A Step Size Based Self-Adaptive Mutation Operator for Evolutionary Programming

Libin Hong
School of Computer Science,
University of Nottingham
P.R.C.
Libin.HONG@nottingham.edu.cn

John H. Drake
School of Computer Science,
University of Nottingham
P.R.C.
John.DRAKE@nottingham.edu.cn

Ender Özcan
School of Computer Science,
University of Nottingham
U.K.
Ender.OZCAN@nottingham.ac.uk

ABSTRACT

The mutation operator is the only genetic operator in Evolutionary Programming (EP). In the past researchers have nominated Gaussian, Cauchy, and Lévy distributions as mutation operators. According to the No Free Lunch theorem [9], no single mutation operator is able to outperform all others over the set of all possible functions. Potentially there is a lot of useful information generated when EP is ongoing. In this paper, we collect such information and propose a step size based self-adaptive mutation operator for Evolutionary Programming (SSEP). In SSEP, the mutation operator might be changed during the evolutionary process, based on the step size, from generation to generation. Principles for selecting an appropriate mutation operator for EP is proposed, with SSEP grounded on the principles. SSEP is shown to outperform static mutation operators in Evolutionary Programming on most of the functions tested. We also compare the experimental results of SSEP with other recent Evolutionary Programming methods, which uses multiple mutation operators.

Keywords

Evolutionary Programming, Step Size, Mutation Operator, Self-Adaptive, Function Optimization, Convergence.

1. INTRODUCTION

Evolutionary Programming (EP) is a branch of Evolutionary Computation used to evolve numerical values, in an attempt to find a global optimum of a function. The only genetic operator available in an EP system is mutation. Probability distributions used as mutation operators in EP include Gaussian, Cauchy and Lévy, among others. Classical Evolutionary Programming (CEP) [10] uses Gaussian mutation, while Fast Evolutionary Programming (FEP) [10] uses Cauchy mutation. For Evolutionary Programming methods using Lévy distribution as a mutation operator (LEP) [4], it is also possible perform Gaussian or Cauchy mutation through the settings of the α parameter.

GECCO'14, July 12-16, 2014, Vancouver, BC, Canada.

In recent years, many improvements to EP have been proposed. Improved FEP (IFEP) [11], mixes mutation operators, and uses both Gaussian and Cauchy mutations. Later a mixed mutation strategy (MSEP) [2] was proposed: four mutation operators are used and the mutation operator is selected according to their probabilities during the evolution. In 2010 MSEP with local fitness landscape (LMSEP) [8] was proposed: based on MSEP, local fitness landscape is considered. Ensemble Strategies with Adaptive Evolutionary Programming [7] was also proposed in 2010: each mutation operator has an associated population, with every population benefiting from every function call. In 2013, researchers proposed to automatically design the mutation operator for EP using Genetic Programming [3].

This paper proposes a novel method, using a self-adaptive mutation operator to keep the convergence rate stable for EP throughout the evolutionary process. From previous research and our observations, useful information is collected to help EP make a decision which mutation operator to use at each generation. We propose principles to design an algorithm which utilises this information. According to the principles, a step size based self-adaptive mutation operator for Evolutionary Programming (SSEP) is presented.

Section 2 introduces function optimization and the EP algorithm. In Section 3 we describe the principles used to design the SSEP mutation operator. Section 4 provide the details of the EP designed by the principles. In Section 5 we present and compare the experimental results of Gaussian, Cauchy, and Lévy mutation with the SSEP. We also compare SSEP to the MSEP and LMSEP methods take from the literature. In Section 6 we discuss and explain future work. Section 7 summarizes and concludes the article.

2. FUNCTION OPTIMIZATION BY EVOLUTIONARY PROGRAMMING

Global minimization in $\mathbb{R}^n$ can be formalized as a pair (S, f), where $S \in \mathbb{R}^n$ is a bounded set on $\mathbb{R}^n$ and $f : S \to \mathbb{R}$ is an n-dimensional real-valued function. The aim is to find a point $x_{\min} \in S$ such that $f(x_{\min})$ is a global minimum on S. More specifically, it is required to find an $x_{\min} \in S$ such that

$$\forall x \in S : f(x_{\min}) \leq f(x)$$

Here f does not need to be continuous or differentiable but it must be bounded. The algorithm of EP is implemented as follows [1, 10]:

1. *Generate the initial population of p individuals, and set $k = 1$. Each individual is taken as a pair of real-valued*

vectors, (x_i, η_i), $\forall i \in \{1, \cdots, \mu\}$. The initialization value of the strategy parameter η is set to 3.0.

