Applied Intelligence

A modification to Classical Evolutionary Programming by shifting strategy parameters

--Manuscript Draft--

Manuscript Number:	APIN-1354
Full Title:	A modification to Classical Evolutionary Programming by shifting strategy parameters
Article Type:	Original Submission
Keywords:	Evolutionary programming; Rotation operator; Strategy parameters; Step size, Fixed and adaptive lower bound strategy parameter

Title page

First and Corresponding Author: **Yousef Alipouri**, Ph.D.

Affiliation: Electrical Engineering Department, Iran University of Science and Technology

Email: alipouri_yousef@elec.iust.ac.ir

Address: Faculty of Electrical Engineering, Iran University of Science and Technology, Narmak, Tehran 16844, Iran.

Tel: (+9821)77240492
Fax: (+9821)7454055

Second author: **Javad Poshtan**, Associate Professor, Dr.

Affiliation: Electrical Engineering Department, Iran University of Science and Technology

Email: jposhtan@iust.ac.ir

Address: Faculty of Electrical Engineering, Iran University of Science and Technology, Narmak, Tehran 16844, Iran.

Third author: **Yagub Alipouri**, Ms.c

Affiliation: Department of Civil Engineering, Amirkabir University of Technology, Iran.

Email: yagub.alipouri@aut.ac.ir

Address: AUT university, 424 Hafez Ave, Tehran, Iran, 15875-4413. +98 (21) 64540

A modification to Classical Evolutionary Programming by shifting strategy parameters

Yousef Alipouri[1], Javad Poshtan[1], and Yagub Alipouri[2]

[1] Electrical Engineering Department, Iran University of Science and Technology
[2] Department of Civil Engineering, Amirkabir University of Technology

Abstract

Many methods have been recently suggested for promoting the performance of Evolutionary Programming (EP) in finding optimum point of functions or applications. EP has some shortcomings which slow down its convergence to global minimum, specially for multimodal functions. As it is known, mutation is one of the most important operators in EP which produces new attributes in variables. Mutation must be under control; otherwise, it destroys heritage information. In EP, mutation is implemented by adding strategy parameters to variable vectors of parents in order to produce offspring. When one of the strategy parameters takes on a big value, adding it to the related variable causes abrupt changes in the variable. So, the variable grows with big steps and deviates far from the optimum point whereas some of other variables do not sense considerable changes. If this event continues for some iterations, the variable will go further. This event slows down EP in some iterations. To avoid such an occurrence, this paper introduces a new method that can overcome these disadvantages and enhance the performance of classical evolutionary programming. The paper describes a modification of evolutionary programming by using a rotational method to prevent large and small changes to the strategy parameters. This method adds one function to the mutation operator. This function operates on strategy parameters and changes the sequence of these parameters. Due to the fact that this method does not directly operate on variables, it will not destroy the heritage information of parents. This method was tested on fifty well known test functions used in the literature and was compared with nine well-known EP variants. The Results are robust and demonstrate the efficiency of the technique.

Keywords

Evolutionary programming; Rotation operator; Strategy parameters; Step size, Fixed and adaptive lower bound strategy parameter

I. Introduction

Evolutionary Computation (EC) is a kind of self-organizing and self-adaptive intelligent technique which analogies the process of natural evolution. It is through the reproduction, mutation, selection and competition that the evolution of life is fulfilled [1].

Darwin's principle "survival of the fittest" which captured the popular imagination can be used as a starting point for introducing evolutionary computation.

The theory of natural selection proposes that the plants and animals that exist today are the result of millions of years of adaptation to the demands of the environment. Evolutionary computation techniques abstract these

evolutionary principles into some algorithms which may be used in searching for optimal solutions to a problem. [2].

In the case of evolutionary computation, there are four historical paradigms that have served as the basis for a large extent of the activities in this field: Genetic Algorithms (GA) [3] (Holland, 1975), Genetic Programming (GP) [4] (Koza, 1992), Evolution Strategies (ES) [5] (Recheuberg, 1973), and Evolutionary Programming (EP) [6] (Fogel et al., 1966). The basic differences between the paradigms lie in the nature of the representation schemes, the reproduction operators and selection methods [2]. B¨ack et al. [7] compared these paradigms from empirical and theoretical points of view.

EP is a useful method of optimization when other techniques such as gradient descent or direct analytical discovery are not possible. Combinational and real-valued function optimization, in which the optimization surface or fitness landscape is "rugged", with many locally optimal solutions are well suited for evolutionary programming [8].

Unlike other EAs (like GA) that focused on the crossover operator, the mutation operator is the main operator in EP; consequently, it is the break through point of EP [9]. Since 1970 in which Classical Evolutionary Programming (CEP) was introduced [6], many methods have been proposed for overcoming the disadvantages of this algorithm. Almost all the efforts made in these methods attempted to the enhance mutation operator. The mutation operator controls the diversity of individuals, so enhancing its performance prevents form premature convergence and speeds up searching the whole part of the searching map.

I.I Main EP Variants

Classical Evolutionary Programming uses Gaussian distribution function for updating offspring and strategy parameters (mutation parameters). A study on non-Gaussian mutation in EP was initially carried out in the mid-1990s [10]. A mutation operation based on the Cauchy distribution was proposed as a "Fast Evolutionary Programming (FEP)". FEP converges faster to a better solution than the conventional EP for many multivariate functions.

Unlike the Gaussian probability distribution, the Cauchy probability distribution has an infinite second moment and, as a result, has a much longer tail. Therefore, Cauchy mutated offspring can be quite different from their parents. It was shown analytically that FEP has some advantages over the conventional EP [10].

Lee et al. [11] proposed LEP (evolutionary programming using the Levy probability distribution) in 1999. As Lee et al. indicated, LEP can be considered as the generalization of both Gaussian and Cauchy mutation EP.

Beside CEP and FEP, Narihisa et al. [9,12] proposed EEP (exponential mutation evolutionary programming) with the mutation operator based on a double exponential probability distribution and showed the potential efficiency compatible to the FEP. The eminent merit of EEP is that the size of search step for the solution search processes is controlled by the distribution parameter of double exponential distribution according to the convergence state for given problems [13].

Chellapilla [14] proposed MMO (mean mutation operator) algorithm which defines a new distribution function by combining two distribution functions namely Cauchy and Gaussian.

In 1991, *Fogel et al.* [15] advanced meta_EP random distribution algorithm and in 1993, *Back* introduced another mutation logarithm-normal distribution mutation algorithm, which also improved the performance of EP.

As each mutation operator has its own advantages and disadvantages, the overall performance of the EP can be improved by using different mutation operators simultaneously or by integrating several mutation operators into one algorithm or by adaptively controlled usage of mutation operators. These various properties have let to the idea of mixing some distribution functions. The idea of integrating several mutation strategies into one algorithm within one population is referred to as the mixed mutation strategy [16]. There are different ways for designing a mixed mutation strategy [17,18] , the earliest of which is a linear combination of Gaussian and Cauchy distributions [10]. Improved Fast EP (IFEP) [10] implements Cauchy and Gaussian mutations simultaneously and generates two offspring; the better one is chosen for competition with the parent population during the tournament selection stage. The IFEP variants also include Levy's mutation with various scaling parameters [19]. The idea of IFEP using all the three mutation operators was also investigated in [20]. The weighted mean of Gaussian and Cauchy (MFEP) was introduced by Chellapilla and Fogel [21].

Another proposed way for enhancing the performance of evolutionary programming is the adaptive use of mutation operators. The self-adaptation rules from ESs [22] were incorporated into EP [23]. Self-adaptive EP for continuous parameter optimization problems is now a widely accepted method in evolutionary computation. It should be noted that EP has many similarities to ESs for continuous parameter optimization, although their development proceed independently [18]. In EP, the initialization of the strategy parameter is a critical problem as it depends on given cost functions. To overcome this problem, an Adaptive Evolutionary Programming (AEP) [24] has been recently proposed. AEP is similar to the CEP, except for the initialization and adaptation of the strategy parameter (mutation parameter) values. In AEP, the value of strategy parameter are randomly initialized scaled to the search range of the parameters [25].

Liang et al. [25] proposed the MSAEP (Mixed Mutation Strategy with AEP) algorithm which promotes AEP by mixing the benefits of some mutation functions.

Chellapilla et al. [14] proposed the AMMO (adaptive mean mutation operator) which adapts the shape of distributed function while running the algorithm adaptively.

These are well-known algorithms suggested for enhancing the performance of EP. In this part, some aspects of advantages and disadvantages of evolutionary algorithms, especially evolutionary programming, are discussed. As mentioned, mutation is the most important operator in EP. Most of new methods have focused on the mutation operator for promoting EP and have tried to define a proper mutation distribution function or mixed distribution functions in order to benefits the advantages of these functions simultaneously. But, in this paper mutation function is not changed.

I.II Problem Definition

In EP, mutation is performed by adding strategy parameters to the coordinate of parents. The strategy parameters have main role in deciding the place of offspring. Determining an optimal lower bound for the strategy parameter is essential for the EP algorithm in most applications. A lower bound was used in EP at [15]. Applying a recombination operator for the strategy parameter (mutation parameter η) may reduce the chance of losing the control of step size since the probability of recombining two individuals with very small η values is small [7, 26]. Some previous empirical studies [14, 27] have shown that a properly set lower bound on η can significantly improve the EP's performance.

The optimal setting of the lower bound is problem dependent and cannot be the same throughout the evolution process. Hence, a fixed lower bound for all problems is not appropriate. In [25], two methods were proposed for adjusting the lower bound dynamically during the evolution and they were referred to as Success Rate Based Dynamic Lower Bound Scheme (DLB1) and mutation step size based Dynamic Lower Bound Scheme (DLB2). DLB1 was based on the success rate of the whole population while DLB2 used the median of the mutation step size from all the accepted (successful) offspring as the new lower bound for the next generation.

Setting a near optimal lower bound for η is a difficult task since it is problem dependent. Liang et al. [25] suggested an adaptive method for setting the value for the lower bound. Forcing bounds on strategy parameters has some defects. Therefore, in this paper, after deeply studying the shortcomings of EP, we propose a novel EP algorithm for overcoming this problem without determining the lower bound for strategy parameters. The proposed method of this paper adds a new operator for promoting the performance of EP in regard to speed, accuracy, less recalling cost function and stability of final answers (less standard deviation). Fifty test functions were gathered from several references in order to better compare the results and show the generality of the method's performance. Nine well-known algorithms were selected for comparison. These algorithms are frequently used for comparing new algorithms.

