_i}^{(0)} = \underline{minx} + rand() * (\underline{maxx} - \underline{minx})$ for i=1,2…,no Of Particles

(b) Set the global best position $\underline{GBest}^0$ to the particle position with the best fitnesss value

Iterative step:

for k=1,2,…….noOfIterations **do** the following looping

for (i=1 ,….., noOfParticles) **do the following**

{(α)UPDATE xik to obtain xik+1 :

(a) (i) **for** (j=0, 1 … to dim) **do**

if (rand()>cRate) **then** $u_j = GB_j^k$

else $u_j = c_1\ LB_{i,j}^k + c_2\ GB_j^k$

(ii) **If** (u_j is not in the interval [$minx_j$, $maxx_j$]) **then** u_j= $minx_j$ + rand()*($maxx_j$ - $minx_j$)

(b) **for** (j=0, 1 … to dim) **do**

{ (i) $v_j = x_{ij}^k + rand() * c_0 * (GB_j^k - x_{ij}^k)$

(ii) **If** ($v_j$is not in the interval [$minx_j$, $maxx_j$]) **then** v_j= $minx_j$ + rand()*($maxx_j$ - $minx_j$)

}
(c) **If** (fitValue(v) >fitValue(u)) **then** set $\underline{x_i}^{k+1} = \underline{v}$
else set $\underline{x_i}^{k+1} = \underline{u}$; {where $\underline{v} = (v_1, v_2, \ldots, v_{dim})$ and $\underline{u} = (u_1, u_2, \ldots, u_{dim})$}
(d) **if** (distance($\underline{x_i}^{k+1}$, $\underline{GB}^k$) < ϵ) **then** $\underline{x_j}^{(k+1)} = \underline{minx} + rand() * (\underline{maxx} - \underline{minx})$
(β) UPDATE global best position fitness value :
if (fitValue($\underline{GB}^k$) < fitValue($\underline{x_i}^{k+1}$)) **then** $\underline{GB}^{k+1} = \underline{x_i}^{k+1}$
else $\underline{GB}^{k+1} = \underline{GB}^k$
}
Output the current global best position, $\underline{GB}^{noOfIteration}$, and it fitness value, fitValue ($\underline{GB}^{noOfIteration}$).

3. Results and Discussion

Algorithms can be compared along many directions: simplicity (in terms of structure and ease of implementation), performance (in terms of convergence and ability to produce required result) and cost (in time and space requirement) for accomplishing the job. While the proposed algorithm cannot compete in terms of simplicity with such method as particle swarm optimization (PSO) it does favourably, with some evolutionary based algorithms (such as the genetic algorithm (GA), Differential Evolution (DE)) or swarm based algorithm such as the artificial Bee colony (ABC). Its non- consideration of the velocities of the particles (possibly a disadvantage unlike in PSO) makes the need for extra store for its implementation not necessary. However we are more interested here on performance (the ability to produce result) rather than these other features and for this we need functions on which to test the method.

Different functions have different features that tell on how difficult it is to obtain their global optimum. For example:

a. Inter-relationship of dependent variables: a function is said to be separable if it can be written as a sum of functions of just one variable. A non-separable function thus has its dependent variables some how tangled together and this makes its global optimum a bit difficult to obtain than those of the separable type.

b. The function's dimension: obtaining the global optimum of a high dimensioned function is in many cases more difficult than those of lower ones and the computational time it takes may be more.

c. Multi-modal: multi-modal functions are functions with more than one local optimum: Methods that have no means of getting out of a local optimum would usually get trapped in such point and so return a local optimum for a global optimum. The difficulty of handling such problem may be compounded further by the:

1. Number of such local optimum. Functions with large number of local optimum may be a bit more difficult to handle
2. Distribution of the local optimums (i.e whether randomly distributed or not). Those with randomly distributed local optimums may be more difficult to handle

d. Position of the global optima: for instance whether close to some local optimums, in which case a local one may mistakenly be taken for the global one. Global optimum near the boundary of the search space can cause problem to a method deficient in boundary searching.

e. Relative size of the global optimum compared with the whole search space. A very small global optimum relative to the search space size may be a bit more difficult to be noticed.