2. *Evaluate the fitness value for each (x_i, η_i), $\forall i \in \{1, \cdots, \mu\}$.*

3. *Each parent (x_i, η_i), $\forall i \in \{1, \cdots, \mu\}$, creates λ/μ offspring on average, so that a total of λ offspring are generated: for i=1, $\cdots$, μ, j=1, $\cdots$, n.*

$$x_i'(j) = x_i(j) + \eta_i(j)D_j \quad (1)$$

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

The above two equations are used to generate new offspring. Objective function is used to calculate the fitness value, the survival offspring is picked up according to the fitness value. The factors γ and γ' have set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(\sqrt{2n})^{-1}$.

4. *Evaluate the fitness of each offspring (x_i', η_i'), $\forall i \in \{1, \cdots, \mu\}$, according to $f(x')$.*

5. *Conduct pairwise comparison over the union of parents (x_i, η_i) and offspring (x_i', η_i'), $\forall i \in \{1, \cdots, \mu\}$. Q opponents are selected randomly from the parents and offspring for each individual. During the comparison, the individual receives a "win" if its fitness is no greater than those of opponents.*

6. *Pick the μ individuals out of parents and offspring, $i \in \{1, \cdots, \mu\}$, that have the most wins to be parents, to form the next generation.*

7. *Stop if the stopping criterion is satisfied; otherwise, k++ and goto Step3.*

If D_j in Eq.(1) is the Gaussian distribution, then the algorithm is CEP [10]. If D_j is the Cauchy distribution, it is FEP [11]. If D_j is the Lévy distribution, it is LEP [4]. The above algorithm acts as a template for SSEP, which is the contribution of this paper.

3. PRINCIPLES OF A STEP SIZE BASED SELF-ADAPTIVE MUTATION OPERATOR

In this section we introduce the principles used to design SSEP and explain how it works. The idea of SSEP is inspired by a driver tuning an analogue car radio. A driver will usually tune the radio in a wide range to search channels, and several channels may be discovered. Once the driver decides which channel to listen to, he will adjust the radio slightly to make the signal more clear. We try to design an EP system that moves in the search space with a long step size at the beginning, and uses a short step size later on in the search, where step size is the distance between a 'survived' offspring and its parent.

Liang et al. [5, 6] proposed to control the lower bound of offspring in order to improve the performance in EP, and noted that self-adaptation may rapidly lead to a search step size that is far too small to explore the search space any further. This is known as *loss of step size control*. Liang et al. [5, 6] also analyse how the step size control was lost. In this paper we propose to change the mutation operator in EP based on the step size that the parent jumped, rather than control the lower bound.

Yao et al. [11] point out that Cauchy mutation is more likely to generate an offspring further away from its parent than Gaussian mutation. In other words, Gaussian offers better performance if the parent is close to the global optimum, otherwise Cauchy mutation is a better choice for EP.

From a large number of EP experiments we observed an interesting phenomena: when using a static mutation operator in EP, usually more offspring survive in early generations and less offspring survive in later generations of the run. According to previous research conclusions and our own observations, we believe that in early generations of function optimization, a long step size mutation operator is necessary, as it guarantees more of the search space can be covered. However, once the individuals are close to the global optimum, a long step size mutation operator should be replaced by a short step size mutation operator in order to keep the convergence rate of the run consistent in later generations. But how 'short' is short enough? In previous research, a standard normal distribution $N(0,1)$ represents a short step size mutation operator. In our experiments we use $N(0, 0.1)$ or $N(0, 0.01)$ as 'short' step size mutation operators.

3.1 Symbols Used in Principles

Below are the symbols used in the principles:

1. S_k: a single value representing the single step size at each generation k. In EP, after tournament selection at each generation, the new population consists of both parents and offspring. The offspring which appear in the new population are known as "survived" offspring, S_k is calculated as the mean value of the jumped step size of all "survived" offspring. This value will be updated at each generation (in Algorithm 2 line 8), $\forall k \in \{1, \cdots, n\}$ where n is the maximum generation of EP.

