Information Processing Letters 82 (2002) 55–63

Information Processing Letters

www.elsevier.com/locate/ipl

Performance improvement of self-adaptive evolutionary methods with a dynamic lower bound

Anjan Kumar Swain [*], Alan S. Morris

Department of Automatic Control and Systems Engineering, University of Sheffield, UK

Abstract

Recent research on self-adaptive evolutionary programming (EP) methods evidenced the problem of premature convergence. Self-adaptive evolutionary programming methods converge prematurely because their object variables evolve more slowly than do their strategy parameters, which subsequently leads to a stagnation of object variables at a non-optimum value. To address this problem, a dynamic lower bound has been proposed, which is defined here as the differential step lower bound (DSLB) on the strategy parameters. The DSLB on an object variable depends on its absolute distance from the corresponding object variable of the best individual in the population pool. The performance of the self-adaptive EP algorithm with DSLB has been verified over eight different test functions of varied complexities. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Evolutionary computing algorithms; Self-adaptive evolutionary algorithms; Dynamic lower bound; Differential step lower bound; Evolutionary programming

1. Introduction

Evolutionary computing (EC) algorithms broadly cover three distinct areas: genetic algorithms (GAs) [8,3], evolution strategies (ESs) [13], and evolutionary programming (EP) [6,5]. The widely used genetic algorithms model evolution on the basis of observed genetic mechanisms, i.e., gene level modeling. Evolution strategy algorithms model evolution of individuals to better exploit their environment, and use a pure deterministic selection. Whereas, evolutionary programming algorithms model evolutions of individuals of multiple species competing for shared resources, and essentially utilize a stochastic selection.

All EC algorithms, in general terms, can be described with the four operations of reproduction (r), inheritance (i), variation (v), and selection (s). This can be expounded as that the evolution will take place as long as individuals generate copies of themselves that inherit their parent characteristics with some degree of variation (mutation, recombination, etc.) along with a selection method that selects individuals to survive and reproduce. This can simply be described mathematically for a population P consisting of individuals p_i, $\forall i \in \{1, \dots, \mu\}$, which are subjected to a series of operators such that

$$P(t+1) = s\big(v(P(t))\big), \tag{1}$$

where $P(t)$ is a vector of individuals p_i with each p_i representing a vector of object variables p_{ij}: $j \in \{1, \dots, n_0\}$ at generation "t" under a particular representation, $v(\cdot)$ is the random variation operator,

[*] Corresponding author. Currently in the Department of Electrical Engineering, Indira Gandhi Institute of Technology, Sarang, India.

E-mail addresses: swainanjan@hotmail.com (A.K. Swain), a.morris@sheffield.ac.uk (A.S. Morris).

and $s(\cdot)$ is the selection operator. Now, it can be seen that all EC algorithms carry out reproduction of individuals either by just copying the parent individuals or by using variation operators such as recombination, mutation etc. to allow inheritance with variation. Usually, variation operators reproduce with inheritance in addition to small variations. Hence, reproduction and inheritance operators are not explicitly included in the above expression. The action of variation operator on parent individuals generates offspring. Then, the selection operator $s(\cdot)$ operates on both offspring and/or parent individuals to determine which of the individuals will have the opportunity to reproduce. This selection operator usually depends on the fitness or the quality of the individuals determined by some predefined criteria.

In EC methods, the parameters that control the evolutionary search are defined as *strategy parameters*. Thus, mutation rates, mutation variances and recombination probabilities are classified as strategy parameters. Adaptation of these strategy parameters is precisely called *parameter control*, whereas setting the strategy parameters before executing the EC algorithm may be called *parameter tuning* [4].

The parameter control can be classified into *fixed/deterministic*, *adaptive* and *self-adaptive* parameter control methods. In fixed parameter control, the strategy parameters are updated by some deterministic and fixed rule [11] without using any explicit feedback from the search. The adaptive parameter control method explicitly uses feedback information from the search to update the strategy parameters during the evolution process (e.g., Rechenberg's $1/5$ success rule in [14]). Self-adaptive parameter control involves the updating of strategy parameters by considering them as a part of the evolution process, i.e., the strategy parameters are allowed to evolve along with the object variables.

