Using Genetic Programming to Learn Probability Distributions as Mutation Operators with Evolutionary Programming

Libin Hong, John Woodward, Jingpeng Li, Ender Özcan

Abstract. The mutation operator is the only source of variation in Evolutionary Programming. In the past these have been human nominated and have included the Gaussian distribution in Classical Evolutionary Programming, the Cauchy distribution in Fast Evolutionary Programming, and the Lévy distribution. In this paper, we automatically design the mutation operators (probability distributions) using Genetic Programming. This is done by using random number generators (uniformly distributed in the range 0 to 1) and using them as terminals in Genetic Programming. In other words, a random number generator is a function of a uniformly generated random number passed through a function generated by Genetic Programming. Rather than attempting to develop mutation operator for arbitrary benchmark functions drawn from the research literature, we have taken existing benchmark functions and included them in a function class, which is a probability distribution over a set of problem instances (functions). The mutation probability distribution is trained on a set of problems instances, and then tested on a second independent test set of instances to confirm that the evolved probability distribution has indeed generalized to the problem class. Initial results are highly encouraging; on each of the problem classes the probability distributions generated using Genetic Programming outperforms both the Gaussian and Cauchy distributions.

Keywords: Evolutionary Programming, Genetic Programming, Evolutionary Computation, Function Optimization, Machine Learning, Metalearning, Hyper-heuristics.

1 Introduction

1.1 Evolutionary Programming

Evolutionary programming (EP) is one of the branches of Evolutionary Computation (EC) and is used to evolve numerical values in order to find a global optimum of a function.

The main genetic operator is mutation. The probability distributions used as mutation operators include: Gaussian, Cauchy and Lévy, among others. Classical EP (CEP) [3] uses a Gaussian distribution as mutation operator. Fast EP (FEP)

[3] uses a Cauchy distribution as mutation operator. A Lévy distribution (LEP) has also been used as a mutation operator [4].

In recent years, many improvements of EP have been proposed. Improved FEP (IFEP) [5], mixes mutation operators, and uses both Gaussian and Cauchy distributions. Later mixed mutation strategy (MSEP) [8] was proposed; four mutation operators are used and the mutation operator is selected according to its probability in each generation during the evolution.

This paper proposes a novel method to generate new mutation operators to promote the convergence speed of EP. That is using genetic programming (GP) to train EP's mutation operators, and then use the new distribution as new mutation operator for EP on functions similar (i.e. drawn from the same problem class) to functions in the training set.

1.2 Problem Instances, Problem Classes and Meta-learning

In previous work on function optimization, typically an algorithm is applied to a *single* function to be optimized. As the algorithm is applied, it learns better values for its best-so-far value. We regard a problem instance as a single function to be optimized which is drawn from a probability distribution over functions, which we call a function class. In this paper we are employing a meta-learning approach consisting of a base-level and meta-level; EP sitting at the base-level, learning about the specific function, and GP sitting at the meta-level which is applied across problem instances, learning about the function class as a whole. By taking this approach we can say that one mutation operator developed by GP on one function class is suitable for problem instances drawn from that class, while another mutation operator is more suited to problem instances drawn from a different problem class.

1.3 Recent Research

In 1992 and 1993, David B. Fogel and Bäck et al [1, 2] indicated that CEP with adaptive mutation usually performs better than CEP without adaptive mutation. In 1996, a new mutation operator, the Cauchy distribution was proposed to replace the Gaussian distribution which is used in classical evolutionary programming (CEP) with adaptive mutation, in order to distinguish CEP, EP uses Cauchy mutation and is called FEP, to compare CEP and FEP, the authors have done the experiment which followed Bäck et al's algorithm [3].

In 1999, Improved FEP (IFEP) was proposed, it mixes both Cauchy and Gaussian mutations in EP [5].

In 2004, EP that uses Lévy probability distribution $L_{\alpha,\gamma}(y)$ as mutation operator was proposed [4]. According to their experimental results, they obtained the following conclusion: Lévy mutation can lead to a large variation and a large number of distinct values in evolutionary search, in comparison with traditional Gaussian mutation [4].