According to [30], "attempting to design a perfect test set where all the functions are present in order to determine an algorithm is better than another for every function is a fruitless task. That is the reason why when an algorithm is evaluated, we must look for the kind of problems where its performance is good in other to characterize the type of problems for which the algorithm is suitable".

In view of this, we decided using some benchmark functions for this purpose. Many benchmark functions have been devised for testing optimization methods. Some of these are unimodal (U), having no local optimum value apart from the global optimum, others are multimodal (M), having [many] local optimum. Some are separable (S) while others are non-separable (N). Having these classification in mind the following benchmark functions were selected. The functions are labeled F0, F1, to F10 for ease of identification. The minimization

problem is turned into optimization problem by negating the objective function (i.e min { F(x)} is turned into max{-F(x)}).

F0: Rosenbrock's (UN): $f(x_1, x_2, \dots, x_d) = \sum_{i=1}^{d-1}[(1 - {x_i}^2)^2 + 100({x_{i+1}} - {x_i}^2)^2]$. Global Min: 0 at x_i=1 in [-3,3] [d].

F1: De Jong's $f(x_1, x_2, \dots, x_d) = \sum_{i=1}^{d} {x_i}^2$. Global Min:0 at (0,0,…..,0) in [-10,10] [d].

F2: Schwefel (UN) $f(x_1, x_2, .., x_d) = \sum_{i=1}^{d}\left(\sum_{j=1}^{i} x_j\right)^2$. Global Min: 0 at (0,0,..,0) in [-10,10] [d].

F3: Eggerate: $f(x_1, x_2) = (x_1^2 + x_2^2) + 25(\sin^2 x_1 + \sin^2 x_2)$. Global Min: 0 at (0,0,..,0) in [-2π, 2π]^2.

F4:Ackley's (MN) $f(x_1, .., x_d) = 20 + e - 20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} {x_i}^2}\right) - \exp(\frac{1}{d}\sum_{i=1}^{d} cos 2\pi x_i)$

Global Min:0 at point (0,0,…..,0) in [-10,10] [d].

F5: Griewank (MN): $f(x_1, x_2, \dots, x_d) = 1 + \sum_{i-1}^{d} \frac{{x_i}^2}{4000} - \prod_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}})$. Global Min: 0 at (0,0,…,0) in [-10,10] [d].

F6: $f(x_1, x_2, \dots, x_d) = \frac{1}{0.1+1+\sum_{i-1}^{d} \frac{{x_i}^2}{4000} - \prod_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}})}$. Global Min: 10 at (0,0,…,0) in [-10,10] [d].

F7 Rastrigin (MS): $f(x_1, x_2, \dots, x_d) = \sum_{i=1}^{d}({x_i}^2 - 10\, cos 2\pi x_i + 10)$. Global Min: 0 at (1,1,..,1) in [-10,10] [d].

F8 Schwefel(MS): $f(x_1, x_2, \dots, x_d) = 418.9829 * d - \sum_{i=1}^{d} x_i sin\sqrt{|x_i|}$. Global Min: 0 at x_i=420.9867 in [-500,500] [d].

F9 Styblinski-Tang (): $f(x_1, x_2, \dots, x_d) = \frac{1}{2}\sum_{i=1}^{d}({x_i}^4 - 16\,{x_i}^2 + 5x_i\,)$. Optimum. value: 39.165999*d at x_i=-2.903534

F10 Dixon-Price (MS): $f(x_1, x_2, \dots, x_d) = (x_1 - 1)^2 + \sum_{i=2}^{d} i(2{x_i}^2 - x_i - 1)^2$. Global Min:0, in [-10,10] [d]

F11 Zakharov(MS):$f(x_1, \dots, x_d) = \sum_{i=1}^{d} {x_i}^2 + (\sum_{i=1}^{d} 0.5 i x_i)^2 + (\sum_{i=1}^{d} 0.5 i x_i)^4$.Global Min:0 at x_i=0 in [-5,0] [d].