In recent years, there has been much effort to increase the overall performance of EP on a variety of problem domains. All the varieties of EP algorithms use the basic self-adaptive EP algorithm, which is also known as canonical EP (CEP), as the baseline algorithm. The CEP method is described in Section 2. The CEP method uses Gaussian mutation operator to generate offspring. A Gaussian mutation operator generates offspring by adding Gaussian random variates to parent individuals. The major disadvantage of self-adaptive evolutionary algorithms is the premature convergence of the solution due to the rapid drop in the values of the strategy parameters compared to the corresponding object variables to very low values. Thus, it becomes practically ineffective to progress any further evolution. This was first reported by Liang et al. [9]. They observed that self-adaptive evolutionary algorithms are not even able to find a global optimum for simple functions. Then, they suggested the use of a fixed lower bound on each of the strategy parameters to improve the overall performance of these algorithms. Subsequently, Liang et al. [10] proposed a dynamic lower bound on strategy parameters η_{ij} that resulted in performance improvement on some test functions. Due to the lack of any concrete method to avoid the premature convergence, researchers usually use a fixed lower bound on the strategy parameters [2]. This method is known as the CEP with fixed lower bound (CEPFLB). Recently, Glickman and Sycara [7] presented three conditions that may be the possible causes of the premature convergence of all self-adaptive evolutionary algorithms. However, their experimentation and assertions are mainly based on training recurrent artificial neural networks (RANNs). In general, artificial neural networks used for system modeling often generate local models rather than global ones [16]. Hence, relatively large numbers of standard global optimization problems should be used for any general conclusion on global optimization methods.

In this paper, a dynamic lower bound on η_{ij} has been proposed, which depends on the distance between the jth object variable of the ith individual and the corresponding object variable of the fittest individual in the population pool. This is based on the concept that a particular object variable can search effectively only if the search domain around that individual is comparable with it. This resulted in a very effective method whose efficacy has been tested on eight test functions of various complexities. The proposed method has been named as CEP with differential step lower bound (CEPDSLB). The convergence results of the CEP, CEPFLB, and CEPDSLB are compared, and also necessary statistical tests have been performed to compare their statistical significance.

2. Self-adaptive evolutionary programming

Self-adaptive evolutionary programming (EP) algorithms have been used extensively in recent years for global numerical optimization problems [5,12]. The well-established self-adaptive EP methods work by evolving simultaneously all the object variables and their corresponding strategy parameters. A particular set of strategy parameters associated with an individual survives only when it produces better object variables. The most common variant of self-adaptive EP is the canonical self-adaptive EP (CEP) where these methods modify the individual representation by incorporating strategy parameters into them. Thus, the ith individual p_i in a population pool $P = [p_i],\ \forall i \in \{1, \dots, \mu\}$ can now be redefined as:

$$p_i = \{(p_{ij}, \eta_{ij}) \mid i = 1, \dots, \mu;\ j = 1, \dots, n\}, \tag{2}$$

where n is the number of object variables and also the number of standard deviations; η_{ij} is the standard deviation or the strategy parameter associated with the jth object variable of the ith individual; μ is the number of individuals in the population pool. Then, the strategy parameters η_{ik} and the corresponding object variable p_{ij} can be updated as per the following equations:

$$\eta_{ij}(t+1) = \eta_{ij}(t) \exp\big(\tau N_{ij}(0,1) + \tau' N_i(0,1)\big), \tag{3}$$

$$p_{ij}(t+1) = p_{ij}(t) + \eta_{ij}(t+1) N_{ij}(0,1), \tag{4}$$

where η_{ij} and p_{ij} are the jth component of the ith strategy parameter vector and jth object variable of the ith individual, respectively; $N_i(0,1)$ and $N_{ij}(0,1)$ are one-dimensional Gaussian random variates with expectation zero and standard deviation one; and the exogenous parameters τ and τ' are set to $(\sqrt{2n})^{-1}$ and $(\sqrt{2\sqrt{n}})^{-1}$, respectively [1]. Here, $N_i(0,1)$ serves as a global factor allowing an overall change of the mutability in an individual-level and $N_{ij}(0,1)$ represents a local factor, thus allowing adjustment of each component of the individual acting at a component-level.

3. Differential step lower bound (DSLB)

It is evident from the previous discussions that the value of η_{ij} largely depends on the object variable p_{ij} from its global optimum. Unfortunately, the global optimum of a system is usually not known in advance. Hence, to deal with this problem, the concept of pseudoglobal optimum has been used [15]. A pseudoglobal optimum in a particular population pool is defined as the fittest individual in that pool. Hence, the lower bound on η_{ij} corresponding to the object variable p_{ij} is proposed to vary in proportion to its distance from the jth object variable of the fittest individual in that population pool. In this way, the distance information is incorporated into the adaptation equations of strategy parameters. In addition, this lower bound is active at the component level of an individual. This serves the purpose of not allowing η_{ij} to fall below its distance from the pseudoglobal optimum. The operation of this lower bound can be explained as that when η_{ij} is very small but the corresponding p_{ij} is far from the true global optimum, then the lower bound is dominant, and hence controls the convergence.