Ensemble strategies with adaptive EP (ESAEP), evaluation of novel adaptive EP on four constraint handling techniques, EP using a mixed mutation strategy were proposed [6–8]. However, all these technologies are focus on the improvement of EP itself, they are not trying to find new mutation operators to replace the exists mutation operator like Gaussian, Cauchy or Lévy distribution.

1.4 Genetic Programming

GP is considered a specialization of GAs where each individual is a computer program [9]. GP can be succinctly described as a GA wherein the population contains programs rather than bit strings [10]. GP automatically generates computer programs to solve specified problems.

It develops programs through the process of a "create-test-modify" cycle which is similar to the way a human writes programs.

1.5 Function Optimization by Evolutionary Programming

In section 2 we describe function optimization. Section2 describes how EP is applied to the task of finding a probability distribution which can be used as a mutation operator in EP to find the global minimum of a function.

Todo

2 Function Optimization by Evolutionary Programming

A global minimization problem can be formalized as a pair (S, f), where $S \in \mathbb{R}^n$ is a bounded set on $\mathbb{R}^n$ and $f : S \longrightarrow \mathbb{R}$ is an n-dimensional real-valued function. The problem 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.

According to the description by Bäck et al [3], the EP is implemented as follows:

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, and k=1, $\cdots$, λ.*
$$x_i'(j) = x_i(j) + \eta_i(j)D_j \tag{1}$$
$$\eta'(j) = \eta_i(j)exp(\gamma' N(0,1) + \gamma N_j(0,1)) \tag{2}$$
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. *Step4. 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 the opponents's, it receives a "win".*
6. *Pick up the* μ *individuals out of parents and offspring,* $i \in \{1, \cdot \cdot \cdot, \mu\}$, *that have the most wins to be parents, as the next generation.*
7. *Stop if the stopping criterion is satisfied; otherwise,* $k = k + 1$ *and go to Step3.*

If D_j in Eq.(1) uses the Gaussian distribution, it is called CEP. If D_j in Eq.(1) use Cauchy distribution, it is FEP. If D_j in Eq.(1) use Lévy distribution, it is LEP.

3 Using Genetic Programming to Train Evolutionary Programming

In this section, we give the details of how and why we use GP to train a EP mutation operator.

In the past, candidate distributions have been nominated by humans and tested on a set of benchmark instances. Here we are automating this process using GP to generate and test the distributions.

The research question we are addressing in this paper is the following; is it possible for GP to automatically generate mutation operators (i.e. probability distributions) which can be used in EP as the mutation operator and outperform the human generate distributions.

3.1 Problem Classes

In previous work, researchers use particular functions as a benchmark to test the performance of particular mutation operator. Our work differs in this respect. We define a set of problem classes from which each of those functions are drawn from. In this way, we can train an EP mutation operator which is tuned to that function class. It would not make sense to apply an EP algorithm to arbitrary functions.

For example: when $a = 1$, $f(x) = \sum_{i=1}^{n} x_i^2$ is an instance of problem class $f(x) = a\sum_{i=1}^{n} x_i^2$ where $a \in [1,\ 2]$. GP generates mutation operators for EP, when GP processing, the fitness value is calculated by EP, we random generated a from range $[1, 2]$ for EP.

In this paper, as an example of a function class, we define $a\sum_{i=1}^{n} x_i^2$. The motivation for defining a function class like this is that a is in the range [1,2] and so we are evolving a distribution which is fit-for-purpose on functions from this class.

Below is the pseudo-code of the algorithm:

```
1: i = 0  /* Set generation counter*/
2: Initialize terminal set  /* Set terminal set for GP*/
3: Initialize functional set /* Set functional set for GP*/
4: Initialize population Q(i) /* Generate probability distributions*/
5: Evaluate population Q(i) /* Compute fitness values by EP*/
6: while not reach the max generation do
7: i = i + 1
8: Select Q(i) from Q(i - 1)
9: Crossover Q(i) /* Crossover in GP*/
10: Mutate Q(i) /* Mutation in GP*/
11: Evaluate Q(i) /* Compute fitness values by EP*/
12: end while
```

In step 2, we set $N(\mu, \sigma^2)$ in terminal set, when $\mu = 0$, $\sigma = 1$, it is standard normal distribution, in our test the value of $\mu = 0$ and the value of σ is in range $[0, 5]$. In step 3, we assign the function set as $\{+, -, \times, \div\}$. In step 11, we use EP as fitness function to evaluate the fitness value. In step 11 we evaluate the utility of the distribution, and assign it the best fitness of each EP run, averaged over the 20 EP runs. When evaluating the fitness value, EP will run 20 times and calculate the mean value in last generation as fitness value for GP.