The organization of this paper is as follows: Section II presents the background of the proposed method. Section III introduces the basic idea behind the shifting classical evolutionary programming (SCEP). In section IV, details of the test functions are given and the simulation and comparison results are shown. Section V compares the sensitivity of the method to the change of the parameters. Section VI compares the algorithms as their CPU time and their speed. Section VII compares the algorithms with statistical test and the final section presents the conclusion and summarizes the simulation results.

II. Background

Mathematically, the algorithm of CEP can be described as follows:

1) Generate the initial population of μ individuals and set $k = 1$. Each individual is taken as a pair of real valued vectors, $(x_i, \eta_i), \forall i \in \{1, \cdots, \mu\}$, where x_is are objective variables and η_i s are standard deviations for Gaussian mutations (also known as strategy parameters in self-adaptive evolutionary algorithms).

2) Evaluate the fitness score for each individual $(x_i, \eta_i), \forall i \in \{1, \cdots, \mu\}$ of the population based on the objective function, $f(x_i)$.

3) Each parent $(x_i, \eta_i), i = 1, \ldots, \mu$ creates a single offspring (x_i', η_i') by: for $j = 1, \ldots, n$

$$x_i'(j) = x_i(j) + \eta_i(j) N_j(0,1) \quad (1)$$

$$\eta_i'(j) = \eta_i(j) \exp(\tau' N(0,1) + \tau N_j(0,1)) \quad (2)$$

where $x_i(j)$, $x_i'(j)$, $\eta_i(j)$ and $\eta_i'(j)$ denote the j–th component of the vectors x_i, x_i', η_i and η_i', respectively. $N(0,1)$ denotes a normally distributed one-dimensional random number with mean zero and standard deviation one. $N_j(0,1)$ indicates that the random number generates anew for each value of j. The factors τ and τ' are commonly set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(2\sqrt{n})^{-1}$.

4) Calculate the fitness of each offspring $(x_i', \eta_i'), \forall i \in \{1, \ldots, \mu\}$.

5) Conduct a pairwise comparison over the union of parents (x_i, η_i) and offspring $(x_i', \eta_i'), \forall i \in \{1, \ldots, \mu\}$. For each individual, q opponents are chosen uniformly at random from all the parents and offspring. For each comparison, if the individual's fitness is not smaller than the opponent's, it receives a "win."

6) Select the μ individuals out of (x_i, η_i) and $(x_i', \eta_i'), \forall i \in \{1, \ldots, \mu\}$, that have the most wins to be parents of the next generation.

7) Stop if the halting criterion is satisfied; otherwise, $k = k + 1$ and go to Step 3.

Mutation is performed by adding strategy parameters to the coordinate of parents. The strategy parameters have main role in deciding the place of offspring. Liang et al. [25] showed the necessity of initializing the strategy parameter and step size by theoretical and empirical tests. As explained in the introduction section, determining an optimal lower bound for the strategy parameter is essential for the EP algorithm in most applications. A common feature of EA, especially evolutionary programming (EP) and evolution strategies (ES), in numerical function optimization is a self-adaptive step size. During mutation, each object variable x_i, $i = 1, \ldots, \mu$ is added by a normally distributed random number with mean of 0 and standard deviation of η_i(Equation (1)) which is referred to as the search step size. These η_is are not predefined and fixed. They are self-adaptive and evolve along with x_is. It has been shown by many researchers that self-adaptation helps evolutionary search. However, self-adaptation is not perfect. It has also been shown that self-adaptation may lead quickly to a very small search step size in EP and prevent the search from making progress [27].

The abrupt (very slow) changes in the value of (3) lead to quick (very slow) changes in the value of strategy parameters and variables in Equation (1) and (2).

$$\exp(\tau' N(0,1) + \tau N_j(0,1)) \tag{3}$$

Due to the fact that the mean of the Gaussian distribution function is zero, the value of the random number generated by Gaussian distribution function in Equation (3) is almost near zero. This causes the value of (3) to be mostly in the region of one. However, sometimes the Gaussian distribution function generates big random values (e.g. 10 or bigger) which give a very high value to the exponential function. It increases the strategy parameter undesirably. In other cases, when the negative amount of normal distribution function is so big $(N(0,1) << 0)$ then, (3) takes on a very small value. This pulls down the value of strategy parameter $\eta_i(j)$, so $x_i(j)$ does not sense considerable changes.

Corollary (4) shows that after *k* iteration the variance of normal distribution function can be a very large number which gives a big number to the exponential function. This causes a very big step size (if a positive normal number) or a very small step size (if a negative normal number) in strategy parameters.

$$\eta^{(1)}{}_i(j) = \eta_i(j) \exp(\tau' N_j(0,1) + \tau N_j(0,1))$$
$$\eta^{(2)}{}_i(j) = \eta^{(1)}{}_i(j) \exp(\tau' N_j(0,1) + \tau N_j(0,1))$$
$$\vdots$$
$$\eta^{(k)}{}_i(j) = \eta^{(k-1)}{}_i(j) \exp(\tau' N_j(0,1) + \tau N_j(0,1))$$

$$\Downarrow$$

$$\eta^{(k)}{}_i(j) = \eta_i(j)\prod_{d=1}^{k}\exp(\tau' N_{j,d}(0,1) + \tau\ N_{j,d}(0,1))$$

$$= \eta_i(j)\exp\left[\sum_{d=1}^{k}\left(\tau' N_{j,d}(0,1) + \tau\ N_{j,d}(0,1)\right)\right] = \eta_i(j)\exp\left[\sum_{d=1}^{k}\left(N_{j,d}(0,(\tau')^2) + N_j(0,\tau^2)\right)\right]$$

$$= \eta_i(j)\exp\left[\sum_{d=1}^{k}\left(N_{j,d}(0,(\tau'_d)^2 + \tau_d{}^2)\right)\right] = \eta_i(j)\exp\left(N_{j,d}\left(0,\sum_{d=1}^{k}\left[(\tau')^2 + \tau^2\right]\right)\right)$$

If $n = 10 \quad \Rightarrow \quad \tau = 0.397$, $\tau' = 0.158$

$$\Downarrow$$

$$\eta^{(k)}{}_i(j) = \eta_i(j)\exp\left(N_{j,d}(0, 0.183\times k)\right) \tag{4}$$

So the variance of normal distribution function is 183 after 1000 iterations, which is a very big number and gives a very big (small) value to the exponential function.

Suppose an example with only one individual which produces on offspring in each generation or namely (1+1) EP. Fig. 1 shows the value of strategy the parameter during 10000 samples.

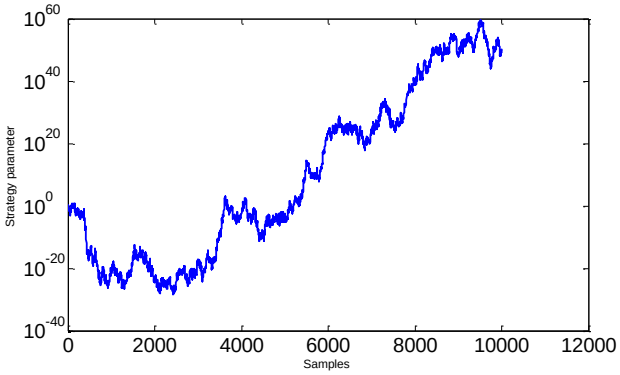

Fig1. The value of strategy parameter during 10000

The ideal situation is to have a large step size for the objective variable at the beginning of the evolutionary process in order to explore different regions of the search space, and have a smaller step size at the later stage for better exploitation within a good region. However, sometimes, self-adaptation may not work. The search step size η_i may quickly reduce to a very small value, thus; it prevents EP from searching for better solutions [27].

For example, if the distance from x_i to the minimum $x^*{}_i$ for the i -th component is $\left|x_i - x_i^*\right| \geq 1$ and the adaptive parameter is $\eta_i \langle 10^{-6}$, the probability of x_i to mutate into a small neighborhood of $x^*{}_i$ is extremely small. If such an individual, x, survives in the population, it propagates poor η values and stagnates the whole search process. Liang et al. [25] demonstrated an example of the way this happens and the reason of its harmfulness for the search.

The problem can be more critical. As is known, strategy parameters in the next generation are a multiple of strategy parameters in the previous one (Equation (2)); so when, at one generation, the amount of strategy parameter takes a very small value, it affects the next generations. Consequently, the relevant variable cannot change during these iterations unless (3) takes a big value and extracts a relevant variable from slow varying. If the amount of normal distribution function is a big positive value, it makes similar problems.

To overcome these disadvantages, one way is to limit the value of strategy parameters between the two threshold values. A lower bound on the search step size is often required for avoiding this problem [25]. This

method (LBEP) was proposed by [27]. But, this method creates two kinds of problems: 1) The first problem is to determine the best value for threshold value and 2) threshold values omit big (small) step sizes which are needed in the first (last) iterations which can slow down (damage the accuracy of) the algorithm.

Another way for solving this problem is to adapt the value of the lower bound as suggested in [25]. The key issue in developing a dynamic lower bound scheme (DLB2) is how to adjust a lower bound based on the information accumulated so far in the evolutionary search. Two schemes were proposed in [25]. One was based on the success rate. Another was based on the mutation's step size. However, this method has some disadvantages. It needs more memory for saving previous step sizes. Also, adaptation of lower bound value needs some more evaluation process which slows down the speed of the algorithm and sophisticate use of these algorithms. Besides, this method has some parameters which must be adapted according to the given cost function.

In this paper, it was not intended to discard the large (small) steps. The objective was to find a more easy way which avoids the repetition of adding big values to only one variable. This prevents the fast growth of one variable, so pulling down its value becomes much easier. How can this idea be implemented? Following section explains the idea of SCEP for implementing this idea.

III. Shifting Classical Evolutionary Programming (SCEP)

The existence of small or big values for the strategy parameter does not make any critical problem. Even it is required for attaining speed and accuracy in the algorithm. The main problem is in the repetition of adding constantly small or big step sizes to the individuals, which stagnates the algorithm. Therefore, in the place of omitting small and big step sizes, the algorithm can avoid the repetition of adding small and big step sizes to individual. For implementing this goal, we propose that the algorithm sometimes rotates strategy parameters. By rotation, advantages and disadvantages of one specie draft to another one. In other words, species sometimes share their gathered information with each other. By implementing this idea, the variable gives its own strategy parameter to its neighbor which avoids repetition of adding very fast (slow) steps to only one variable. By this method each individual can share its big or small step size with other individuals.