Now, mathematically DSLB can be represented as

$$b_{ij} \propto |p_{ij} - p_{kj}|. \tag{5}$$

The basic philosophy of real number mutation works with the concept that small variations are more likely to occur than large variations because a Gaussian random distribution is utilized. By extending this concept to b_{ij}, we have

$$b_{ij} = \gamma |p_{ij} - p_{kj}| N_j(0,1), \tag{6}$$

where $N_j(0,1)$ is a normal distribution with zero mean and unity standard deviation, and $\gamma = 1/\sqrt{n}$ is the proportionality constant for CEP. The factor γ is system dependent, i.e., it is very unlikely that this factor will remain same for all other variants of CEP.

It can be seen that, when the object variables are situated far from the pseudoglobal optimum and the initial search domain is very large, then the probability of b_{ij} being large is greater. Hence, this makes the strategy parameters to be larger than the initial η_{ij} values. This can be avoided with the following heuristics:

$$\eta_{ij}(k+1) = \begin{cases} \eta_{ij} + b_{ij}, & \eta_{ij} + b_{ij} < (\eta_{ij})_{\text{initial}}, \\ \eta_{ij}, & \text{otherwise}. \end{cases} \tag{7}$$

With the progress in the evolution, the strategy parameter η_{ij} reduces to a low value. Hence, at the start of the evolution process, the mutation is very large and the main effect is to explore the search space.

Then, at later generations, the search domain narrows down and so it is more likely to exploit the search space. This concept of varying the strategy parameter to avoid the premature convergence leads to a novel dynamically varying lower bound. This dynamic lower bound is defined here as the differential step lower bound (DSLB).

4. Simulation method and results

For the experimental verification of the performance of the algorithms to find the global minimum, a large set of 23 benchmark functions, which are the same as those originally considered by Yao et al. [17], have been used. This large set of functions ensures that the algorithms used for verification are not biased towards a particular class of problems. These functions can be grouped as follows:

(i) Functions f_1 to f_{13} are high-dimensional problems. Out of these, the functions f_1 to f_7 are unimodal, and f_8 to f_{13} are multimodal functions with many local minima. The number of these local minima increases with the dimension of the problem. Further, function f_6 is a discontinuous unimodal step function and f_7 is a unimodal noisy quartic function with an additive uniform random number in the range 0 to 1.

(ii) Functions f_{14} to f_{23} are all low-dimensional. Further, these are all multimodal functions with few local minima.

All the 23 benchmark functions are given in Table 1.

All the experiments on the CEP and FEP method have been performed under exactly the same conditions with initial standard deviation $\eta_{ij} = 3$, $\forall i \in \{1, \ldots, \mu\}$ and $\forall j \in \{1, \ldots, n\}$, and the same initial population size $\mu = 100$. For the CEPFLB method, the fixed lower bound is taken as 10^{-4}. The simulation results for all the 23 benchmark functions are shown in Table 2. This table shows the average best and average mean results for all the benchmark functions. Average best and average mean indicate the average of the best scores and the mean scores of all the individuals in a population pool after the indicated number of generations, over 50 runs for all the functions. In addition to this, the standard deviations of the best and mean scores of all the benchmark functions have also been indicated inside the brackets in Table 2. The standard deviation of the 50 best and mean results obtained after the specified number of generations depict the distribution of the best and mean results around their respective means. This essentially shows the consistency of the results over 50 runs. In general, lower values of the standard deviation signify better consistency of the obtained results. In most of the test cases, the CEPDSLB exhibited statistically consistent results.

Also, in Table 2, the statistical t-test results with 49 degrees of freedom are presented along with the significance values. The t-test is performed to test the statistical significance of the results obtained for the three different methods. The negative t-scores indicate the improved performance of the first of the two methods under test, whereas positive t-scores indicate the superior performance of the second method.

Fig. 1 shows the progress of the average best values of the population by CEP over 50 runs for the unimodal functions f_1, f_2, f_5 and f_7. It can be observed that, on all the functions, CEP with differential step lower bound (CEPDSLB) performs consistently better than CEP with no lower bound (CEPNLB) and CEP with fixed lower bound (CEPFLB). For functions f_2, f_5 and f_7, the results of CEPDSLB and CEPFLB are comparable. This shows that a fixed lower bound of 10^{-4} serves adequately on these functions. The results for functions f_4 and f_6 not shown here of the CEPFLB is almost identical with that of CEPNLB, whereas the CEPDSLB outperforms CEPFLB. This indicates that CEPFLB does not yield better results on all the functions.

The performance of CEPNLB, CEPFLB, and CEPDSLB for high-dimensional multimodal functions with many local minima are shown in Fig. 2 for f_8 and f_{11}. Here, the performance of CEPFLB and CEPNLB do not differ at all for functions f_8 to f_{10}. Whereas, on functions f_{11} to f_{13}, the CEPFLB exhibits some improvements over CEPNLB. However, on all the functions f_8 to f_{13}, CEPDSLB outperforms CEPFLB and CEPNLB.