2. S_H_k: a single value representing the average step size from generation 1 to the current generation k. This value is calculated as $mean(S_1 ... S_k)$ where k is current generation number. This value is also updated at each generation (in Algorithm 2 line 10).

3. $N(\mu, \sigma^2)$: Gaussian distribution, in SSEP ($\mu = 0, \sigma^2 = 0.1$) or ($\mu = 0, \sigma^2 = 0.01$) is used according to the principles.

4. C: represents the Cauchy distribution.

5. T: the distance coefficient. When S_k is very small, it helps EP to have sufficient generations with a long step size mutation operator to prevent it from falling into local optimum. In our experiments, the value of T for each function is given in Table 2. The values of T we set for SSEP are empirically determined.

3.2 Principles for SSEP

The following are the principles for SSEP:

1. If $S_k \in (1e-4, 1e-2]$ at the current generation and $S\_H_k \geq S_k \times T$, set a σ^2=0.1 for $N(0, \sigma^2)$, and use $N(0, 0.1)$ as the mutation operator for EP.

2. If $S_k \in (0, 1e-4]$ at the current generation and $S\_H_k \geq S_k \times T$, set σ^2=0.01 for $N(0, \sigma^2)$, and $N(0, 0.01)$ is selected as mutation operator for EP.

3. If $S\_H_k \geq 1$ or $S\_H_k \leq S_k \times T$, C is selected as mutation operator for EP.

4. For all other cases, σ^2 is set as S_H_k for $N(0, \sigma^2)$, and $N(0, S\_H_k)$ is selected as the mutation operator for EP.

Previous works in the literature often use standard normal distribution $N(0, 1)$ as a short step size mutation operator.

However according to our observations, the random values generated by $N(0,1)$ are not 'short' enough to keep convergence rate stable in later stages of EP. This is a particular problem when the population is 'very close' to the global optimum. Hence we use principle 1 and 2 to keep the convergence rate stable in later generations of EP. In SSEP, $S\_H_k = 1$ is a watershed, $S\_H_k \geq 1$ represents more offspring survived using long step size mutation operator. As a result we will use a mutation operator that can generate long step size values for the next generation, hence we propose principle 3.

Table 1 Parameter settings for EP.

Parameter	Settings
DIM	n in Table 3
$POPNO$	100

Algorithm 1 Algorithm to calculate step size for each population

```
1: DIM /*Initial dimensional size*/
2: POPNO /*Initial population number*/
3: pop[POPNO × 2][DIM]
4: /*Parents and offsprings in current generation*/
5: for i : POPNO do
6:    indStepSize = 0;
7:    for j : DIM do
8:       /*Calculate step size for each individual*/
9:       temp(i, j) = pop[i][j] - pop[POPNO + i][j]
10:      /*Calculate total step size for all individuals*/
11:      indStepSize += abs(temp(i, j))
12:   end for
13:   /*Calculate average step size for pop[i]*/
14:   popStepSize(POPNO + i) = indStepSize/DIM
15: end for
```

4. DETAILS OF SSEP IMPLEMENTATION

In this section we list the the pseudo-code of the SSEP algorithm designed according to the principles in Section 3.2.

Algorithm 1 is used to calculate the step size of the population at each generation of EP. The updating of S_k and S_H_k in Algorithm 2 will use the data prepared in this algorithm. Algorithm 1 is injected in between step 4 and step 5 of the EP algorithm introduced in Section 2.

Algorithm 2 implements the principles proposed in Section 3.2. Algorithm 2 is injected in between step 6 and step 7 of the EP algorithm introduced in Section 2.

5. EXPERIMENTAL RESULTS AND DISCUSSION

In previous work, most researchers have tested their algorithms on a benchmark suite of 23 functions [10]. In this paper, we use the first 10 (see Table 3).

Table 3 lists the functions used in this paper. In the function suite, f_1-f_7 are unimodal functions and f_8-f_{10} are multimodal functions.