After training a new distribution will survived, this distribution find by GP is called GP-Dist. The function classes is in table 1.

Table 1: The 10 function classes used in our experimental studies, where n is the dimension of the function, f_{min} is the minimum value of the function, and $S \subseteq \mathbb{R}^n$, n is 30, a and b are random number from the specified range.

Function Classes	S	a	b	f_{min}
$f_1(x) = a\sum_{i=1}^n x_i^2$	$[-100, 100]^n$	$a \in$ [1, 2]	N/A	0
$f_2(x) = a\sum_{i=1}^n \mid x_i \mid + b\prod_{i=1}^n \mid x_i \mid$	$[-10, 10]^n$	$a \in$ [1, 2]	$b \in [0, 10^{-5}]$	0
$f_3(x) = \sum_{i=1}^n (a\sum_{j=1}^i x_j)^2$	$[-100, 100]^n$	$a \in$ [1, 2]	N/A	0
$f_4(x) = max_i\{a \mid x_i \mid, 1 \le i \le n\}$	$[-100, 100]^n$	$a \in$ [1, 2]	N/A	0
$f_5(x) = \sum_{i=1}^n [a(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-30, 30]^n$	$a \in$ [1, 2]	N/A	0
$f_6(x) = \sum_{i=1}^n (\lfloor ax_i + 0.5 \rfloor)^2$	$[-100, 100]^n$	$a \in$ [1, 2]	N/A	0
$f_7(x) = a\sum_{i=1}^n ix_i^4 + random[0, 1)$	$[-1.28, 1.28]^n$	$a \in$ [1, 2]	N/A	0
$f_8(x) = \sum_{i=1}^n -(x_i sin(\sqrt{\mid x_i \mid}) + a)$	$[-500, 500]^n$	$a \in$ [1, 2]	N/A	[-12629.5, -12599.5]
$f_9(x) = \sum_{i=1}^n [ax_i^2 + b(1 - cos(2\pi x_i))]$	$[-5.12, 5.12]^n$	$a \in$ [1, 2]	$b \in$ [5, 10]	0
$f_{10}(x) = -aexp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^n x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^n cos 2\pi x_i) + a + e$	$[-32, 32]^n$	$a \in$ [1, 2]	N/A	0

Table 2: Comparison among GP-Dist, FEP and CEP on f_1-f_{10}. All results have been averaged over 50 runs, where “Mean Best” indicates the mean best function values found in the last generation, and “Std Dev” stands for the standard deviation.

Function	FEP		CEP		GP-Dist	
Classes	*Mean Best*	*Std Dev*	*Mean Best*	*Std Dev*	*Mean Best*	*Std Dev*
f_1	1.24×10^{-3}	2.69×10^{-4}	1.45×10^{-4}	9.95×10^{-5}	6.37×10^{-5}	5.56×10^{-5}
f_2	1.53×10^{-1}	2.72×10^{-2}	4.30×10^{-2}	9.08×10^{-3}	8.14×10^{-4}	8.50×10^{-4}
f_3	2.74×10^{-2}	2.43×10^{-2}	5.15×10^{-2}	9.52×10^{-2}	6.14×10^{-3}	8.78×10^{-3}
f_4	1.79	1.84	1.75×10	6.10	2.16×10^{-1}	6.54×10^{-1}
f_5	2.52×10^{-3}	4.96×10^{-4}	2.66×10^{-4}	4.65×10^{-5}	8.39×10^{-7}	1.43×10^{-7}
f_6	3.86×10^{-2}	3.12×10^{-2}	4.40×10	1.42×10^{2}	9.20×10^{-3}	1.34×10^{-2}
f_7	6.49×10^{-2}	1.04×10^{-2}	6.64×10^{-2}	1.21×10^{-2}	5.25×10^{-2}	8.46×10^{-3}
f_8	TBD	TBD	TBD	TBD	TBD	TBD
f_9	6.24×10^{-2}	1.30×10^{-2}	1.09×10^{2}	3.58×10	1.74×10^{-3}	4.25×10^{-4}
f_{10}	1.67	4.26×10^{-1}	1.45	2.77×10^{-1}	1.38	2.45×10^{-1}