It is supposed that a strategy parameter $(\eta(i))$ takes on a big value at iteration i, and cannot get rid of it for k iteration. If this strategy parameter is allowed for adding its value to the variable during this k iteration, the variable will grow continuously, take on a big value and may go far from the global minimum. Besides, the algorithm needs some more (suppose k) iteration for pulling down the value of the variable. The proposed method of this paper, after iterations, shifts the strategy parameter to another variable and avoids the algorithm from adding the big (small) amount to the previous one. This will avoid variables from going further out of the optimal search space. It is clear that our shifting method avoids the CEP algorithm from wasting nearly 2k iterations (k iteration for growth and next k iteration for decrease). Nonetheless, the rotation operator must be controlled; otherwise it will destroy heritage information. To control the shifting operator, the shifting time of the strategy parameters should be decided, which is a difficult decision. It is proposed to define a new parameter like R_C which specifies the percent of total iterations that strategy parameters must be shifted in. Iterations are randomly chosen. The direction of rotation can be selected from the right or left side. How many strategy parameters must be shifted in one rotation? This can be chosen randomly.

This method was originally inspired from big circle double strand DNAs mostly seen in bacteria. One kind of mutation on these DNAs was to change the place of amino acids. This mutation made a robust behavior on the bacteria against new antibiotics [34].

The procedure of SCEP can be coded as follows:

Step *3* of the CEP explained in the introduction section must be changed as follows:

3) Each parent $(x_i, \eta_i), i \in \{1,....,\mu\}$, creates a single offspring (x_i', η_i') by: for $j = 1,...,n$

$$
\begin{aligned}
&a = \text{int}(rand \times n)\\
&if\ (rand > R_c)\\
&\quad \eta_i = shifting(\eta_i, a)\\
&end\\
&x_i'(j) = x_i(j) + \eta_i(j) N_j(0,1)\\
&\eta_i'(j) = \eta_i(j) \exp(\tau' N(0,1) + \tau N_j(0,1))
\end{aligned}
\tag{5}
$$

where,

shifting: the function that shifts strategy parameter *a* times

a: the number of shifting which is chosen randomly

int(.): rounds to near integer value

rand: generates a uniform random number in [0,1]

Fig2 shows how to shifts strategy parameters and produce offspring.

SCEP makes the opportunity for all the species (variables) in order to have an equal chance and a fare share for having big (small) step sizes.

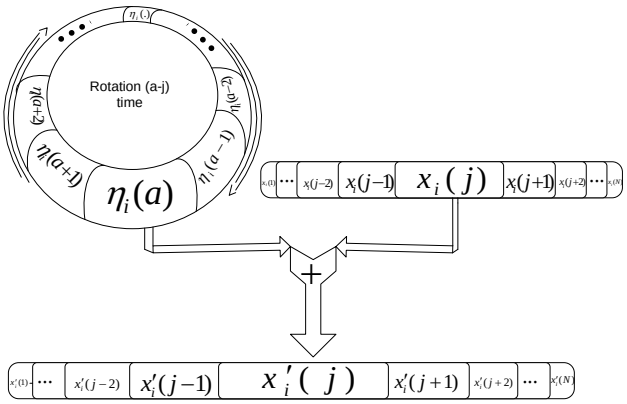

Fig2. Producing the new offspring's variable $x_i'(j)$ by adding the rotated strategy parameter $\eta_i(a)$ to the parent variable $x_i(j)$

Giving a big value to R_C decreases effectiveness of the rotational operator. When it reaches 1, the shifting stops whereas, by decreasing R_C to 0, the rotation operator operates in all the iterations. Now, the question is how to find the best value for R_C? Although, this value must adapt for the problem in use, its optimum region is proposed in the results section. The value of R_C is taken to *0.4* in all simulations. The results showed the capability of this method in increasing the EP's speed and accuracy in finding global minimum.

Supposing a generation with only one member ((1+1) EP), Figs. 3 and 4 compare the effect of shifting on the strategy parameter during 1000 iterations. It can be seen that, although shifting operator does not limited by the value of strategy parameters, the value of the next step of strategy parameter without considering the previous

value can have a small or big step. This step is borrowed from its neighbor individual and is not assigned randomly. This step size helps the individuals to be released from the local minimum and have a very small step size for having acceptable accuracy in finding the global minimum. Suppose that one individual stagnates because of small step size and another individual changes with a very big step size and cannot find optimal regions. Fig. 5 shows that shifting can solve these problems of both individuals with exchanging their step sizes. The following section compares the proposed method with some well-known EP variants.

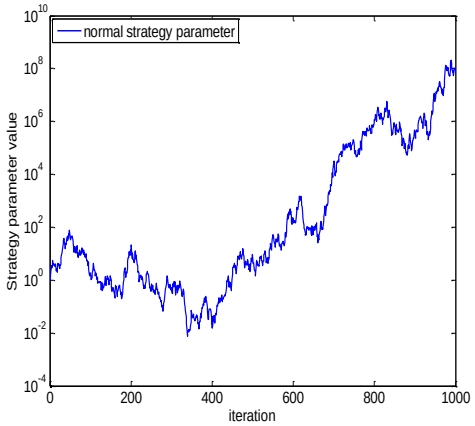

Fig3. 1000 strategy parameters produced by Equation (2)

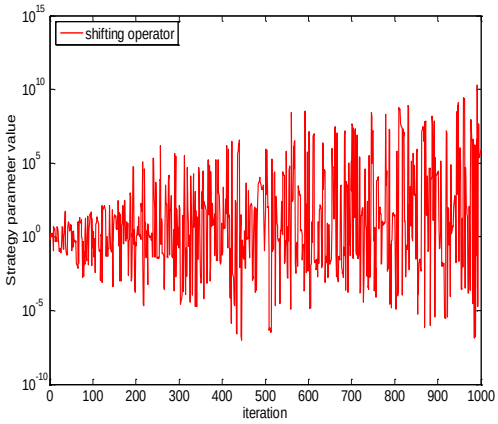

Fig4. 1000 strategy parameters after applying the shifting operator

IV. Simulation Results

New methods are usually compared with respect to cost functions. Some of the cost functions have special properties, so they are selected as benchmark cost functions. These functions can reveal advantages and disadvantages of the new methods. To show the efficiency of the SCEP on different cost functions, 50 benchmark cost functions were selected from [10,22,29]. At the first glance, it seems that these cost functions are more than necessary; however, some of the cost functions that are used in previous papers have symmetrical properties such as the global minimum being located in the center of the search map, so they cannot reveal all of the aspects of the methods. It was tried to select functions that show the properties of the methods more clearly. Twenty seven benchmarks were used in [10] but, for better comparison, we added twenty seven benchmarks whit very interesting properties. These functions are common benchmarks used in many papers. The description of the cost functions are shown in Table II. Some main properties of the functions are explained in this part, but a more detailed description is available in [10,22,28,29].

The cost functions are divided to 3 subgroups of: unimodal- high-dimensional, multimodal-high-dimensional and low-dimensional.

Some properties of these functions are as follows:

Functions *f1 – f29* are high-dimensional problems for which the number of variables can be changed.

Functions *f1 - f11* are unimodals which have only one global minimum which is also their local minimum.

Functions *f11 – f29* are multimodal functions in which the number of local minima increases exponentially with the problem dimension. They appear to be the most difficult class of problems for many optimization algorithms in which CEP has slow convergence on these functions [10].

Functions *f29 – f50* are low-dimensional functions which have only a few local minima but some are hard for minimization by evolutionary algorithms.

For unimodal functions, the convergence rates of methods are more interesting than the final results of optimization as there are other methods which are specifically designed for optimizing the unimodal function. For multimodal functions, the final results are much more important since they reflect an algorithm's ability to escape from poor local optima and locate near-global optimum [10,22].

Gradient based methods can find extermas of low dimensional functions easily, so for these functions, accuracy and repeatability of answers are most important whereas by increasing the number of dimension and local minimums, the ability of gradient based algorithms decreases for finding global minimum.

The ranges of the variables and dimensions of the cost functions were chosen like other references [10,22,29]. For high-dimension functions, the number of variables is *D=30* (if is not limited) and the number of generations is calculated by *100D* or bigger for the functions.

Useful parameters are available in Table I. These parameters are similar to the ones in the literature. The names of fifty assigned cost functions are specified in Table II.

For eliminating the possibility of chance in the results, all of the algorithms were run 20 times; then, the averages of the obtained results were presented in Table III. All of the algorithms were run until a pre-specified generation was reached. The number of the generation is shown in the second column in Table III.

In this table, the best result obtained for each cost function has been highlighted. These results show that, in most of cases, the suggested method can approach the global minimum better than other mentioned methods.

TABLE I
PARAMETERS OF ALGORITHMS

general	Tournament size q	10
	Population size	100
	Range bound of variables	Mentioned in table III
	Number of repetition	50
	Number of generation	Mentioned in tables III
	Initial standard deviation	3
FEP	Parameter *t* for Cauchy distribution function	1
LEP	Parameter α	1.5
EEP	Parameter λ	1
DLB2, LBEP	Initial lower bound	0.0001
AEP	Learning periods *lp*	10
MSEP	Initial probability	1/3
	Parameter λ	1/3
SCEP	Rotation coefficient	0.4

In this research, nine methods of CEP, FEP, LEP, EEP, LBEP, DLB2, MSEP, AEP and SCEP were compared. These were well known methods inside the EP families. For comparison, the values of all the parameters were the same for the nine methods.

Table III compares the accuracy of the methods. The speed of the algorithms will be compared at the CPU TIME section.

The results on Table III show that SCEP was the best one among nine methods in 92 percent of unimodal, 66 percent of multimodal and 76 percent of low-dimensional cost functions.