Algorithm 2 Algorithm to implement SSEP

```
1: for i : POPNO do
2:    if pop[i] is offspring then
3:       totalPopStepSize += popStepSize[i]
4:       survivedOffspringNo ++
5:    end if
6: end for
7: /*update S_k*/
8: S_k = totalPopStepSize/survivedOffspringNo
9: /*update S_H_k*/
10: S_H_k = mean(S_1...S_k);
11: /*Initial T*/
12: if S_k < 1e − 2 and S_k > 1e − 4 and S_H_k ≥ S_k × T
    then
13:    /*If mutation type is 2, use N(0, σ^2) as mutation
       operator*/
14:    muttype ← 2
15:    σ^2 ← 0.1
16: else if S_k ≤ 1e − 4 and S_k < 0 and S_H_k ≥ S_k × T
    then
17:    muttype ← 2
18:    σ^2 ← 0.01
19: else if S_H_k ≥ 1 or S_H_k ≤ S_k × T then
20:    /*If mutation type is 1, use C as mutation operator*/
21:    muttype ← 1
22: else
23:    muttype ← 2
24:    σ^2 ← S_H_k
25: end if
```

The results of SSEP are given for each function in Table 4, alone with the results of various previous methods taken from the literature.

Table 5 shows the results of a Wilcoxon Signed-Rank Test within a 95% confidence interval between SSEP and other methods from literature. In this table, "$\geq$" and "$\leq$" indicate that SSEP performs better or worse on average respectively, compared to a particular existing method. In the case that this difference is statistically significant, "$>$" and "$<$" are used.

5.1 Analysis and Comparisons

The results in Table 4 show that SSEP gives the best performance when compared with using a static mutation operator on f_1, f_2, f_4 and f_7. The results of the 2-tailed Wilcoxon Signed-Rank Tests given in Table 5, show a statistically significant difference between SSEP and CEP, FEP and the LEP variations. SSEP also beats some of the static mutation operators on f_6, f_8 and f_{10}. We notice that SSEP has good performance on f_{10}, the original experimental data shows that the mean fitness value in final generation reaches a level of 10^{-5} more than 30 times over 50 runs.

The new methods proposed have successfully kept the convergence rate stable for some of the functions, not only in early generations, but also in later generations of EP. In general, SSEP outperforms EP methods using static mutation operators, such as FEP, CEP and LEP. The only function on which SSEP has worst results is f_5. This may be because the principles used are either over-fitting or under-fitting and is discussed in future work.

Table 2 Settings of T for $f_1 - f_{10}$.

f_n	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
T	$1.5e+2$	$1.0e+2$	$1.5e+2$	$1.0e+2$	$1.5e+2$	$1.5e+2$	$1.5e+2$	$1.5e+2$	$1.5e+2$	$1.5e+2$

Table 3 The 10 functions used in our experimental studies, where n is the dimensional size of the function, f_{min} is the minimum value of the function, and $S \subseteq \mathbb{R}^n$.

Function	n	S	f_{min}
$f_1(x) = \sum_{i=1}^{n} x_i^2$	30	$[-100, 100]^n$	0
$f_2(x) = \sum_{i=1}^{n} \mid x_i \mid + \prod_{i=1}^{n} \mid x_i \mid$	30	$[-10, 10]^n$	0
$f_3(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	30	$[-100, 100]^n$	0
$f_4(x) = \max_i \{\mid x_i \mid, 1 \leq i \leq n\}$	30	$[-100, 100]^n$	0
$f_5(x) = \sum_{i=1}^{n} [(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	$[-30, 30]^n$	0
$f_6(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$	30	$[-100, 100]^n$	0
$f_7(x) = \sum_{i=1}^{n} i x_i^4 + random[0, 1)$	30	$[-1.28, 1.28]^n$	0
$f_8(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\mid x_i \mid})$	30	$[-500, 500]^n$	-12569.5
$f_9(x) = \sum_{i=1}^{n} [x_i^2 - 10cos(2\pi x_i) + 10]$	30	$[-5.12, 5.12]^n$	0
$f_{10}(x) = -20 \exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n} \cos 2\pi x_i) + 20 + e$	30	$[-32, 32]^n$	0

Table 4 The results of SSEP, FEP, CEP and LEP with α=(1.2,1.4,1.6,1.8) on f_1-f_{10}, "Mean" indicates the mean best fitness values found in the last generation over 50 runs, "Best" indicates the best fitness value in last generation in 50 runs, "Std Dev" stands for standard deviations.