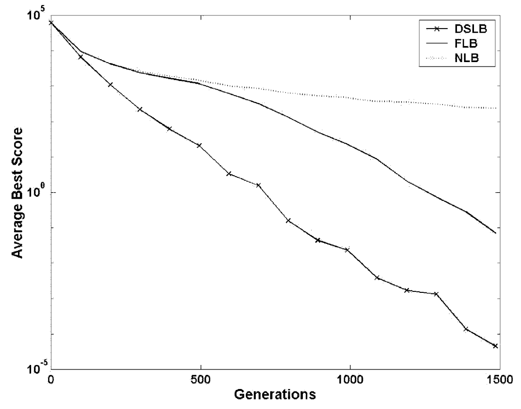
On the low-dimensional functions f_{14} to f_{23}, the performance of all the methods are very similar. However, as shown in Table 2, on f_{14}, f_{22} and f_{23}, the performance of CEPDSLB is impressively better than CEPFLB and CEPNLB. This is also shown in Fig. 3 for f_{14} and f_{23}. On functions f_{19} and f_{20}, the convergence of CEPDSLB is faster than both CEPNLB and CEPFLB. This is shown in Fig. 4.

Table 1
The 23 benchmark functions, where n_0 is the function dimension, $f_{\min}$ is the minimum function value and SD is the user supplied search domain

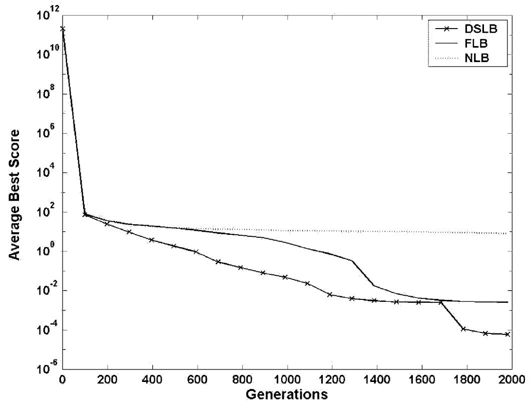
Test functions	n_0	SD	$f_{\min}$
$f_1(x) = \sum_{i=1}^{n} x_i^2$	30	$[-100, 100]^{n_0}$	0
$f_2(x) = \sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$	30	$[-10, 10]^{n_0}$	0
$f_3 = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	30	$[-100, 100]^{n_0}$	0
$f_4(x) = \max_i \{\lvert x_i\rvert, 1 \leqslant i \leqslant n\}$	30	$[-100, 100]^{n_0}$	0
$f_5(x) = \sum_{i=1}^{n-1} \{100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\}$	30	$[-30, 30]^{n_0}$	0
$f_6(x) = \sum_{i=1}^{n} ([x_i + 0.5])^2$	30	$[-100, 100]^{n_0}$	0
$f_7(x) = \sum_{i=1}^{n} i x_i^4 + random(0, 1)$	30	$[-1.28, 1.28]^{n_0}$	0
$f_8(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i\rvert})$	30	$[-500, 500]^{n_0}$	0
$f_9 = \sum_{i=1}^{n} \{x_i^2 - 10\cos(2\pi x_i) + 10\}$	30	$[-5.12, 5.12]^{n_0}$	0
$f_{10}(x) = -20\exp(-0.2\sqrt{(1/n)\sum_{i=1}^{n} x_i^2})$ $- \exp((1/n)\sum_{i=1}^{n} \cos(2\pi x_i)) + 20 + \exp(1)$	30	$[-32, 32]^{n_0}$	0
$f_{11} = 0.00025\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1$	30	$[-600, 600]^{n_0}$	0
$f_{12}(x) = (\pi/n)\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\}$ $+ \sum_{i=1}^{n} u(x_i, 10, 100, 4), \quad y_i = 1 + 0.25(x_i + 1)$ $u(x_i, a, k, m) = k(x_i - a)^m,\ x_i > a;\ 0, -a \leqslant x_i \leqslant a;\ k(-x_i - a)^m,\ x_i < a$	30	$[-50, 50]^{n_0}$	0
$f_{13}(x) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})]$ $+ (x_n - 1)^2 + [1 + \sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	30	$[-50, 50]^{n_0}$	0
$f_{14}(x) = [0.002 + \sum_{j=1}^{25} (j + \sum_{i=1}^{n} (x_i - a_{ij})^6)^{-1}]^{-1}$	2	$[-65.536, 65.536]^{n_0}$	−0.980004
$f_{15}(x) = \sum_{i=1}^{11} [a_i - x_1(b_i^2 + b_i x_2)(b_i^2 + b_i x_3 + x_4)^{-1}]^2$	4	$[-5, 5]^{n_0}$	0.0003075
$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + (1/3)x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	$[-5, 5]^{n_0}$	−1.0316285
$f_{17}(x) = (x_2 - (5.1/(4\pi^2))x_1^2 + (5/\pi)x_1 - 6)^2 + 10(1 - (1/8\pi))\cos x_1 + 10$	2	$[-5, 10]^{n_0}$ $[0, 15]^{n_0}$	0.398
$f_{18}(x) = \lfloor 1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\rfloor$ $\times \lfloor 30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\rfloor$	2	$[-2, 2]^{n_0}$	3.0
$f_{19}(x) = -\sum_{i=1}^{4} c_i \exp\lfloor -\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\rfloor$	3	$[0, 1]^{n_0}$	−3.86
$f_{20}(x) = -\sum_{i=1}^{4} c_i \exp\lfloor -\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\rfloor$	6	$[0, 1]^{n_0}$	−3.32
$f_{21}(x) = -\sum_{i=1}^{5} [(x - a_i)(x - a_i)^{\mathrm{T}} + c_i]^{-1}$	4	$[-10, 10]^{n_0}$	−10
$f_{22}(x) = -\sum_{i=1}^{5} [(x - a_i)(x - a_i)^{\mathrm{T}} + c_i]^{-1}$	4	$[-10, 10]^{n_0}$	−10
$f_{23}(x) = -\sum_{i=1}^{10} [(x - a_i)(x - a_i)^{\mathrm{T}} + c_i]^{-1}$	4	$[-10, 10]^{n_0}$	−10