Table 3: t-tests comparing GP-Dist, with FEP and CEP on f_1-f_{10}

Function	Number of	GP-Dist vs FEP	GP-Dist vs CEP
Classes	Generations	*t-test*	*t-test*
f_1	1500	2.78×10^{-47}	4.07×10^{-2}
f_2	2000	5.53×10^{-62}	1.59×10^{-54}
f_3	5000	8.03×10^{-8}	1.14×10^{-3}
f_4	5000	1.28×10^{-7}	3.73×10^{-36}
f_5	20000	2.80×10^{-58}	9.29×10^{-63}
f_6	1500	1.85×10^{-8}	3.11×10^{-2}
f_7	3000	3.27×10^{-9}	2.00×10^{-9}
f_8	9000	TBD	TBD
f_9	5000	6.37×10^{-55}	6.54×10^{-39}
f_{10}	1500	9.23×10^{-5}	1.93×10^{-1}

4 Test Function Classes

In table 2, we list out the results for all 10 function classes. From the results, GP-Dist outperforms both Cauchy and Gaussian on all function classes, and GP-Dist has significant improvement on $f1, f2, f3, f4, f5, f6, f9$. The finally result of $f1$ is promoted from level 10^{-3} (made by FEP) and 10^{-4} (made by CEP) to 10^{-5}, $f2$ is promoted from level 10^{-1} and 10^{-2} to 10^{-4}, $f3$ is promoted from level 10^{-2} and 10^{-2} to 10^{-3}, $f4$ is promoted from level 10^{0} and 10^{1} to 10^{-1},$f5$ is promoted from level 10^{-3} and 10^{-4} to 10^{-7}, $f6$ is promoted from level 10^{-2} and 10^{1} to 10^{-3}, $f9$ is promoted from level 10^{-2} and 10^{2} to 10^{-3}.

5 Experimental Studies

In previous work, most of the authors have tested their algorithms on a benchmark suit of 23 function instances.

In this paper, we use our method test 10 of them.

In our test, we run EP 20 times, and use the mean value of all 20 runs as fitness value for GP. If a mutation operator, we call it GP-Dist in later part of this paper, find by GP has good performance on a problem class, it should have good performance on particular instance of that class as well.

the new methods we proposed has successfully found a new mutation operator for each function class. All the mutation operators found beat both Cauchy and Gaussian mutation operator. The only function on which good results were not found was f_{10}, but this may be because GP was either over-fitting or under-fitting.

5.1 Analysis and Comparisons

Below is a GP-Dist finally survived in GP for function class 1:

$$(\div(*(-(\div(-\ 0\ N(\mu,\sigma^2))(\div\ N(\mu,\sigma^2)\ 0))(-\ (\div\ 0\ 0)\ 0))\ N(\mu,\sigma^2))(\div\ 0\ 0))$$

Here σ is 0.171281, σ is randomly generated by GP in range $[0,5]$.

To compare the difference among GP-Dists, Gaussian and Cauchy, we plot the distribution in below graph. For each distribution we plot it for 3000 samples.

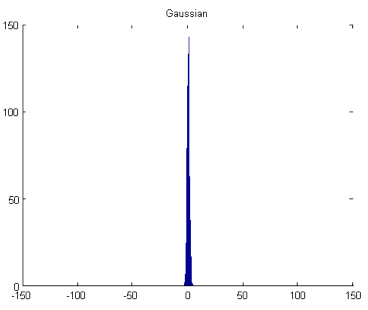

Fig. 1. A histogram of the Gaussian Distribution for 3000 samples.

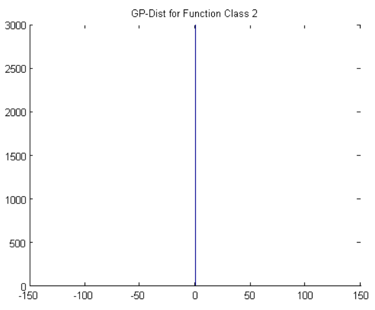

Fig. 4. A histogram of the GP generated distribution for Function Class 2 for 3000 samples.