TABLE II

THE 50 BENCHMARK FUNCTIONS USED IN OUR EXPERIMNTAL STUDY, SECOUND COLUMN INTRODUCES NAME OF FUNCTIONS

Test function	name	Test function	name
$f_1(x)=\sum_{i=1}^{n} x_i^2$	Sphere Model	$f_{26}(x)=-\sum_{j=1}^{30}\frac{1}{c_j+\sum_{i=1}^{n}(x_i-a_{ji})^2}$	Shekel's Foxholes
$f_2(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	Schwefel's problems 2.22	$f_{27}(x)=0.1\sum_{i=1}^{n}\cos(5\pi x_i)-\sum_{i=1}^{n}x_i^2$	Cosine Mixture (CM)
$f_3(x)=\sum_{i=1}^{n}(\sum_{j=1}^{i}x_j)^2$	Schwefel's problems 1.2	$f_{28}(x)=\prod_{i=1}^{n}\left(\sum_{i=1}^{5} i\cos((i+1)x_i+i)\right)$	Shubert
$f_4(x)=\max_i\{\lvert x_i\rvert,1\le i\le n\}$	Schwefel's problems 2.21	$f_{29}(x)=-\left[A\prod_{i=1}^{n}\sin(x_i-z)+\prod_{i=1}^{n}\sin(B(x_i-z))\right]$	Sinuoidal Problem
$f_5(x)=\sum_{i=1}^{n-1}\left[100(x_{i+1}-x_i^2)^2+(x_i-1)^2\right]$	Generalized Rosenbrock function	$f_{30}(x)=[\frac{1}{500}+\sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^6}]$	Shekel's Foxholes function
$f_6(x)=\sum_{i=1}^{n}(\lfloor x_i+0.5\rfloor)^2$	Step function	$f_{31}(x)=\sum_{i=1}^{11}[a_i-\frac{x_1(b_i^2+b_ix_2)}{b_i^2+b_ix_3+x_4}]^2$	Kowalik's Function
$f_7(x)=\sum_{i=1}^{n} ix_i^4+random[0,1]$	Quartic Function with Noise	$f_{32}(x)=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4$	Six-hump Camel-Back Function
$f_8(x)=\sum_{i=1}^{n}2^i.x_i^2$	Hyper-Ellipsoid	$f_{33}(x)=(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6)^2+10(1-\frac{1}{8\pi})\cos x_1+10$	Branin Function
$f_9(x)=\sum_{i=1}^{n}(x_i-1)^2-\sum_{i=1}^{n}x_i.x_{i-1}$	Neumaier #3	$f_{34}(x)=[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]$ $\times[30+(2x_1-3x_2)^2(18-32x_1$ $+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$	Goldstein-Price Function
$f_{10}(x)=\sum_{i=1}^{10}\left[(\ln(x_i-2))^2+(\ln(10-x_i))^2\right]-\left(\prod_{i=1}^{10}x_i\right)^{0.2}$	Paviani Problem	$f_{35}=-\sum_{i=1}^{4}c_i\exp\left[-\sum_{j=1}^{4}a_{ij}(x_j-p_{ij})^2\right]$	Hartman's Family1
$f_{11}(x)=-\exp\left(-0.5\sum_{i=1}^{n}x_i^2\right)$	Exponential Problem	$f_{36}=-\sum_{i=1}^{4}c_i\exp[-\sum_{i=1}^{4}a_{ij}(x_j-p_{ij})^2]$	Hartman's Family2
$f_{12}(x)=\sum_{i=1}^{n}-x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)$	Generalized Schwefel's problem 2.26	$f_{37}(x)=-\sum_{i=1}^{5}[(x-a_i)(x-a_i)^T+c_i]^{-1}$	Shekel's Family1
$f_{13}(x)=\sum_{i=1}^{n}[x_i^2-10\cos(2\pi x_i)+10]$	Generalized Rastrigin's function	$f_{38}(x)=-\sum_{i=1}^{7}[(x-a_i)(x-a_i)^T+c_i]^{-1}$	Shekel's Family2
$f_{14}(x)=-20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2})-\exp(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i)+20+\exp(1)$	Ackley's function	$f_{39}(x)=-\sum_{i=1}^{10}[(x-a_i)(x-a_i)^T+c_i]^{-1}$	Shekel's Family3
$f_{15}(x)=\frac{1}{4000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}})+1$	Generalized Griewank Function	$f_{40}(x)=x_1^2+2x_2^2-0.3\cos(3\pi x_1)\times\cos(4\pi x_2)+0.3$	Bohachevsky 1 (BFI)
$f_{16}(x)=\frac{\pi}{n}\{10\sin^2(\pi y_i)+\sum_{i=1}^{n-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]+(y_n-1)^2\}$ $+\sum_{i=1}^{n-1}u(x_i,10,100,4)$ $y_i=1+\frac{1}{4}(x_i+1)$ $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0 & -a\le x_i\le a\\ k(-x_i-a)^m & x_i<-a\end{cases}$	Generalized Penalized function 1	$f_{41}(x)=\sum_{i=1}^{5}\left(\frac{x_1x_3t_i}{(1+x_1t_i+x_2v_i)}-y_i\right)^2$	Meyer and Roth
$f_{17}(x)=0.1\{\sin^2(3\pi x_1)+\sum_{i=1}^{n-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})]+(x_n-1)[1+\sin^2(2\pi x_n)]\}$ $+\sum_{i=1}^{n}u(x_i,5,100,4)$	Generalized Penalized function 2	$f_{42}(x)=(\exp(x_1)-x_2)^4+100(x_2-x_3)^6+(\tan(x_3-x_4))^4+x_1^8$	Miele and Cantrell
$f_{18}(x)=-\cos(2\pi\lVert x\rVert)+0.1\lVert x\rVert+1,\quad \lVert x\rVert=\sqrt{\sum_{i=0}^{n-1}x_i^2}$	Salomon	$f_{43}(x)=-\sum_{i=1}^{5}a_i exp\left(\frac{-((x_1-b_i)^2+(x_2-c_i)^2)}{d_i^2}\right)$	Multi-Gaussian
$f_{19}(x)=\sum_{k=1}^{n}\sum_{i=1}^{n}\left[\frac{y_{i,k}^2}{4000}-\cos(y_{i,k})+1\right],\quad y_{i,k}=100(x_k-x_i^2)^2+(1-x_j)^2$	Whitley	$f_{44}(x)=\gamma^2+\sum_{k=1}^{4}(\alpha_k^2+\beta_k^2),$ $\alpha_k=(1-x_1x_2)x_3\{\exp[x_5(g_{1k}-g_{3k}x_7\times10^{-3}+g_{5k}x_9\times10^{-3})]-1\}-g_{5k}+g_{4k}x_2,$ $\beta_k=(1-x_1x_2)x_4\{\exp[x_6(g_{1k}-g_{2k}-g_{3k}x_7\times10^{-3}+g_{4k}x_9\times10^{-3})]-1\}-g_{5k}x_1+g_{4k},$ $\gamma=x_1x_3-x_2x_4,$	Price's Transistor Modelling
$f_{20}(x)=\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}\left[\frac{1}{d_{i,j}^2}-\frac{2}{d_{i,j}}\right],\quad d_{i,j}=\left[\sum_{k=0}^{2}(x_{3i+k}-x_{3j+k})^2\right]^3$	Lennard-Jones	$f_{45}(x)=0.5+\frac{(\sin\sqrt{x_1^2+x_2^2})^2-0.5}{(1+0.001(x_1^2+x_2^2))^2}$	Schaffer 1
$f_{21}(x)=\sum_{i=0}^{n-1}\sum_{k=0}^{n-1}\lvert w_{i,k}\rvert,\quad HZ-I=W=(w_{i,k}),\quad I=\begin{bmatrix}1&0&\dots&0\\0&1&\dots&0\\\vdots&&&\\0&0&\dots&1\end{bmatrix},$ $H=(h_{i,k}),\quad h_{i,k}=\frac{1}{i+k+1},\quad Z=(z_{i,k}),\quad z_{i,k}=x_{i+nk}$	Hilbert	$f_{46}(x)=(x_1^2+x_2^2)^{0.25}\left(\sin^2\left(50(x_1^2+x_2^2)^{0.1}\right)+1\right)$	Schaffer 2
$f_{22}(x)=\sum_{k=0}^{n-1}c_k\left\{\exp\left(\frac{-\lVert x-A_k\rVert^2}{\pi}\right)\right\}\cos(\pi\lVert x-A_k\rVert^2),\quad \lVert x-A_k\rVert^2=\sum_{i=0}^{n-1}(x_i-a_{k,i})^2$	Modified Langerman	$f_{47}(x)=100(x_2-x_1^2)^2+(1-x_1)^2+90(x_4-x_3^2)^2+(1-x_3)^2$ $+10.1[(x_2-1)^2+(x_4-1)^2]+19.8(x_2-1)(x_4-1)$	Wood's Function
$f_{23}(x)=-\exp(\frac{-d}{2\pi})\cos(\pi d)\left(1+\frac{0.02h}{d+0.01}\right),\quad d=n.\max((x_i-b_i)^2),\quad h=\sum_{i=0}^{n-1}(x_i-b_i)^2$	Odd Square	$f_{48}(x)=2x_1^2-1.05x_1^4+\frac{1}{6}x_1^6+x_1x_2+x_2^2$	Camel Back 3 Three Hump
$f_{24}(x)=\prod_{i=0}^{n-1}\left(1+(j+1)\sum_{k=}^{m}n\,\mathrm{int}(2^k x_i)2^{-k}\right)$	Katsuura	$f_{49}(x)=-\cos(x_1)\cos(x_2)\exp\left(-(x_1-\pi)^2-(x_2-\pi)^2\right)$	Easom
$f_{25}(x)=\sum_{i=0}^{n-1}\sin(y_i)\left(\sin\left(\frac{(i+1)y_i^2}{\pi}\right)\right)^{2m},$ $y_i=\begin{cases}x_i\cos\left(\frac{\pi}{6}\right)-x_{i+1}\sin\left(\frac{\pi}{6}\right) & if\quad (i+1)\bmod(2)=1\\ x_{i-1}\sin\left(\frac{\pi}{6}\right)+x_i\cos\left(\frac{\pi}{6}\right) & if\quad (i+1)\bmod(2)=0\\ y_{n-1}=x_{n-1} & if\quad i=n-1\end{cases}$	Epistatic Michalewicz	$f_{50}(x)=100(x_2-x_1^2)^2+\left[6.4(x_2-0.5)^2-x_1-0.6\right]^2$	Modified Rosenbrock