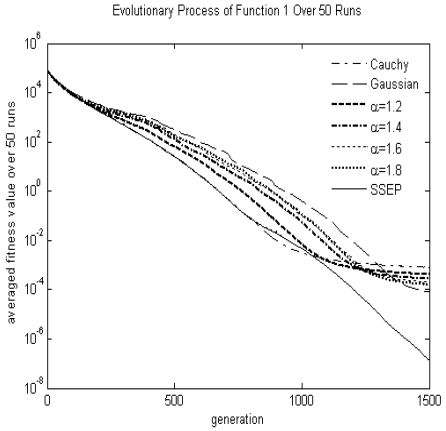
f_n		SSEP	FEP	CEP	$\alpha = 1.2$	$\alpha = 1.4$	$\alpha = 1.6$	$\alpha = 1.8$
f_1	Mean	**1.1679E-07**	4.8402E-04	5.8365E-05	2.5452E-04	1.7666E-04	1.3584E-04	1.0588E-04
	Best	**6.9710E-09**	3.0336E-04	2.7631E-05	1.4416E-04	8.5229E-05	8.1850E-05	7.6026E-05
	Std Dev	(2.15E-07)	(7.27E-05)	(2.36E-05)	(4.49E-05)	(3.66E-05)	(2.83E-05)	(1.70E-05)
f_2	Mean	**2.4397E-04**	6.8787E-02	2.2292E-02	5.2286E-02	4.2083E-02	3.6502E-02	3.3497E-02
	Best	**2.0062E-04**	5.5150E-02	1.9973E-02	4.5706E-02	3.4859E-02	3.1018E-02	2.8360E-02
	Std Dev	(1.67E-05)	(5.61E-03)	(1.28E-03)	(3.15E-03)	(2.84E-03)	(2.50E-03)	(2.03E-03)
f_3	Mean	7.9868E-03	1.2047E-02	7.4353E-03	6.1633E-03	5.9409E-03	**4.2044E-03**	5.4785E-03
	Best	2.4046E-03	2.9402E-03	**1.3768E-04**	1.1369E-03	6.3364E-04	4.1240E-04	4.0410E-04
	Std Dev	(6.50E-03)	(1.10E-02)	(1.14E-02)	(5.06E-03)	(1.44E-02)	(5.42E-03)	(1.00E-02)
f_4	Mean	**8.8458E-04**	6.6422E-03	1.2984	5.9469E-03	2.97E-02	2.4820E-01	5.5215E-01
	Best	**2.0549E-05**	5.0559E-03	1.8176E-01	4.4390E-03	3.7107E-03	3.4234E-03	3.6725E-03
	Std Dev	(1.99E-03)	(6.73E-04)	(9.67E-01)	(1.34E-03)	(1.06E-01)	(5.29E-01)	(6.86E-01)
f_5	Mean	32.0200	25.8343	5.9900	15.2293	9.0767	**5.5955**	7.5573
	Best	0.1599	0.0431	0.0787	0.0416	0.0578	**0.0203**	0.0305
	Std Dev	(34.10)	(28.96)	(11.90)	(23.72)	(11.82)	(5.30)	(11.66)
f_6	Mean	**0**	**0**	119.72	**0**	**0**	0.08	26.28
	Best	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	Std Dev	(0)	(0)	(402.51)	(0)	(0)	(0.27)	(154.00)
f_7	Mean	**7.5877E-03**	8.6825E-03	1.9285E-02	8.4072E-03	1.0126E-02	1.2125E-02	1.3827E-02
	Best	0.0040	0.0039	0.0095	**0.0029**	0.0047	0.0052	0.0048
	Std Dev	(2.11E-03)	(2.27E-03)	(6.14E-03)	(2.56E-03)	(2.80E-03)	(4.08E-03)	(4.66E-03)
f_8	Mean	-11087.73	**-11363.42**	-7974.68	-10553.20	-10261.56	-9522.00	-8807.46
	Best	**-12095.73**	**-12095.73**	-9330.51	-11500.51	-11049.53	-10832.39	-10215.79
	Std Dev	(376.22)	(344.28)	(640.90)	(367.01)	(486.21)	(579.09)	(633.00)
f_9	Mean	47.73	**3.8752E-02**	79.40	4.7913E-02	2.12	20.64	67.03
	Best	0.0329	0.0413	60.6957	**0.0297**	2.0129	20.9487	66.6733
	Std Dev	(28.09)	(5.34E-03)	(22.79)	(0.14)	(1.35)	(8.89)	(20.21)
f_{10}	Mean	0.1661	0.0160	8.1118	0.0118	**0.0113**	1.1340	5.0703
	Best	**5.4074E-05**	1.2613E-02	9.3137E-01	8.4018E-03	8.0068E-03	7.7945E-03	5.7543E-03
	Std Dev	(5.80E-01)	(1.45E-03)	(3.29)	(1.32E-03)	(3.30E-03)	(1.29)	(3.65)