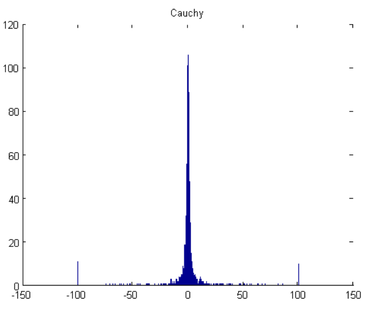

Fig. 2. A histogram of the Cauchy Distribution for 3000 samples.

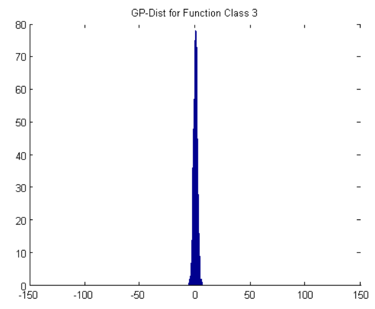

Fig. 5. A histogram of the GP generated distribution for Function Class 3 for 3000 samples.

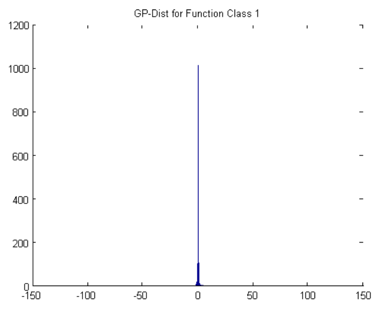

Fig. 3. A histogram of the GP generated distribution for Function Class 1 for 3000 samples.

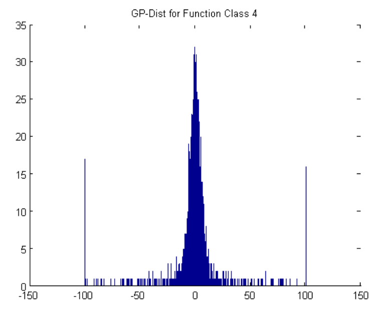

Fig. 6. A histogram of the GP generated distribution for Function Class 4 for 3000 samples.

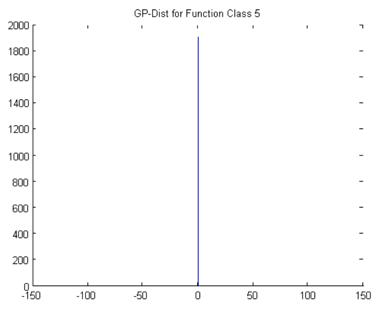

Fig. 7. A histogram of the GP generated distribution for Function Class 5 for 3000 samples.

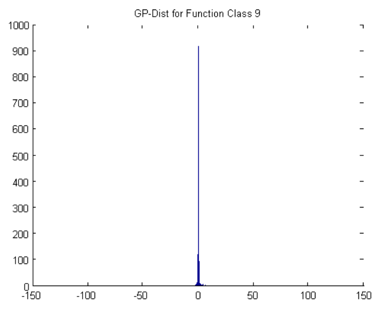

Fig. 10. A histogram of the GP generated distribution for Function Class 9 for 3000 samples.

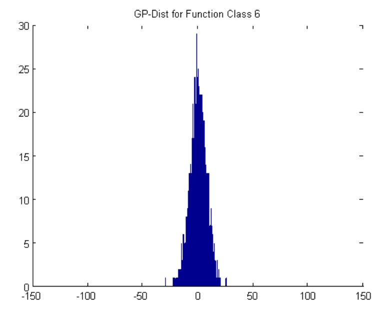

Fig. 8. A histogram of the GP generated distribution for Function Class 6 for 3000 samples.

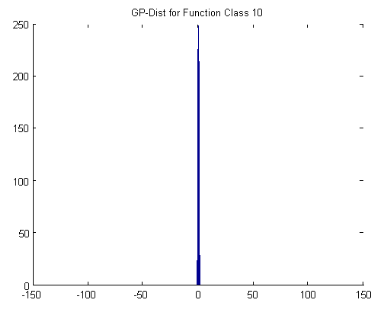

Fig. 11. A histogram of the GP generated distribution for Function Class 10 for 3000 samples.