TABLE III

COMPARISON AMONG NINE METHODS ON *F1 – F50*. ALL THE RESULTS WERE AVERAGED OVER 20 RUNS. WHERE THE RESULTS INDICATES THE VALUES FOUND IN THE LAST GENERATION, N IS DIMENSION OF FUNCTIONS, F_{MIN} STANDS FOR GLOBAL MINIMUM, LAST COLUM IS RANGE OF VARIABLES

	generation	CEP	FEP	AEP	LEP	DLB2	LBEP	MSEP	EEP	SCEP	N	F_{MIN}	Range bound
F1	1500	62.3	633	55.67	0.34	4.79	0.198	9.56	4.97	6.1E*-6	30	0	$[-100,100]^n$
F2	2000	0.051	18.38	29.69	5.16	1.46	0.29	7.43	1.6E+6	7.2E-4	30	0	$[-10,10]^n$
F3	5000	129	7591	10813	32.39	5398	58.84	119	43833	5.8e-8	30	0	$[-100,100]^n$
F4	5000	0.073	7.94	4.70	0.13	0.071	0.119	0.113	7.65	1.3E-9	30	0	$[-100.100]^n$
F5	20000	76.50	2180	413	53.11	297	94.97	37.19	20087	5.4	30	0	$[-30,30]^n$
F6	1500	35	579	122	4.5	38	0	38.5	58	0	30	0	$[-100,100]^n$
F7	3000	0.74	78.3	2.14	39.60	6.89	0.09	78.39	78.39	0.021	30	0	$[-1.28,1.28]^n$
F8	3000	125	6 E+8	2E+9	6.57	1E+7	4.59	7893	6.94	1.3E+04	30	0	$[-100,100]^n$
F9	30000	3588	28593	29363	24315	-1298	-4320	-3536	-2.51	-4929.95	30	-4930	$[-900,900]^n$
F10	1000	-45.77	-45.77	-42.61	-45.77	-45.77	-45.77	-45.77	-45.77	-45.77	10	-45.77	$[2,10]^n$
F11	3000	-1	-0.98	-0.52	-1	-0.97	-1	-1	-1	-1	30	-1	$[-1,1]^n$
F12	3000	-8552	-12569	-9188	-11542	-8661	-12354	-11868	-11026	-9410	30	-12569	$[-500,500]^n$
F13	5000	46.26	68.21	82.87	27.36	81.09	0.009	9.45	369.6	30.34	30	0	$[-5.12,5.12]^n$
F14	1500	1.92	6.92	13.82	2.23	9.72	0.50	2.05	20.01	0.006	30	0	$[-32,32]^n$
F15	2000	10.83	2.58	80.45	1.98	15.36	0.027	0.36	3.31	0.0098	30	0	$[-600,600]^n$
F16	1500	8.917	5.19	50.15	0.891	36.65	0.117	15.90	2.42	3.8E-6	30	0	$[-50,50]^n$
F17	1500	20.23	149	457	27.84	992	0.15	25.39	86.38	1.25E-4	30	0	$[-50,50]^n$
F18	3000	4.04	3.04	10.99	1.44	6.19	0.95	1.29	3.55	0.29	30	0	$[-100,100]^n$
F19	3000	34.90	2E+8	1E+13	54.21	8.09	8.94	9.07	8E+6	1.8	30	0	$[-100,100]^n$
F20	3000	-7.53	-8.08	-4.05	-6.92	-7.03	-7.69	-8.29	-8.35	-6.03	30	?	$[-2,2]^n$
F21	3000	2.38	3.08	5.19	1.43	19.92	2.55	2.03	2.00	1.97	9	0	$[-2^9,2^9]^n$
F22	1000	-0.38	-0.23	-0.32	-0.47	-0.34	-0.06	-0.63	-0.49	-0.63	10	-0.965	$[0,10]^n$
F23	2000	-0.004	0	0	-0.02	-4E-9	0	-3E-7	-6E-5	-0.64	20	-1.143	$[-5\pi,5\pi]^n$
F24	1000	27223	3.81	3E+13	1	3E+19	1	844	2E+8	1	10	1	$[-1000,1000]^n$
F25	1000	-4.11	-3.86	-2.98	-3.98	-4.07	-3.51	-4.14	-4.29	-4.29	10	-9.660	$[0, \pi]^n$
F26	1000	-1.47	-1.33	-1.08	-1.47	-0.64	-1.85	-1.47	-1.47	-1.92	10	-10.20	$[0,10]^n$
F27	3000	-32.02	-32.69	-25.85	-25.56	-27.71	-31.23	-25.67	-32.49	-22.76	30	?	$[-1,1]^n$
F28	100	-186.7	-186.7	186.6	-186.7	-186.7	-186.7	-186.7	-186.7	-186.7	2	-186.7	$[-10,10]^2$
F29	3000	-3.49	-2.74	-0.80	-1.40	-2.27	-3.09	-2.22	-3.36	-0.30	30	-3.5	$[0,180]^n$
F30	100	2.064	1.12	1.10	1.26	1.98	1.168	1.160	1.246	1.889	2	1	$[-65.53,65.53]^n$
F31	4000	5.9E-4	0.002	0.0041	0.0044	7.8E-4	0.005	5.9E-4	0.003	0.0043	4	0.0003075	$[-5,5]^n$
F32	100	-1.031	-1.031	-1.031	-1.031	-1.031	-1.031	-1.031	-1.031	-1.031	2	-1.031	$[-5,5]^n$
F33	100	0.398	0.398	0.398	0.398	0.398	0.398	0.398	0.398	0.398	2	0.398	$[-5,10]\times[0,15]$
F34	100	3.00	3.00	3.00	3.00	3.00	4.06	3.00	3.00	3.00	2	3	$[-2,2]^n$
F35	100	-3.862	-3.841	-3.862	-3.862	-3.862	-3.830	-3.862	-3.862	-3.862	3	-3.86	$[0,1]^n$
F36	200	-3.23	-2.71	-3.28	-3.22	-3.17	-3.25	-3.23	-3.25	-3.30	6	-3.32	$[0,1]^n$
F37	100	-6.61	-6.21	-7.57	-8.23	-7.99	-6.90	-7.53	-7.17	-9.04	4	10	$[0,10]^n$
F38	100	-8.28	-5.88	-9.73	-8.62	-7.80	-6.28	-6.66	-7.86	-10.17	4	≅10	$[0,10]^n$
F39	100	-9.07	-6.77	-9.78	-7.88	-7.18	-5.24	-8.49	-8.55	-10.25	4	≅10	$[0,10]^n$
F40	100	0.01	0.051	0.026	8E-13	1E-6	0.034	1E-12	3E-12	2.3E-14	2	0	$[-50,50]^2$
F41	100	0.0020	0.0022	0.0021	0.0020	0.0020	0.0024	0.0019	0.0020	0.0019	3	0.0004	$[-10,10]^3$
F42	100	7E-7	3E-5	1.7E-6	1.5E-6	2.6E-6	7.6E-6	3.4E-6	6.8E-6	2.6E-8	4	0	$[-1,1]^4$
F43	100	-1.294	-1.263	-1.295	-1.281	-1.287	-1.270	-1.277	-1.288	-1.287	2	-1.296	$[-2,2]^n$
F44	1000	48.31	48.94	106	24.22	44.55	25.90	31.81	35.88	6.62	9	0	$[-10,10]^9$
F45	100	1.1E-8	3.0E-6	4.7E-6	9.4E-9	3.6E-8	1.5E-9	3.7E-9	8.1E-8	1.8E-7	2	0	$[-100,100]^2$
F46	100	0.001	0.128	0.188	0.001	0.185	0.328	0.002	0.012	6.6E-4	2	0	$[-100,100]^2$
F47	100	1.094	2.658	0.874	1.087	2.403	3.095	0.731	2.410	0.577	4	0	$[-10,10]^4$
F48	100	1.3E-14	4.3E-5	1.7E-14	2E-14	2.4E-7	1.2E-11	1.3E-14	4.7E-15	2.3E-16	2	0	$[-5,5]^2$
F49	100	-1	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	$[-10,10]^2$
F50	100	0.002	0.005	7E-4	0.001	0.002	0.004	0.002	0.002	0.001	2	0	$[-5,5]^2$

*Notation E-a is meaning 10^{-a}.

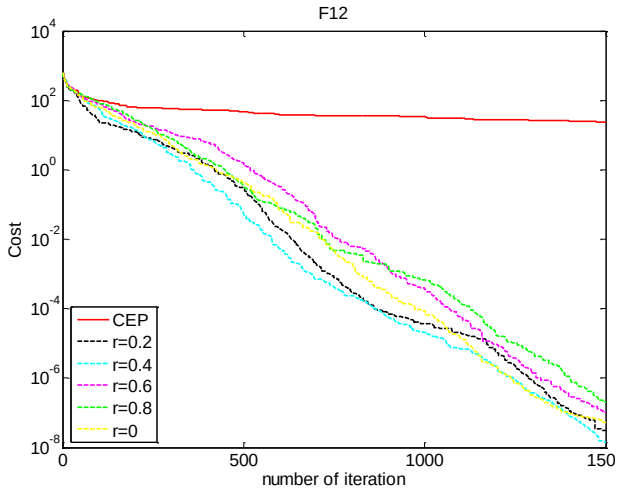

Fig8. Plot of the minimum cost as a function of the generation for *f12* with various values of rotational coefficient (R_C)

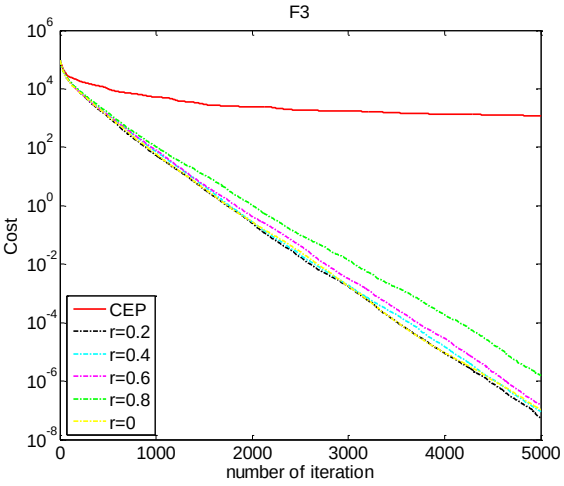

Fig5. Plot of the minimum cost as a function of the generation for *f3* with various values of rotational coefficient (R_C)

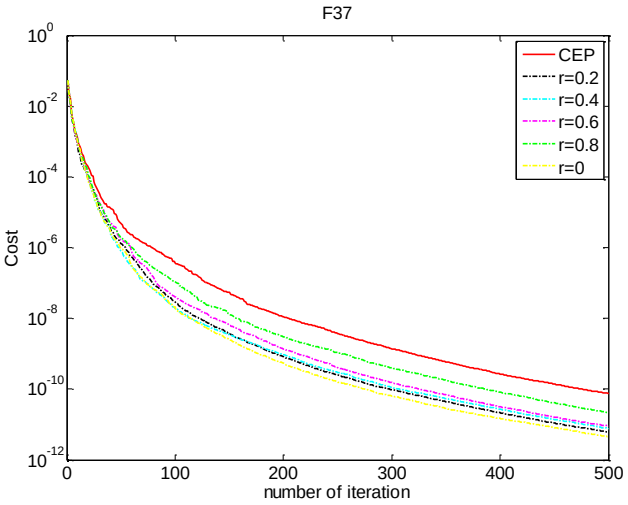

Fig9. Plot of the minimum cost as a function of the generation for *f37* with various values of rotational coefficient (R_C)