A discussion on the behaviour of some of these functions can be found in [31]. For example, Rosenbrock function (F0) has a global optimum inside a long narrow parabolic shaped flat valley, with its variables strongly dependent on each other and its gradient not pointing towards the optimum.

Below are presented, results obtained on testing the method on these benchmark functions, F0, ..., F11. In all, the average (Ave), average best (Ave. Best) and standard deviation (Std. Dev) of the fitness values were taken over 20 runs (with each run made up of 50000 iterations over the particles unless otherwise stated).The population of the particles (used) is 20. Tables 1 and 2 present the output of the algorithm on these functions for dimensions 10, 20, 30 and 40.

The algorithm seems to behave well even as the dimension increases except on functions F0, F10. For which it gives a poorer result as the dimension of the function increases beyond 20.

The graph in Figure 1 shows the effect of dimension increase of the functions on the performance of the algorithm. It plots the standard-deviation of fitness values against the dimensions of the functions with the number of iterations fixed at 50,000.

Table 1. Best, Average and Standard Deviation of the Fitness Values for dim=10, and 20

Func	Dimension:10	Dimension:20

	Best	Ave	St. Dev	Best	Ave	St.Dev
F0	0.000000	0.000000	0.000000	0.000298	004646	0.002408
F1,F2,F3,F4,F5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
F6	10.000000	10.000000	0.000000	10.000000	10.000000	0.000000
F7,F8	0.000000	0.000000	0.000000	-0.000000	-0.000000	0.000000
F9	352.495605	352.495605	0.000038	744.157592	744.157288	0.000137
F10	0.000000	-0.499999	0.288674	-0.666667	-0.666667	0.000000
F11	0.000000	0.000000	0.000000	-0.000000	-0.000000	0.000000

Table 2. Best, Average and Standard Deviation of the Fitness Values for dim=30, and 40

Func	Dimension :30			Dimension :40		
	Best	Ave	Std. Dev	Best	Ave	Std. Dev
F0	-0.010184	-5.652472	2.702841	-0.053101	-16.517511	9.381538
F1	-0.000000	-0.000000	0.000000	-0.000000	0.000000	0.000000
F2	-0.000000	-0.000000	0.000000	-0.000000	-0.037128	0.161281
F3,F4,F5	-0.000000	-0.000000	0.000000	-0.000000	0.000000	0.000000
F6	10.000000	10.000000	0.000000	10.000000	10.000000	0.000000
F7	-0.000000	-0.000000	0.000000	0.000000	0.000000	0.000000
F8	-0.003906	-0.003906	0.000000	-0.009766	-0.009766	0.000000
F9	1135.81958	1135.81958	0.000217	1527.481934	1527.481689	0.000345
F10	-0.666667	-0.666667	0.000000	-0.666667	-0.666667	0.000000
F11	-0.000000	-0.000000	0.000000	0.000000	0.000000	0.000000

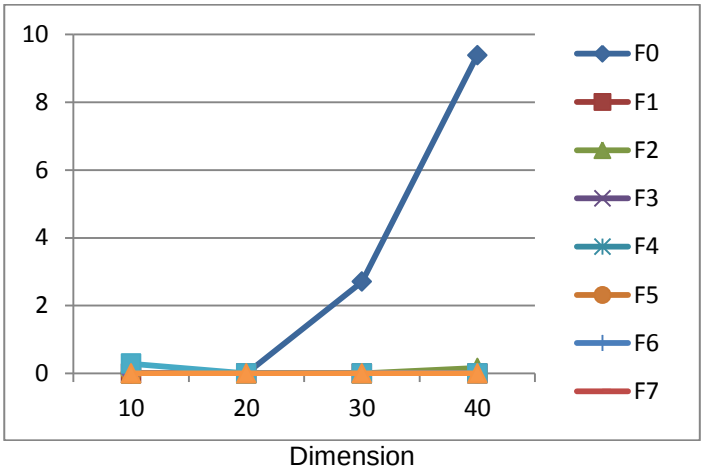

Figure 1. A Plot of Standard Deviation of Best Fitness Values Against the Dimensions of the Functions

The algorithm returns for F0 the value 0.053101 instead of the exact value 0 at dimension 40 with standard deviation 9.381538,thus suggesting a need for more iterations than 50000 used. A trial of 6000 iterations for the function produced average best -0.005736 with standard deviation 8.687126.