Table 5 2-tailed Wilcoxon Signed-Rank Test comparing SSEP with FEP, CEP and LEP with $\alpha = (1.2, 1.4, 1.6, 1.8)$ on f_1-f_{10}.

f_n	Number of Generations	vs FEP	vs CEP	vs $\alpha = 1.2$	vs $\alpha = 1.4$	vs $\alpha = 1.6$	vs $\alpha = 1.8$
f_1	1500	$>$	$>$	$>$	$>$	$>$	$>$
f_2	2000	$>$	$>$	$>$	$>$	$>$	$>$
f_3	5000	$>$	$\leq$	$\leq$	$<$	$<$	$<$
f_4	5000	$>$	$>$	$>$	$>$	$>$	$>$
f_5	20000	$\leq$	$<$	$<$	$<$	$<$	$<$
f_6	1500	$=$	$>$	$=$	$=$	$=$	$>$
f_7	3000	$>$	$>$	$\geq$	$>$	$>$	$>$
f_8	9000	$<$	$>$	$>$	$>$	$>$	$>$
f_9	5000	$<$	$>$	$<$	$<$	$<$	$>$
f_{10}	1500	$<$	$>$	$<$	$<$	$>$	$>$

Table 6 Comparison of SSEP, MSEP and LMSEP, means and (standard deviations

f_n	Number of Generations	SSEP	LMSEP [8]	MSEP [8]
f_1	1500	**1.1679E-07** (2.15E-07)	3.803E-05 (7.97E-05)	6.209E-05 (1.61E-04)
f_2	2000	**2.4397E-04** (1.67E-05)	1.556E-03 (9.37E-04)	8.226E-02 (4.35E-01)
f_4	5000	**8.8458E-04** (1.99E-03)	0.767 (1.09)	0.629 (1)
f_6	1500	0 (0)	**0** (0)	43.8 (126)
f_7	3000	**7.5877E-03** (2.11E-03)	3.514E-02 (1.85E-02)	3.56E-02 (1.74E-02)
f_9	5000	**47.73** (28.09)	61.39 (13.18)	63.44 (13.8)
f_{10}	1500	1.661E-01 (5.80E-01)	**1.956E-03** (1.76E-03)	6.498 (2.49)



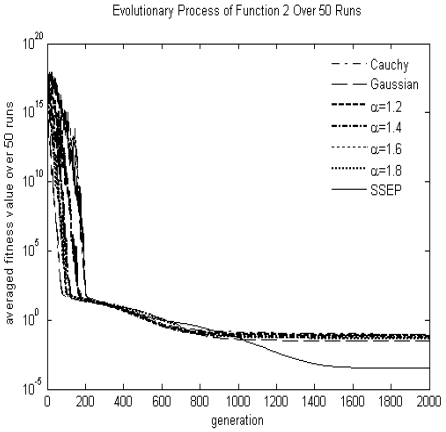
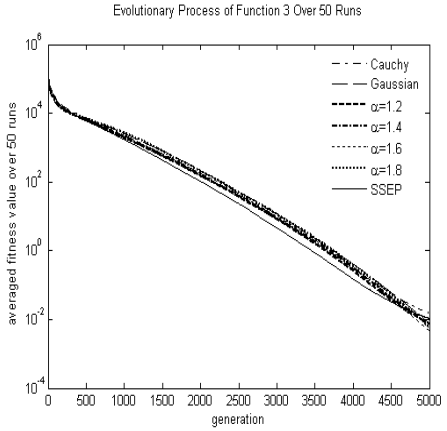


Figure 1 Evolutionary Process of $f1, f2$.