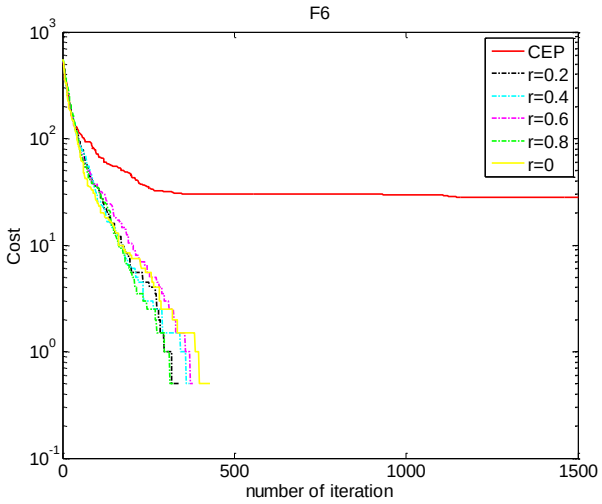

Fig6. Plot of the minimum cost as a function of the generation for *f6* with various values of rotational coefficient (R_C)

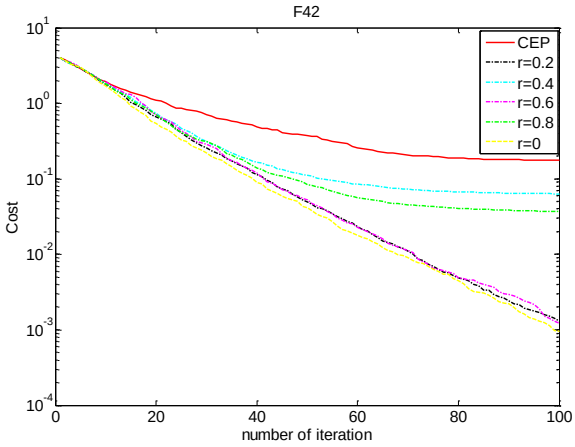

Fig10. Plot of the minimum cost as a function of the generation for *f42* with various values of rotational coefficient (R_C)

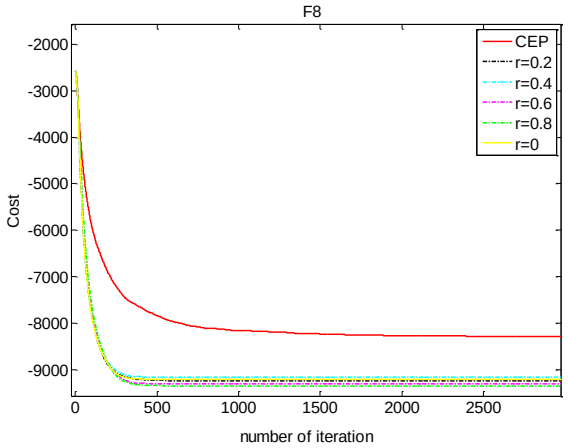

Fig7. Plot of the minimum cost as a function of the generation for *f8* with various values of rotational coefficient (R_C)

As explained, SCEP has one control parameter named R_C. R_C=*0.4* was supposed for simulation results. To find the optimum value for R_C, some experiments were performed. Figs. *5-10* show results of these experiments. These results clarify that the optimum value is in (*0.2,0.4)* region. This test was performed on most of the fifty functions but only six of them were included in this paper. This test clarified the optimum value among five values: R_C=*0, 0.2, 0.4, 0.6, 0.8* and CEP (R_C=*1*). For more reliable results, the algorithms were run for 100 times and the mean values were shown in Figs. *5 - 10*. Figs. *5 - 10* show that the functions are slightly sensitive to R_C. Even by dedicating a big value for R_C (say *0.9*), the responses were improved considerably and by decreasing the value of R_C, the results became more acceptable (remember that big values of R_C cause less rotation on strategy parameters).

V. Sensitivity Analysis of Variation in the Parameters

Some methods are sensitive to the number of populations and variables. When the number of variables or populations increases, the performance of these methods alter. In order to compare methods from these aspects two tables were assigned.

Table IV compares the performance of methods by increasing the number of the population to 20, 50, *100, 200, 500* and *1000* with the same tournament parameter *q=10* and 3000 iterations.

The cost function selected for this comparison is Ackley's function (*5*) named *F14* in table II.

Ackley's Problem (ACK) (Storn and Price, 1997)

$$f(x) = -20\exp(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}) - \exp(\frac{1}{N}\sum_{i=1}^{N}\cos 2\pi x_i) + 20 + e \qquad (6)$$

$$-32 \le x_i \le 32, i \in \{1,....,N\} \qquad \min(f) = f(0,\ldots,0) = 0$$

One of the most commonly cited multi-modal test functions is Ackley's function. This function has one global minimum with an unknown number of local minima [30].

Table IV shows that SCEP algorithm is sensitive to the population number and the best value for the population is about 100.

Table V compares the methods with the number of variables 5, *15, 30, 50, 100, 200*. Cost function is Ackley again. The first 6 rows in Table V compare the algorithms with the fixed number of generation whereas, in last 6 rows, the numbers of generations are calculated with 100D equation. This table shows that SCEP lost its performance when the number of variables increased so much. Another observation which can be derived from Table V is that algorithms are not so sensitive to the number of generation and the equation 100D is enough for reaching around the global minimum.

The last two tables indicate that the variants of EP are not very sensitive to the changes in population but they are very sensitive to the changes in dimensions of the Ackley function. This is SCEP's big shortcoming. As a general conclusion, a comparison must be performed on more cost functions.

TABLE IV
RESULTS OF DIFFERENT ALGORITHMS FOR THE SPECIFIED NUMBER OF INDIVIDUALS TESTED ON THE ACKLEY'S FUNCTION

pop	CEP	FEP	AEP	LEP	DLB2	LBEP	MSEP	EEP	SCEP
20	12.89	5.26	14.18	1.22	13.73	15.80	2.65	9.65	**0.619**
50	4.29	1.05	11.22	0.001	13.37	17.02	0.07	2.80	**5.04E-4**
100	3.72	2.45	12.81	0.01	12.5	16.51	0.03	1.96	**8.3E-6**
200	2.74	1.59	12.54	0.16	9.23	16.32	0.45	3.44	**5.9E-4**
500	2.73	2.57	14.27	2.42	13.21	15.28	2.43	2.51	**0.22**
1000	4.56	6.12	13.31	4.78	12.61	12.89	6.34	6.38	**1.90**

TABLE V
SENSETIVITY RESULTS OF DIFFERENT ALGORITHMS FOR THE SPECIFIED AND VARAIABLE NUMBER OF COST FUNCTION TESTED ON THE ACKLEY'S FUNCTION

Variable	CEP	FEP	AEP	LEP	DLB2	LBEP	MSEP	EEP	SCEP	Generation
5	2.6E-15	0.002	0.311	2.6E-15	5.81	4.9E-5	8.8E-16	1.2E-13	8.8E-16	
15	0.82	0.97	5.01	6.4E-14	7.33	12.95	1.3E-14	6.9E-5	4.4E-15	
30	3.72	2.45	12.81	0.01	12.5	16.51	0.03	1.96	8.3E-6	3000
50	5.93	2.53	14.68	6.37	11.74	18.19	4.75	3.32	1.59	
100	13.75	6.25	16.59	14.27	17.42	19.19	14.07	12.25	10.54	
200	17.52	15.29	18.42	17.55	18.99	19.83	17.88	17.11	15.62	
Variable	CEP	FEP	AEP	LEP	DLB2	LBEP	MSEP	EEP	SCEP	Generation
5	0.164	0.181	0.623	1E-10	2.49	5.025	1.9E-11	3.3E-11	6.5E-15	500
15	0.329	1.866	5.030	0.268	5.424	11.873	3.17E-6	0.423	3.7E-11	1500
30	3.72	2.45	12.81	0.01	12.5	16.51	0.03	1.96	8.3E-6	3000
50	4.95	2.99	14.86	1.60	16.52	17.51	2.01	1.86	0.25	5000
100	12.85	5.68	17.53	11.05	15.76	18.92	12.40	5.85	7.39	10000
200	16.06	7.29	18.93	14.48	17.44	19.53	15.60	13.23	13.48	20000

VI. CPU TIME

CPU time is one of the important factors that clear the value of methods. Some methods are very capable of reaching the goal whereas they are time consuming inasmuch as they cannot be used in real world applications. CPU time is more important in real time applications in which the algorithm must reach its goal in fractional time as fast as possible. Considering this fact, an attempt was made to compare the methods in the aspect of time consumption. All the programs in this study were simulated in the MATLAB R2009b environment with CPU 2.2 GHz, Intel core 2 Duo processor T6600. In this environment, an iteration of CEPs running with 100 individuals and a cost function with 30 variables needs 0.0138119 seconds. SCEP needs more times, because, SCEP must calculate more operators; however, this time is not remarkable. One iteration of running SCEP with the same situation as CEP needs 0.014788723seconds. The difference is about 1 millisecond which is %7 percent of the CEP time consumption. This time will be 3 seconds in 3000 iteration of running CEP for finding the minimum of a cost function with 30 variables. This amount is not remarkable and can be ignored. Meanwhile, this fact that SCEP needs less iteration than CEP to reach the global minimum shows that SCEP's speed is more than CEP. Table VI compares the algorithms in aspect of their one iteration CPU time (units are in millisecond) while applying on function f14 with 30 variables.

TABLA VI
COMPARING CPU TIME OF RUNNING METHODS FOR ONE ITERATION

CEP	FEP	AEP	LEP	DLB2	LBEP	MSEP	EEP	SCEP
13.81	15.79	15.80	16.12	18.93	14.06	15.05	14.43	14.78

Higher CPU time can be fulfilled by higher speed computers but for fulfilling accuracy, the algorithm must be changed. As explained in the previous section, the speed of algorithms is more critical in unimodal- high dimensional but in multimodal-high dimensional ones, the accuracy in finding the global minimum is more important. In multimodal functions, SCEP can reach minimums far from CEP in less iteration, so, it can be considered as having more accuracy, more speed and less CPU time. Notice that in most applications, more than 90 percent of time was wasted on calculating the cost of individuals, especially in real world applications; so, it is more important than the running time of algorithm.

Comparing the Speed of the Methods

There exist two different approaches for collecting data and making measurements from experiments, as schematically depicted in Fig. 11.

- Fixed-cost scenario (vertical cuts). Fixing a number of function evaluations and measuring the function values (costs) reached for this given number of function evaluations. Fixing search costs can be pictured as drawing a vertical line on the convergence graphs (see Fig. 11 where the line is depicted in blue).
- Fixed-target scenario (horizontal cuts). Fixing a target function value (i.e. drawing a horizontal line in the convergence graphs, see Fig. 11 where the line is depicted in green) and measuring the number of function evaluations needed for reaching this target function value.

It is often argued that the fixed-cost approach is close to what is needed for real word applications where the total number of function evaluations is limited. Sometimes in real word applications, the aim is to find a specific cost; so, the fastest algorithm is required to achieve this cost. Fixed-target scenario compares algorithms from this term.