Table 2
Performance comparison of self-adaptive methods on functions $f_1 - f_{23}$ with no lower bound, fixed lower bound and dynamic lower bound. Results are averaged over 50 runs and indicate the value at the end of the mentioned generations. The t-test results with 49 degrees of freedom between two methods (NLB-DSLB/FLB-DSLB) favors the second method (DSLB) if is positive and vice versa. The numbers in the brackets with mean best and mean average indicate the standard deviation, and that with t-test indicates the significance factor. Here, NLB, DSLB, and FLB stand for no lower bound, differential step lower bound, and fixed lower bound, respectively

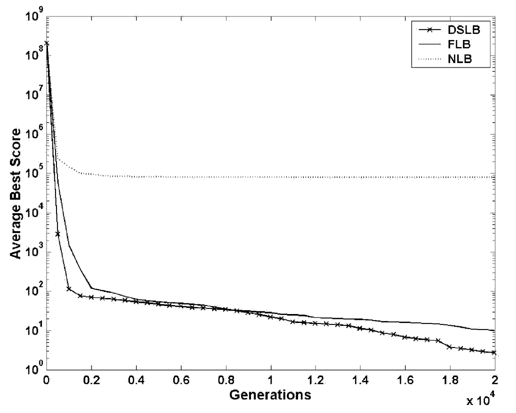
f_n	No. gene.	NLB		FLB		DSLB		t-test results	
		Mean best	Mean average	Mean best	Mean average	Mean best	Mean average	NLB-DSLB	FLB-DSLB
f_1	1500	2.380e02	2.387e02	5.745e−02	6.653e−02	2.524e−06	3.360e−06	4.0879	3.13844
		(411.651)	(411.381)	(1.29e−01)	(1.45e−01)	(2.95e−04)	(2.95e−04)	(1.61e−04)	(2.87e−03)
f_2	2000	8.013e00	8.014e00	2.575e−03	3.421e−03	5.901e−05	5.941e−05	7.00179	42.1379
		(8.09239)	(8.09262)	(2.32e−04)	(2.04e−04)	(3.29e−04)	(3.29e−04)	(6.59e−09)	(3.73e−40)
f_3	5000	2.998e03	2.998e03	4.227	4.310	1.586	1.587	11.4898	1.54641
		(1843.86)	(1843.89)	(8.04)	(8.15)	(8.56)	(8.56)	(1.64e−15)	(0.12844)
f_4	5000	2.866e00	2.875e00	1.636e00	1.645	1.050e−02	1.072e−02	12.4887	9.97549
		(1.61869)	(1.62454)	(1.15e00)	(1.16)	(1.06e−02)	(1.07e−02)	(7.67e−17)	(2.18e−13)
f_5	20 000	8.014e04	8.014e04	1.025e01	1.026e01	2.674e00	2.679e00	1.707720	1.83458
		(331810)	(331810)	(28.4028)	(28.4156)	(4.2328)	(4.23768)	(9.40e−02)	(7.26e−02)
f_6	1500	1.915e03	1.915e03	1.549e03	1.549e03	1.223e02	1.223e02	7.22823	6.57706
		(1839.23)	(1839.23)	(1.52e03)	(1.52e03)	(3.84e02)	(3.84e02)	(2.94e−09)	(3.00e−08)
f_7	3000	2.230e01	6.909e01	8.774e−01	9.636e−01	2.482e−01	3.211e−01	4.60388	1.29758
		(33.7182)	(187.69)	(3.18)	(3.28)	(1.23855)	(1.23999)	(2.97e−05)	(0.20051)
f_8	9000	−7894.610	−7894.610	−7965.082	−7965.082	−9330.650	−9330.650	11.68	10.1247
		(601.779)	(601.779)	(702.89)	(702.89)	(682.442)	(682.442)	(9.08e−16)	(1.33e−13)
f_9	5000	1.096e02	1.096e02	1.065e02	1.065e02	6.195e01	6.195e01	10.261	9.99864
		(25.1061)	(25.1061)	(2.62e01)	(2.62e01)	(1.57e01)	(1.57e01)	(8.50e−14)	(2.02e−13)