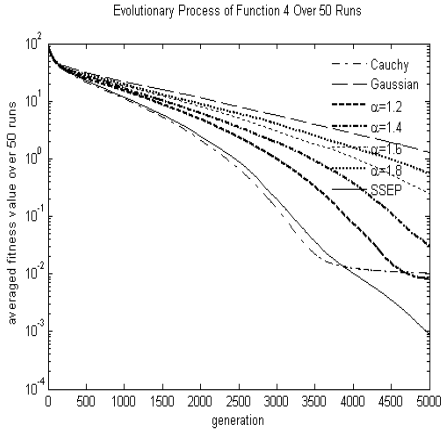


Figure 2 Evolutionary Process of $f3, f4$.

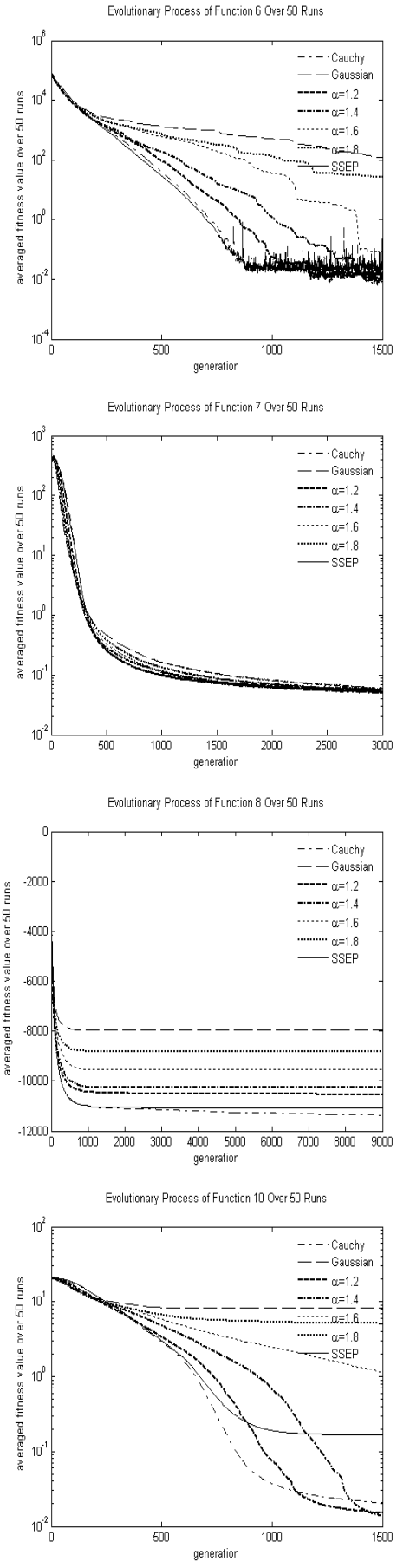


Figure 3 Evolutionary Process of $f6, f7, f8, f10$.

In Figure 1, 2 and 3 , we plot the averaged fitness value over 50 runs at each generation. SSEP shows its consistent convergence rate throughout evolutionary process on f_1, f_2 and f_4.

5.2 Comparison of EP Use None Static Mutation Operator

We list the experimental data of SSEP, MSEP and LMSEP in Table 6. Both MSEP and LMSEP do not use a static mutation operator strategy. SSEP offers better performance on f_1, f_2, f_4, f_7 and f_8.

Note: Shen et al. [8] does not provide results for f_3, f_5 and f_8, so it is only possible to make comparisons on f_1, f_2, f_4, f_6, f_7, f_9 and f_{10}.

6. DISCUSSION AND FUTURE WORK

The main aim of this paper is to implement an algorithm which is capable of selecting a mutation operator for EP at each generation.

We have tested SSEP, using single step size S_k at each generation, history step size S_H_k from generation 1 to current generation k as a self-adaptive factor to pick an appropriate mutation operator for the next generation of EP. The distance coefficient T is used to control SSEP, trying to avoid falling into local optimum in early generations of EP.