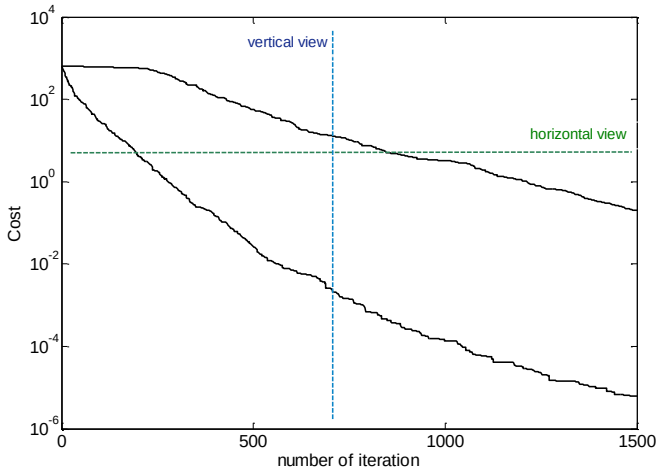

Fig 11. Illustration of fixed-cost and fixed-target scenario

Table VII compares algorithm in terms of fixed-target scenario. As it can be observed in this table, the SCEP outperforms other algorithms. This shows the speed of the SCEP algorithm in reaching lower costs.

However Figs. 12-17 compare the algorithm in term of the first scenario. In these figures, the ability of methods in minimizing cost function from both scenarios is shown. It is obvious that the proposed method has acceptable speed among other introduced methods. It must be considered that, by multiplying the iteration value to CPU time cited in Table VI, running time of algorithms will be evaluated. Thus, the speed of the algorithms can determine the value of the running time, too.

TABLE VII
SPEED COMPARISON OF THE METHODS FROM THE ASPECT OF FIXED-TARGET SCENARIO

Function	Cost	CEP	FEP	AEP	LEP	DLB2	LBEP	MSEP	EEP	SCEP	NI
F1	1000	509	340	__*	532	612	629	601	531	**264**	5000
	10	820	666	__	816	705	904	1156	1827	**517**	
	0.1	1009	1244	__	1279	1192	1079	1649	1595	**679**	
	0.0001	2238	__	__	3319	__	1695	2932	3975	**1207**	
	0.000001	3172	__	__	3777	__	2938	3505	__	**1614**	
F3	1000	1	1	1	1	1	1	1	1	**1**	5000
	100	514	810	__	564	206	802	618	369	**207**	
	10	1390	__	__	1539	4056	1816	2543	1932	**1354**	
	1	4407	__	__	3846	__	2631	3570	__	**1955**	
	0.1	5000	__	__	__	__	3350	__	__	**2523**	
F14	15	110	110	**48**	84	__	__	86	85	50	1500
	10	410	432	__	198	__	__	317	194	**150**	
	7	464	1018	__	330	__	__	629	396	**239**	
	5	1474	1154	__	477	__	__	939	618	**389**	
	1	__	__	__	1471	__	__	__	__	**851**	
F16	15	1165	250	__	393	__	__	461	332	**189**	1500
	10	__	296	__	441	__	__	566	426	**282**	
	7	__	331	__	578	__	__	655	441	**306**	
	5	__	386	__	637	__	__	728	489	**329**	
	1	__	613	__	**960**	__	__	1181	1207	637	
	0.1	__	1306	__	1478	__	__	__	__	**902**	
F39	-1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	100
	-2	6	8	**5**	6	7	14	7	8	7	
	-5	30	54	**17**	37	43	58	29	40	20	
	-7	37	63	26	42	54	63	37	51	**26**	
	-10	51	77	59	54	66	76	52	63	**41**	
F42	0.1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	100
	0.01	**2**	3	**2**	**2**	**2**	3	**2**	**2**	3	
	0.001	11	16	**5**	10	8	23	8	10	9	
	0.00001	46	__	**25**	85	29	__	63	38	36	
	1E-6	__	__	__	__	67	__	__	90	**59**	

* notation __ shows that algorithm couldn't reach the correspond cost

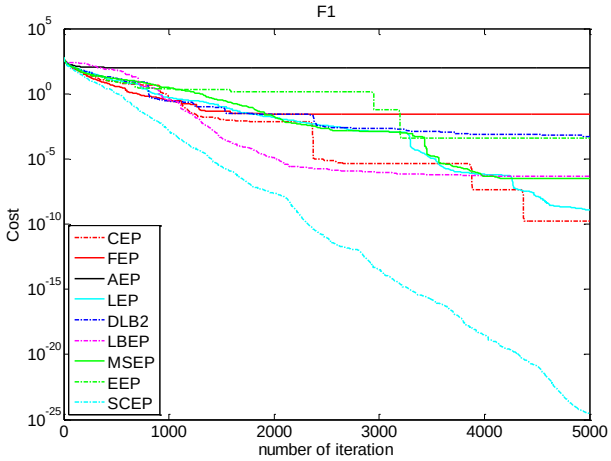

Fig12. Plot of the minimum cost as a function of the generation for *f1*

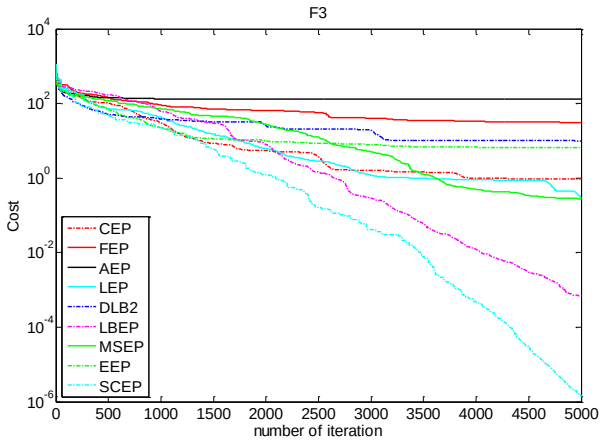

Fig13. Plot of the minimum cost as a function of the generation for *f3*

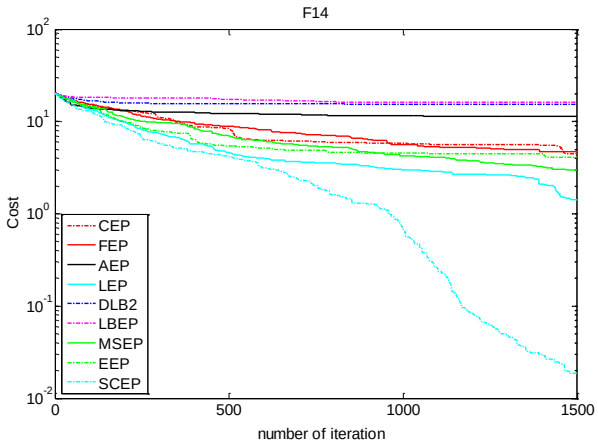

Fig14. Plot of the minimum cost as a function of the generation for *f14*

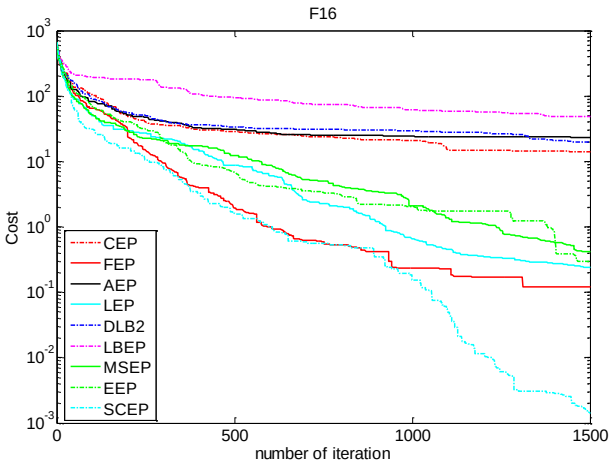

Fig15. Plot of the minimum cost as a function of the generation for *f16*

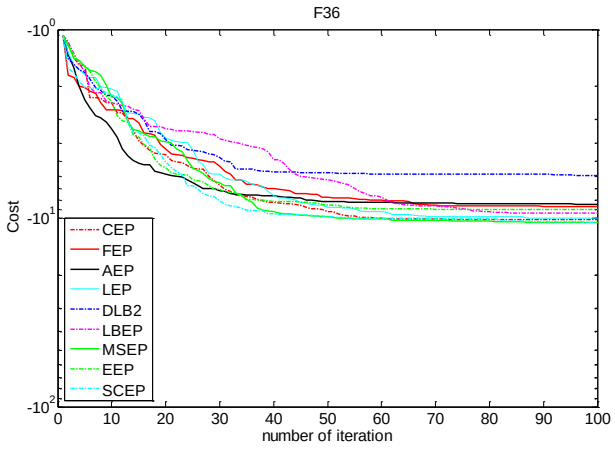

Fig16. Plot of the minimum cost as a function of the generation for *f38*

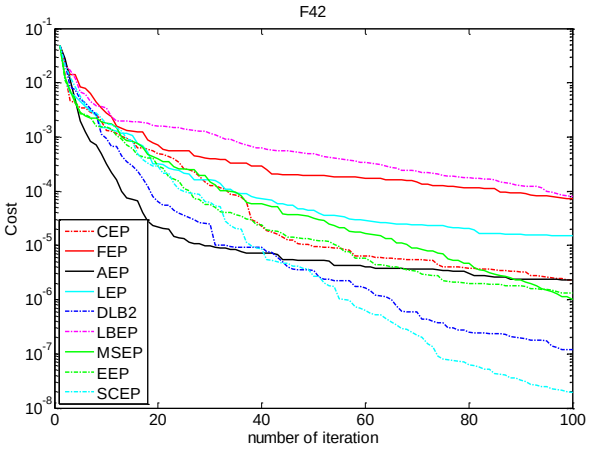

Fig17. Plot of the minimum cost as a function of the generation for *f42*

VII. Statistical Test

In recent years, the use of statistical tests to improve the evaluation process of the performance of a new method has become a widespread technique in computational intelligence. Usually, they are employed inside the framework of any experimental analysis to decide when one algorithm is considered better than another.

This task, which may not be trivial, has become necessary to confirm whether a new proposed method offers a significant improvement, or not, over the existing methods for a given problem.

One of the most frequent situations where the use of statistical procedures is requested is in the joint analysis of the results achieved by various algorithms. The groups of differences between these methods usually associated with the problems met in the experimental study.

In this Section, it is assigned a procedure to estimate the differences between several algorithms: the Contrast Estimation of medians. This method is very recommendable if we assume that the global performance is reflected by the magnitudes of the differences among the performances of the algorithms [31].