For function F10 the algorithm failed to produce the global optimum 0 for dimension greater than 10 but hangs on to the average best 0.6666670 for the higher dimension and with standard deviation 0.

The graph in Figure 2 below presents a plot of the standard-deviation of fitness values against the number of iterations used to obtain them with the functions' dimension fixed at 10. The standard deviation shows how dispersed, the values of the fitness values are, with respect to their mean. It shows the spread of the fitness values. Consequently smaller standard deviation shows smaller variation from the mean so that we might say approximately that the algorithm converging about the mean (when the standard deviation approaches zero as the number of iterations increases).

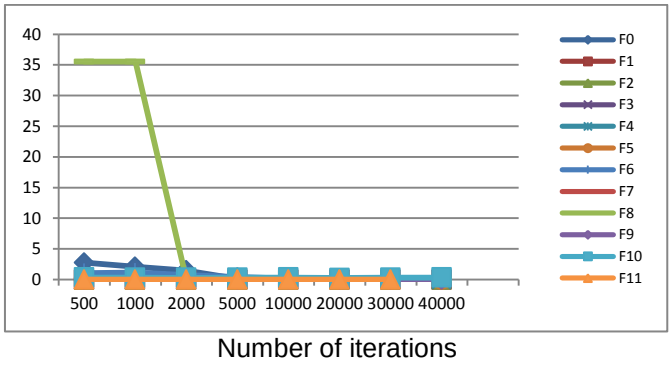

Figure 2. A Plot of Standard Deviation of Best Fitness Values Against Number of Iterations for Functions, Dimension=10.

Apart from F8 (whose convergence rate joins the rest only after 2000 iterations) the others appear to require fewer iterations to produce the solution. F8 is a multi-modal functions whose number of local minimum increases exponentially and this may possibly be the reason.

Table 3 below contains the result when the population size is varied with the dimension fixed at 20. The population size would certainly affect the amount of memory and computational time the algorithm requires to produce the global optimum and so worth considering. The sizes 5, 10 and 15, 20 were tried as the graph in Figure 3 shows. Table 3, 1 contains the result for size 5, 10 and 20]. The graph is a plot of standard deviation of the fitness values against the population. Except for function F0 the population size appears not to affect the algorithm much at least on those functions.

Table 4 below, compares the result of this algorithm, denoted ESH, (standing for Evolution Swarm Hybrid) with those of genetic algorithm (GA), particle swarm optimization (PSO), differential evolution (DE) and artificial bee colony (ABC) as recorded in [30] for functions' dimension 30.

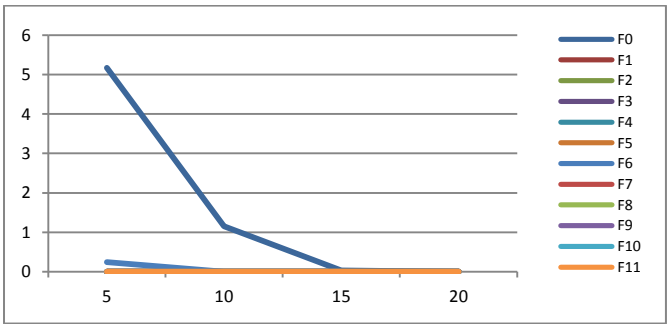

Figure 3. A Plot of Standard Deviation of Best Fitness Values Against Population Size for Dimension= 20

Table 3. Best, Average and Standard Deviation of the Fitness Values for Population=5, and 10 at Dimension: 10