f_{10}	1500	9.212	9.212	9.675	9.675	1.543	1.543	16.6964	17.6761
		(2.83466)	(2.83476)	(2.36)	(2.36)	(1.8783)	(1.87829)	(7.36e−22)	(6.65e−23)
f_{11}	2000	9.227e00	9.230e00	8.350e−01	8.356e−01	1.061e−01	1.061e−01	4.98405	1.70412
		(12.9783)	(12.9776)	(3.01e01)	(3.01e01)	(2.22e−01)	(2.22e−01)	(8.17e−06)	(9.47e−02)
f_{12}	1500	5.976	5.977	2.545	2.571	2.159e−01	2.159e−01	8.79628	7.08787
		(4.63038)	(4.62998)	(2.22)	(2.23)	(3.65e−01)	(3.65e−01)	(1.19e−11)	(4.85e−09)
f_{13}	1500	3.545e04	3.765e04	4.670	4.746	1.349e−01	1.350e−01	1.014	6.5483
		(247189)	(262107)	(4.94)	(4.99)	(5.24e−01)	(5.24e−01)	(0.31556)	(3.32e−08)
f_{14}	100	1.820683	2.146929	1.857361	2.057721	1.256056	1.256057	3.08269	2.94697
		(1.22191)	(1.69498)	(1.43)	(1.66)	(6.28e−01)	(6.28e−01)	(3.36e−03)	(4.90e−03)
f_{15}	4000	9.501e−04	9.502e−04	8.319e−04	8.317e−04	8.753e−04	8.756e−04	1.61831	−0.927863
		(1.78e−04)	(1.78e−04)	(2.33e−04)	(2.32e−04)	(2.65e−04)	(2.65e−04)	(0.11202)	(0.35803)
f_{16}	100	−1.031628	−1.031628	−1.031628	−1.031628	−1.031626	−1.031001	−1.00000	−1.00000
		(4.49e−16)	(4.49e−16)	(4.49e−16)	(4.49e−16)	(1.51e−05)	(4.43e−03)	(0.32222)	(0.32222)
f_{17}	100	0.3978874	0.3978874	0.3978874	0.3978874	0.3978874	0.3978874	0.000000	0.000000
		(2.24e−16)	(2.24e−16)	(2.24e−16)	(2.24e−16)	(2.24e−16)	(2.83e−08)	(1.00000)	(1.00000)
f_{18}	100	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	0.000000	0.000000
		(0.00000)	(0.00000)	(0.00000)	(1.35e−15)	(0.00000)	(0.00000)	(1.00000)	(1.00000)
f_{19}	100	−3.862782	−3.862780	−3.862749	−3.862633	−3.862782	−3.862782	1.000000	1.00122
		(2.83e−07)	(6.29e−06)	(2.36e−04)	(1.05e−03)	(1.79e−15)	(7.82e−07)	(0.32222)	(0.32164)
f_{20}	200	−3.202121	−3.201037	−3.226785	−3.220138	−3.236390	−3.236380	1.17849	0.480055
		(0.20405)	(0.20425)	(1.36e−01)	(1.79e−01)	(5.39e−02)	(5.39e−02)	(0.24429)	(0.63332)
f_{21}	100	−7.192846	−7.183604	−8.016078	−7.891391	−8.146606	−7.928017	1.45843	0.204098
		(3.34144)	(3.33373)	(2.87)	(3.01)	(3.00)	(3.20)	(0.15110)	(0.83912)
f_{22}	100	−8.830418	−8.611691	−8.672426	−8.588784	−9.598160	−9.574040	1.47861	1.86155
		(2.88593)	(3.08769)	(2.87)	(2.97)	(2.23)	(2.28)	(0.14565)	(6.87e−02)
f_{23}	100	−8.99050	−8.690520	−8.760382	−8.694442	−10.22436	−10.18778	3.07897	2.91731
		(3.00038)	(3.16862)	(3.07)	(3.17)	(1.43)	(1.47)	(3.40e−03)	(5.32e−03)