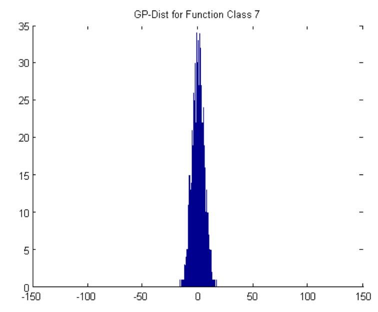

Fig. 9. A histogram of the GP generated distribution for Function Class 7 for 3000 samples.

5.2 Unimodal Function Classes and Multimodal Function Classes

In the function class suite, $f1$-$f7$ are unimodal function classes, $f8$-$f10$ are multimodal function classes, the experimental results prove that our method is well applied on both unimodal and multimodal functions classes.

6 Future Work

The initial aim of this paper is to build a system which is capable for synthesizing distributions and we therefore compare it with only FEP and CEP. Later work will address comparisons with more recent developments in EP including IFEP [5] and MSEP [8].

In this paper we have run the GP system for a fixed number of iterations, and have not optimized these parameters, so there is further scope for improvement of results in this regard. Further work includes using more sophisticated methods of terminating the search for a probability distributions, such as early stopping to prevent either under-fitting or over-fitting.

We have defined functions classes in terms of a random variable which forms a coefficient in the function. This provides a source of related functions to be optimized. Each of these function classes (see table 1) are either unimodal or multimodal functions. None of the currently defined function classes contain both, so it would be interesting to evolve a distribution capable of performing well on both types of function instances.

In future, we would like to pursue more interesting research topics. In previous work, researchers are usually uses 23 functions to test the performance of EP. We have already extended 10 of them as function classes, and will extend another 13 functions to test the performance of the methods we propose in this paper.

7 Conclusion

A GP-EP algorithm is proposed in this paper. When GP is running, EP is working as fitness function to test the performance of mutation operator generated by GP. The mutation operator survived in GP is called GP-Dist. GP-Dist outperforms both Cauchy and Gaussian mutation operator on the function classes in the testing phase. Our experimental results indicates that GP-Dist for each function classes converges to a better near optimal solution much faster than CEP and FEP for both single-modal and multi-modal function classes.

References

1. David. B. Fogel, "Evolving Artificial Intelligence", PhD thesis, University of California, San Diego, 1992.
2. T. Bick and H.-P.Schwefel, "An overview of evolutionary algorithms for parameter optimization" Evolutionary Computation, 1(1):1-23, 1993.

3. Xin Yao and Yong Liu, “Fast Evolutionary Programming”, Proceedings of the Fifth Annual Conference on Evolutionary Programming, MIT Press, P451-P460,1996.
4. Chang-Yong Lee and Xin Yao, “Evolutionary Programming Using Mutations Based on the Lévy Probability Distribution”, IEEE Transactions on Evolutionary Computation, Vol. 8, NO. 1, 2004.
5. Xin Yao, Yong Liu and Guangming Lin, “Evolutionary Programming Made Faster”, IEEE Transactions on Evolutionary Computation, Vol. 3, 82-102, 1999.
6. R. Mallipeddi, S. Mallipeddi, P.N. Suganthan, “Ensemble strategies with adaptive evolutionary programming”, Information Science, 1571-1581, 2010.
7. R. Mallipeddi, P.N. Suganthan, “Evaluation of novel adaptive evolutionary programming on four constraint handling techniques”, IEEE Congress on Evolutionary Computation, 4045-4052, 2008.
8. H. Dong, J. Heet .al, “Evolutionary Programming Using A Mixed Mutation Strategy”, Information Science, 312-327, 2007.
9. Poli, Langdon, and McPhee, “A Field Guide to Genetic Programming”, ISBN 978-1-4092-0073-4, 2008.
10. Moshe Sipper, “Evolved to Win”, Lulu, ISBN: 978-1-4709-7283-7, 2011
11. Gwoing Tina Yu, “An Analysis of the Impact of Functional Programming Techniques on Genetic Programming”, PhD thesis, University College, London, Gower Street, London, WC1E 6BT, 1999.