Func	**population:5** Dimension 20			**population:10** Dimension:20		
	Best	Ave	St. Dev	Best	Ave	St.Dev
F0	-0.000116	-9.326571	5.166404	-0.001510	-1.188875	1.154142
F1,F2,F3,F4,F5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
F6	10.000000	9.945153	0.239073	10.000000	10.000000	0.000000
F7,F8	0.000000	0.000000	0.000000	-0.000000	-0.000000	0.000000
F9	744.157471	744.157288	0.000117	744.157592	744.157288	0.000100
F10	-0.666667	-0.666667	0.000000	-0.666667	-0.666667	0.000000
F11	0.000000	0.000000	0.000000	-0.000000	-0.000000	0.000000

Table 4. Comparison of Result of the Algorithm with GA, PSO, DE, ABC As Lifted from [30]. Dimension: 30

Fun	Algorithm	Ave. Best	Std. Dev	Fun	Algorithm	Ave. Best	Std. Dev
F0	**ESH**	0.010184	2.702841	F7	**ESH,ABC**	0	0
	GA	1.96E+05	3.85E+04		GA	52.92259	4.564860
	PSO	15.088617	24.170196		PSO	43.9771369	11.728676
	DE	18.203938	5.036187		DE	11.716728	2.538172
	ABC	0.0887707	0.077390	F8	**ESH**	-0.003906	0.000000
F1	**ESH, PSO, DE, ABC**	0	0		GA	-11593.4	93.254240
	GA	1.11E+03	74.214474		PSO	-6909.1359	457.957783
F2	**ESH,PSO, DE, ABC**	0	0		DE	-10266	521.849292
	GA	7.40E+03	1.14E+03		ABC	-1256.487	0
F4	**ESH,DE,ABC**	0	0	F10	ESH	0.66666667	0
	GA	14.67178	0.178141		GA	1.22E+03	2.66E+02
	PSO	0.16462236	0.493867		PSO	0.66666667	E-8
F5	**ESH,PSO,DE**	0	0		DE	0.66666667	E-9
	GA	0.013355	0.004532		**ABC**	0	0
	ABC	0.0002476	0.000183	F11	**ESH,ABC**	0	0
					GA	10.63346	1.161455
					PSO	0.1739118	0.020808
					DE	0.0014792	0.002958

The dimension of the functions was 30 as in the article. The table shows in bold (letters), those methods best for each of the functions. The algorithm is among those best methods for these functions except for F11 where it surrenders to ABC and follows PSO and DE with average best value 0.66666667.

4. Conclusion

A new meta-heuristic method for optimization problem in a continous space is presented. To determine the position of each particle at each iterative step, the method compares the offspring obtained from a cross over of the instantaneous global best position namely $\underline{GB}^k$ and $\underline{u}= c_1\ LB_j^k\ + c_2\ GB_j^k$ (i.e a position obtained by taking the linear combination of this global best position and the local best position of the particle) with the position $\underline{v} = \underline{x_i}^k + rand() * c_0 * (\underline{GB}^k - \underline{x_i}^k)$, obtained when the particle's current position is incremented by a factor of the deviation of the current position of the particle from the current global position. The better of the two is taken for the particle's next position. The algorithm was tested over some benchmark functions with dimension 10, 20, 30, 40 and the results obtain are presented on the table above. Based on the results the algorithm appears to have a better success rate of reaching the global optimum for some functions (and with fewer number of iterations) than the popular algorithms such as the PSO, GA, DE, ABC.

References

[1] Wolpert DH, Macready WG. No free lunch theorem for optimization. *IEEE Trans. Evol. Comput*. 1997; 1: 67-82.

[2] Voss S, Osman IH, Roucairol C. Meta-Heuristics: Advances and Trends in Local Search Paradigms for optimization. Norwell. MA, USA: Kluwer Academic Publishers. 1999.

[3] Kirkpatrick, S Gelett CD, Vecchi MP. Optimization by simulated annealing Science. 1983; 220: 621-630.

[4] Fred Glover F. Tabu-search Part 1. *ORSA Journal on computing.* 1989; 1(2): 190-206.

[5] Glover F. Tabu-search Part 2. *ORSA Journal on computing.* 1990; 2(1): 4-32.

[6] Manar Hosny, Shameen Fatima. *Survey of genetic algorithms for university timetabling problem*. International conference on future information technology IPCSIT, IACSIT Press. Singapore. 2011; 13.