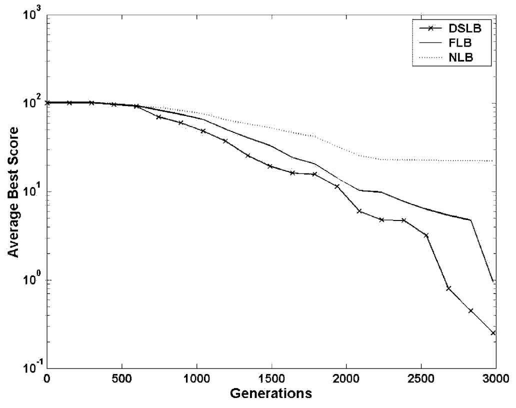
f$_1$



f$_2$

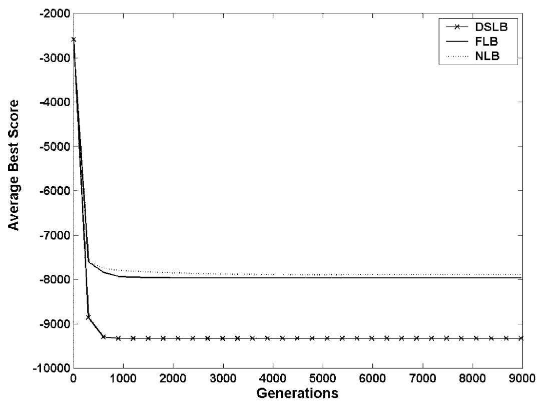


f$_5$

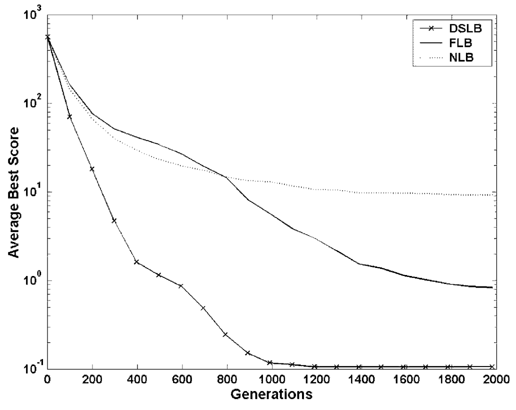


f$_7$

Fig. 1. Average best results averaged over 50 runs of CEPDSLB, CEPFLB and CEPNLB on unimodal functions f_1, f_2, f_5 and f_7.

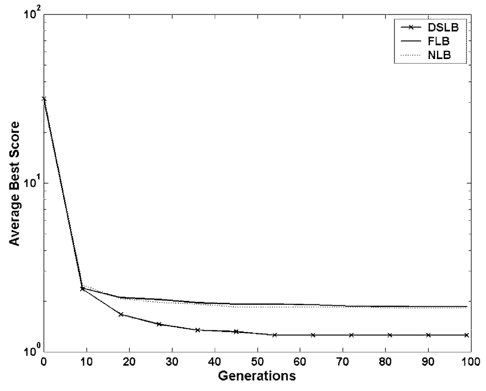


f$_8$

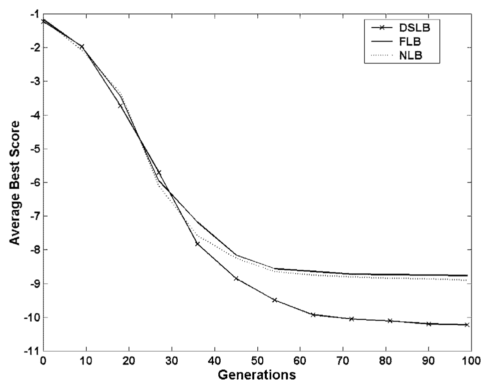


f$_{11}$

Fig. 2. Average best results averaged over 50 runs of CEPDSLB, CEPFLB and CEPNLB on multimodal functions f_8 and f_{11}.