So far we have compared it with the static mutation operators used in CEP [10], FEP [10] and LEP [4]. We also compared SSEP to MSEP [2] and LMSEP [8], which both use multiple mutation operators. Several principles are proposed to implement SSEP. Later work will focus on improving the principles in order to fit more functions, hopefully improving the accuracy of the self-adaptive mutation operator.

Another extension in SSEP is to examine how the distance coefficient T affects the convergence speed on different functions. Search space is not considered in the current principles. We believe that the search space of functions is another important factor of self-adaptive mutation operators worthy of future investigation. Currently SSEP works on the same dimensional size for all functions, we are also planning to investigate how SSEP works on functions with different dimensional sizes.

SSEP does not show good performance on some functions, especially f_5, this suggests that the principles are not robust enough. Regardless of this dilemma we have proposed principles and designed an algorithm which outperforms existing EP systems with static mutation operators and dynamic mutation operators on most of the functions tested.

One possible criticism of this method is that the principles do not fit all of the functions tested so far. This is because some important factors that affect the selection of mutation operators is omitted. In future, we will collect as much useful information as possible throughout the evolutionary process to make SSEP more accurate and robust.

7. SUMMARY AND CONCLUSIONS

Evolutionary Programming is a robust method of solving numerical optimization problems. In the past it has involved using probability distributions (Gaussian, Cauchy and Lévy, etc) as the unique mutation operator. Recently multiple mutation operators have been adopted in EP methods, such as IFEP [11], MSEP [2] and LMSEP [8]. EP with a static mu-

tation operator usually keeps high convergence rate in early generations, but less offspring survive in tournament selection in later generations. In this paper, we propose principles to keep the high convergence rate not only in early generations of EP, but also in later generations. According to the principles, an algorithm is presented and injected into EP, to help EP adaptively select appropriate mutation operators at each generation throughout the evolutionary process.

SSEP is inspired by the process of tuning an analogue car radio, when a driver usually tunes the radio in a wide range to search for a channel, before slightly adjusting the radio to get a clear signal. SSEP moves in the search space with a long step size mutation operator at beginning and uses a short step size mutation operator towards the end of the search.

From the experimental results, SSEP shows good performance on a set of 10 benchmark functions. The initial results are highly encouraging, we cannot claim that our principles can fit over all functions from the literature, but our principles are successful on some functions, outperforming existing EP methods.

8. REFERENCES

[1] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1:1–23, 1993.

[2] H. Dong, J. He, H. Huang, and W. Hou. Evolutionary programming using a mixed mutation strategy. *Information Science*, 177:312–327, 2007.

[3] L. Hong, J. Woodward, J. Li, and E. Özcan. Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming. In *Genetic Programming*, volume 7831 of *Lecture Notes in Computer Science*, pages 85–96. Springer Berlin Heidelberg, 2013.

[4] C.-Y. Lee and X. Yao. Evolutionary programming using mutations based on the lévy probability distribution. *IEEE Transactions on Evolutionary Computation*, 8:1–13, 2004.

[5] K.-H. Liang, X. Yao, Y. Liu, C. Newton, and D. Hoffman. An experimental investigation of self-adaptation in evolutionary programming. In V. Porto, N. Saravanan, D. Waagen, and A. Eiben, editors, *Evolutionary Programming VII*, volume 1447 of *Lecture Notes in Computer Science*, pages 291–300. Springer Berlin Heidelberg, 1998.

[6] K.-H. Liang, X. Yao, and C. S. Newton. Adapting self-adaptive parameters in evolutionary algorithms. *Applied Intelligence*, 15(3):171–180, 2001.

[7] P. N. S. Rammohan Mallipeddi, S. Mallipeddi. Ensemble strategies with adaptive evolutionary programming. *Information Science*, 180:1571–1581, 2010.

[8] L. Shen and J. He. A mixed strategy for evolutionary programming based on local fitness landscape. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8, July 2010.

[9] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[10] X. Yao and Y. Liu. Fast evolutionary programming. *Proceedings of the Fifth Annual Conference on Evolutionary Programming, MIT Press*, pages 451–460, 1996.

[11] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3:82–102, 1999.