[7] Noora Ham Abdulmajeed, Masri Ayob. A firework Algorithm for solving capacitated vehicle routing problem. *International Journal of Advancements in computing Technology (IJACT)*. 2014; 6(1).

[8] Sousa T, Silva A, Neves A. Particle Swarm based Data mining algorithms for classification. *Parallel computing*. 2004; 30: 767-783.

[9] Burke EK, Curtois, Post G, Qu R, Veltman B. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operation Research*. 2008; 188: 330-341.
[10] Emergency Resource Scheduling Problem based on improved Particle Swarm Optimization. *TELKOMNIKA Indonesian Journal of Electrical Engineering.* 2014; 12(6): 4609-4616.
[11] Özgü ÚLKER. Office space allocation by using mathematical programming and meta-heuristics. PhD Thesis. University of Nottingham; 2013.
[12] Xuesong Yan, et al. Demonstration of the application of orthogonal Particle Swarm optimization to Circuit design. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11(6): 2926-2932.
[13] Hadi Eskandar, et al. Water cycle: A novel metaheuristic optimization method for solving constrained engineering optimization problems. *Computers and structures.* 2012: 151-166.
[14] Yuxin Sun, Qinghua, Xuesong Yan. An improved constrained engineering optimization design algorithm. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2014; 12(11): 7079-7978
[15] Holland JH. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. University of Michigan Press. 1975.
[16] Storn R, Price KV. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *J. Global Optimization.* 1997: 11(4): 341-359.
[17] Rashedi E, et al. A Gravitational Search Algorithm*. Information Sciences.* 2009; 179: 2232-2248.
[18] Hamed Shah-Hosseini. *Problem solving by intelligent water drops.* Evolutionary Computation. CEC 2007, IEEE Congress. 2007: 3226-3231.
[19] Green ZW, et al. A new heuristic optimization algorithm: Harmony search. *Simulation.* 2001; 76: 60-68.
[20] Kennedy J, Eberhart J. *Particle swarm optimization.* In Proc. IEEE International Conference Neural Networks. Piscataway, NJ. 1995; 4: 1942-1948.
[21] Dorigo M, Gambardella LM. Ant colony System: a cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary computation*. 1997; 1(1): 53-66.
[22] Socha K, Dorigo M. Ant colony optimization for continous domains. *European Journal of operation research*. 2008; 185(3): 1155-1173.
[23] Christian B. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews 2.* 2005: 353-375.
[24] Karaboga, Bahriye Basturk. A powerful and efficient algorithms for numerical function optimization, artificial bee colony (ABC) algorithm. *J. Glob. Optim*. 2007; 39: 459-471.
[25] Cheng Min-Yuang, Prayogo Doddy. Symbiotic Organisms search: A new metaheuristic optimization algorithm. *Computers and Structures*. 2014; 139: 98-112.
[26] K Lenin, B Ravindhranath Reddy, M Suryakalavathi. Modified Monkey optimization Algorithm for solving optimal Reactive Power Dispatch *Problem. Indonesian Journal of Electrical Engineering and informatics (IJEEI).* 2015; 3(2): 55-62.
[27] Parpinelli RS, Lopes HS. New inspirations in swarm intelligence: A survey. *International Journal of Bio- inspired computation*. 2011; 3(1): 1-15.
[28] Jorge A, et al. Meta-Heuristics Algorithms based on the Grouping of animals by Social behavior for the travelling saleman problem. *International Journal of Combinational optimization Problems and Informatics.* 2012; 3(3): 104-123.
[29] Bianchi, Leonora, M Dorigo, et al. A survey on metaheuristic for stochastic combinational optimization. *Natural computing: an international Journal.* 2009; 8(2): 239-289.
[30] Dervis Karaboga, Bahriye Akay. Comparative study of Artificial Bee Colony Algorithm. *Applied Mathematics and computation*. 2009; 214: 108-132.
[31] Dervis Karaboga, Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J. Glob optim*. 2007; 39: 459-471.