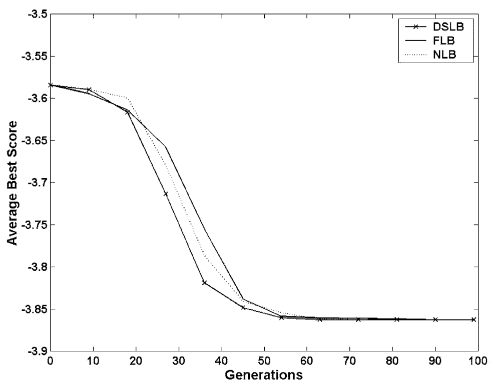


f$_{14}$

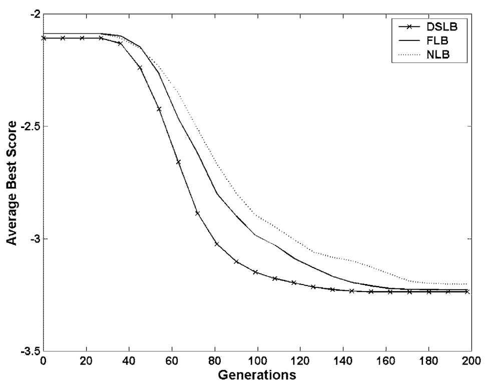


f$_{23}$

Fig. 3. Average best results averaged over 50 runs of CEPDSLB, CEPFLB and CEPNLB on multimodal functions f_{14} and f_{23}.



f$_{19}$



f$_{20}$

Fig. 4. Average best results averaged over 50 runs of CEPDSLB, CEPFLB and CEPNLB on multimodal functions f_{19} to f_{20}.

5. Conclusions

In this paper, a novel dynamic lower bound on the self-adaptive EP has been proposed for the function optimization task. The power and effectiveness of the proposed scheme have been shown by simulating the results on eight well investigated and most typical test functions. Then, the results are compared with that of the results obtained from CEP and CEPFLB. In all the cases, the EP with lower bound converges to a region very close to the global optimum faster than its conventional counterparts. Thus, this algorithm proves to be faster, accurate and robust. This concept has also been tested on the fast evolutionary programming (FEP) method as proposed in [17], and the results with lower bound were consistently better than that without lower bound.

Acknowledgement

Authors thank the reviewers for their constructive comments to improve the quality of the paper.

References

[1] T. Bäck, H.-P. Schwefel, An overview of evolutionary algorithms for parameter optimization, Evolutionary Comput. 1 (1) (1993) 1–23.

[2] K. Chellapilla, Combining mutation operators in evolutionary programming, IEEE Trans. Evolutionary Comput. 2 (3) (1998) 91–96.

[3] L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

[4] Á.E. Eiben, R. Hinterding, Z. Michalewicz, Parameter control in evolutionary algorithms, IEEE Trans. Evolutionary Comput. 3 (2) (1999) 124–141.

[5] D.B. Fogel, Evolutionary Computation: Towards a New Philosophy of Machine Intelligence, IEEE Press, New York, 1995.

[6] L.J. Fogel, A.J. Owens, M.J. Walsh, Artificial Intelligence through Simulated Evolution, John Wiley, New York, 1966.

[7] M.R. Glickman, K. Sycara, Reasons for premature convergence of self-adapting mutation rates, in: Proc. Congress on Evolutionary Computation (CEC2000), San Diego, CA, 2000, pp. 62–69.

[8] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing, Reading, MA, 1988.

[9] K.-H. Liang, X. Yao, C. Newton, D. Hoffman, An experimental investigation of self-adaptation in evolutionary programming, in: Proc. 7th Internat. Conf. Evolutionary Programming, Springer, Berlin, 1998, pp. 291–300.

[10] K.-H. Liang, X. Yao, C. Newton, Dynamic control of adaptive parameters in evolutionary programming, in: Proc. Simulated Evolution and Learning (SEAL98): Second Asia Pacific Conference, Springer, Berlin, 1998, pp. 42–49.

[11] Z. Michalewicz, Genetic Algorithms + Data Structure = Evolution Programs, Springer, New York, 1994.

[12] N. Sarvanan, D.B. Fogel, K.M. Nelson, A comparison of methods for self-adaptation in evolutionary algorithms, BioSystems 36 (1995) 157–166.

[13] H.P. Schwefel, Numerical Optimization of Computing Models, John Wiley, Chichester, UK, 1981.

[14] H.-P. Schwefel, Evolution and Optimum Seeking, Wiley, New York, 1995.

[15] A.K. Swain, A.S. Morris, A novel hybrid evolutionary programming method for function optimization, in: Proc. Congress on Evolutionary Computation (CEC2000), San Diego, CA, 2000, pp. 1369–1376.

[16] A.K. Swain, Dynamic modelling and control of robotic manipulators with an investigation of evolutionary computation methods, PhD dissertation, University of Sheffield, UK, 2001.

[17] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Trans. Evolutionary Comput. 3 (2) (1999) 82–102.