VII.I Contrast Estimation of median

Contrast Estimation based on medians [32,33] can be used to estimate the difference between the performance of two algorithms. It assumes that the expected differences between performances of algorithms are the same across problems. Therefore, the performance of algorithms is reflected by the magnitudes of the differences between them in each domain.

The interest of this test lies in estimating the contrast between medians of samples of results considering all pairwise comparisons. The test obtains a quantitative difference computed through medians between two algorithms over multiple problems, proceeding as follows [31].

1. For every pair of k algorithms in the experiment, compute the difference between the performances of the two algorithms in each of the n problems. That is, compute the differences

   $$D_{i(u,v)} = x_{iu} - x_{iv} \tag{7}$$

   where $i = 1,\ldots,n$; $u = 1,\ldots,k$; $v = 1,\ldots,k$. (Consider only performance pairs where $u < v$.)
2. Find the median of each set of differences (Z_{uv}, which can be regarded as the unadjusted estimator of the medians of the algorithms u and v, $M_u - M_v$). Since $Z_{uv} = Z_{vu}$, it is only required to compute Z_{uv} in those cases where $u < v$. Also, note that $Z_{uu} = 0$.
3. Compute the mean of each set of unadjusted medians having the same first subscript, m_u:

   $$m_u = \frac{\sum_{j=1}^{k} Z_{uj}}{k}, \quad u = 1,\ldots,k. \tag{8}$$
4. The estimator of $M_u - M_v$ is $m_u - m_v$, where u and v range from *1* through k. For example, the difference between M_1 and M_2 is estimated by $m_1 - m_2$.

These estimators can be understood as an advanced global performance measure. It is especially useful to estimate by how far an algorithm outperforms another one [31].

In our experimental analysis, we can compute the set of estimators of medians directly from the average error results. Table VIII shows the estimations computed for each algorithm. Focusing our attention in the rows of the table, we may highlight the performance of SCEP (all its related estimators are negative; that is, it achieves very low error rates considering median estimators); on the other hand, AEP and FEP achieve higher error rates in our experimental study.

TABLE VIII

CONTRAT ESTIMATION RESULTS. THE ESTIMATORS HIGHLIGHT SCEP AS THE BEST PERFORMING ALGORITHM.

	CEP	FEP	AEP	LEP	DLB2	LBEP	MSEP	EEP	SCEP
CEP	0	- 0.155	- 0.476	0.041	- 0.100	0.113	0.017	- 0.04	0.227
FEP	0.155	0	- 0.320	0.197	0.055	0.269	0.173	0.110	0.383
AEP	0.476	0.320	0	0.517	0.375	0.59	0.493	0.431	0.704
LEP	- 0.041	- 0.197	- 0.517	0	- 0.142	0.072	- 0.024	- 0.086	0.186
DLB2	0.100	- 0.053	- 0.375	0.142	0	0.214	0.117	0.061	0.328
LBEP	- 0.113	- 0.269	- 0.59	- 0.072	- 0.214	0	- 0.096	- 0.158	0.144
MSEP	- 0.017	- 0.173	- 0.493	0.024	- 0.117	0.096	0	- 0.062	0.210
EEP	0.045	- 0.110	- 0.431	0.086	- 0.061	0.158	0.062	0	0.272
SCEP	- 0.227	- 0.383	- 0.704	- 0.186	- 0.328	- 0.144	- 0.210	- 0.272	0

VIII. CONCLUSION

A new method, Shifted Classical Evolutionary Programming (SCEP), was presented. This method was applied to the classical evolutionary programming. Nine methods, CEP, FEP, LEP, EEP, LBEP, DLB2, MSEP, AEP and SCEP, were compared using fifty cost functions. These cost functions are divided into three groups of unimodal-high dimensional, multimodal- high dimensional and low dimensional. Simulation results showed that using SCEP can speed up the convergence of the CEP in all three groups. SCEP performed best on 92 percent of unimodal, 66 percent of multimodal and 76 percent of low dimensions among fifty cost functions. These revealed the capability of the proposed method in dealing with various kinds of functions. Sensitivity and speed of SCEP was compared with those of other methods. Results indicate that the variants of EP are not very sensitive to the changes in population but they are very sensitive to the changes in dimensions of the Ackley function. In multimodal functions, SCEP can reach minimums far from CEP in less iteration, so, it can be considered as having more accuracy, more speed and less CPU time. SCEP is compared pairwise with introduced algorithms by Contrast Estimation of Median method. This statistical test shows SCEP can outperform other introduced methods.

Different approaches have been proposed to problem optimization, such as the human evolutionary model approach, particle swarm optimization, differential evolution and evolution strategy with covariance matrix adaptation amongst others. Their effectiveness has been extensively studied. In future, further performance comparison between the proposed approach of this article and those of other approaches using benchmark functions will be made. Shifting method was capable of being adapted with other EAs, especially with other variants of EP like FEP, so the future prospect of the approaches proposed here was to test this method on other algorithms.

IX. References

[1] Yang S., Jiao L., (2003) "The Quantum Evolutionary Programming", Proceedings of the Fifth International Conference on Computational Intelligence and Multimedia Applications (ICCIMA'03).

[2] Sivanandam S. N., Deepa S .N., (2008), Introduction to Genetic Algorithms, Springer.

[3] Holland J.H., (1975), Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Harbor.

[4] Koza J., (1992), Genetic Programming: on the Programming of Computers by Means of Natural Selection, MIT Press.

[5] Rechenberg I., (1973), Evolutionsstrategie: Optimierung technischer Systeme nach PrinzISien der biologischen Evolution. Frommann-Holzboog, Stuttgart.

[6] Fogel L.J., (1966), Artificial Intelligence through Simulated Evolution, Wiley, New York.

[7] B¨ack T., Schwefel H. P., (1993) "An overview of evolutionary algorithms for parameter optimization," Evolutionary Computation, vol. 1, no. 1, pp. 1–23.

[8] Ghosh A., Dehuri S., (2004) "Evolutionary Algorithms for Multi-Criterion optimization: A Survey", International journal of computing & Information Sciences, Vol. 2, No, 1.

[9] Narihisa H., Kohmoto K., Taniguchi T., Ohta M., Katayama K., (2006), "Evolutionary Programming With Only Using Exponential Mutation", IEEE Congress on Evolutionary Computations Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada July 16-21.

[10] Yao X., Liu Y., Lin G., (1999), "Evolutionary Programming Made Faster", IEEE Trans. on Evolutionary Computation, vol. 3,no. 2.

[11] Lee C.Y., Song Y., (1999) "Evolutionary Programming using the Levy probability Distribution", Proc. of Genetic and Evolutionary Computation Conference (GECCO'99), Morgan Kaufman, pp.886-893.

[12] Narihisa H., Kohmoto K., Katayama K., (2002) "Evolutionary Programming with Double Exponential Probability Distribution", Proc. of The Second International Association of Science and Technology for Development (IASTED) International Conference on Artificial Intelligence and Applications (AIA2002), pp.358-363.

[13] Kohmoto K., Narihisa H., Katayama K., (2002), "Evolutionary Programming Using Exponential Mutation", Proc. of the 6th World Multiconference on Systematics, Cybernetics and Informatics, vol.11, Computer Science 2, July 14-18, USA, pp.405-410.

[14] Chellapilla K., (1998), "Combining Mutation Operators in Evolutionary Programming", IEEE Tran. on Evolutionary Computation, vol. 2, no. 3,.

[15] Fogel D., Fogel L., Atmar J., (1991), "Meta-evolutionary programming," in Proc. of the 25th Conference on Signals, Systems and Computers, edited by R. Chen, Maple Press: San Jose, CA, pp. 540–545.

[16] Weibull J.W., (1995), Evolutionary Game Theory, MIT Press, Cambridge, MA.

[17] Yao X., Liu Y., (1998) "Scaling up evolutionary programming algorithms, Evolutionary Programming" VII: Proc. of the Seventh Annual Conference on Evolutionary Programming (EP98), Lecture Notes in Computer Science, 1447, Springer-Verlag, Berlin, pp. 103–112.

[18] Dong H., He J., Huang H., Hou W., (2007) "Evolutionary programming using mixed mutation strategy", Information Sciences 177 (1), 312–327.

[19] Lee C.Y., Yao X., (2004), "Evolutionary programming using mutations based on the Levy probability distribution", IEEE Trans. on Evolutionary Computation 8 (1).

[20] Wu H., He J., Yao X., (2005), "A online demo of evolutionary programming using mixed mutation strategy for solving function optimization", UKCI, 264–271.

[21] Chellapilla K., Fogel D.B., "Two new mutation operators for enhanced search and optimization in evolutionary programming". Proc. SPIE Int. Symp. on Optical science and engineering instrument., Bellingham, WASPIE Press, pp. 260–269.

[22] Schwefel H. P., (1981), Numerical Optimization of Computer Models, Chichester, U.K.: Wiley.

[23] Saravanan N., Fogel D. B., (1994), "Learning of strategy parameters in evolutionary programming: an empirical study," in Proc. 3rd Annual Conf. Evolutionary Programming, pp. 269–280.

[24] Mallipeddi R., Suganthan P.N., (2008), "Evaluation of novel adaptive evolutionary programming on four constraint handling techniques", IEEE Congress on Evolutionary Computation, 4045–4052.

[25] Liang K. H., Yao X., Newton C. S., (2001), "Adapting Self-Adaptive Parameters in Evolutionary Algorithms", Applied Intelligence 15, pp. 171–180.

[26] B¨ack T., (1996), Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University Press: New York.

[27] Liang K. H., Yao X., Liu Y., Newton C., Hoffman D., (1999), "An experimental investigation of self-adaptation in evolutionary programming," in Evolutionary Programming VII: Proc. of the Seventh Annual Conference on Evolutionary Programming, edited by

[28] Vologodskii A. V., (1999), Biophysical Chemistry Textbook, 5th ed., online book, Bloomfield.

[29] Montaz M.A., khompatraporn C., Zabinsky Z. B., (2005), "A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems", Journal of Global Optimization, Springer.

[30] Chellapilla K., Rao S., (1998), "Optimization of Bilinear Time Series models Using Fast Evolutionary Programming", IEEE signal processing letters, vol. 5, no. 2.

[31] Derrac J., García S., Molina D., Herrera F., (2011), "A practical tutorial on the use of on parametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms", Swarm and Evolutionary Computation 1, 3–18.

[32] Doksum K., (1967), "Robust procedures for some linear models with one observation per cell", Annals of Mathematical Statistics 38, 878–883.

[33] García S., Fernández A., Luengo J., Herrera F., (2010), "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," Information Sciences 180,2044–2064.