Automated Design of Probability Distributions as Mutation Operators for Evolutionary Programming Using Genetic Programming

Libin Hong[1], John Woodward[1], Jingpeng Li[1], and Ender Özcan[2]

[1] Department of Computer Science, University of Nottingham P.R.C.
[2] Department of Computer Science, University of Nottingham U.K.
{Libin.HONG,John.WOODWARD,Jingpeng.LI}@nottingham.edu.cn
Ender.Ozcan@nottingham.ac.uk

Abstract. The mutation operator is the only source of variation in Evolutionary Programming. In the past these have been human nominated and included the Gaussian, Cauchy, and the Lévy distributions. We automatically design mutation operators (probability distributions) using Genetic Programming. This is done by using a standard Gaussian random number generator as the terminal set and and basic arithmetic operators as the function set. In other words, an arbitrary random number generator is a function of a randomly (Gaussian) generated number passed through an arbitrary function generated by Genetic Programming.

Rather than engaging in the futile attempt to develop mutation operators for arbitrary benchmark functions (which is a consequence of the No Free Lunch theorems), we consider tailoring mutation operators for particular function classes. We draw functions from a function class (a probability distribution over a set of functions). The mutation probability distribution is trained on a set of function instances drawn from a given function class. It is then tested on a separate independent test set of function instances to confirm that the evolved probability distribution has indeed generalized to the function class.

Initial results are highly encouraging: on each of the ten function classes the probability distributions generated using Genetic Programming outperform both the Gaussian and Cauchy distributions.

Keywords: Evolutionary Programming, Genetic Programming, Function Optimization, Machine Learning, Meta-learning, Hyper-heuristics, Automatic Design.

1 Introduction

Evolutionary Programming (EP) is one of the branches of Evolutionary Computation and is used to evolve numerical values in order to find a global optimum of a function. The only genetic operator is mutation. The probability distributions used as mutation operators include Gaussian, Cauchy and Lévy, among others.

In 1992 and 1993, Fogel and Bäck et al. [4][1] indicated that Classical Evolutionary Programming (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. The authors Yao and Yong have done experiments which followed Bäck and Schwefel's algorithm [1]. Fast EP (FEP) [14] uses a Cauchy distribution as mutation operator. The aim of this paper is to develop a Genetic Programming (GP) system which has a function set and terminal set which is capable of (easily) expressing either the Gaussian or Cauchy distribution, and then embracing the search utility of GP to discover more suitable mutation probability distributions.

In recent years, many improvements on EP have been proposed. Improved FEP (IFEP) [13], mixes mutation operators, and uses both Gaussian and Cauchy distributions. Later a mixed mutation strategy (MSEP) [3] was proposed: four mutation operators are used and the mutation operator is selected according to their probabilities during the evolution.

In 2004, EP that uses Lévy probability distribution $L_{\alpha,\gamma}(y)$ as mutation operator was proposed [5]. According to their experimental results, they obtained the following conclusion: Lévy based mutation can lead to a large variation and a large number of distinct values in evolutionary search, in comparison with traditional Gaussian mutation [5]. From 2007, Ensemble strategies with adaptive EP (ESAEP), Novel adaptive EP on four constraint handling techniques, and EP using a mixed mutation strategy were proposed [7][6][3]. Thus research into EP is still very much an active area of research.

This paper proposes a novel method to generate new mutation operators to promote the convergence speed of EP. It applies GP to train EP's mutation operators, and then use the new GP-generated distribution (GP-distribution) as new mutation operator for EP on functions similar (i.e. drawn from the same function class) to functions in the training set, which we now explain.

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 function instance as a single function 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 [9] [10]. EP sits at the base-level, learning about the specific function, and GP sits at the meta-level, which is applied across function 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 function instances drawn from that class, while another mutation operator is more suited to function instances drawn from a different function class. To phrase it differently, a mutation operator developed on one function class will be able to exploit characteristics of functions that are drawn from that function class.

In Section 2 we describe function optimization and the EP algorithm. In Section 3 we describe how GP is applied to the task of finding a probability distribution which can be used as a mutation operator in EP, and define the Function Classes used in this study are also presented. In Section 4 we compare Gaussian, Cauchy and the GP-distributions found by GP, and plot histograms of GP-distributions. We also list the experimental results. In Section 5 we discuss and explain future work. In Section 6 we summarize and conclude the article.

2 Function Optimization by Evolutionary Programming

Global minimization 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 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. According to the description by Bäck et al [1], 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.*

$$x_i{}'(j) = x_i(j) + \eta_i(j) D_j \tag{1}$$

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

   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. *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. If D_j is the Cauchy distribution, it is FEP [14]. If D_j is the Lévy distribution, it is LEP [5]. Thus this algorithm acts as a template into which we can substitute distributions evolved by GP, which is the contribution of this paper.

3 Genetic Programming to Train Mutation Operators for Function Classes

In this section, we give the details of how we use GP to train an EP mutation operator. In the past, candidate distributions have been nominated by humans and tested on a set of benchmark function instances. Here we automate this process by 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 to outperform the human generated distributions? As we have a terminal set containing a Gaussian distribution, it is not surprising that we can evolve a new distribution which can outperform a Gaussian distribution. Nor is it surprising that we can evolve a new distribution which can outperform a Cauchy distribution, as a Cauchy distribution can be generated by dividing a Gaussian distribution by another and can easily be generated by GP containing division in its function set.

At this stage we should also point out that we are doing more than "just" parameter tuning. That is, we are not just altering the numerical parameters (mean and variance) of a Gaussian distribution, but actually generating new distributions which do not belong to the Gaussian distribution.

3.1 Genetic Programming and Automatic Design

GP can be considered a specialization of the more widely known Genetic Algorithms (GAs) where each individual is a computer program [8]. GP automatically generates computer programs to solve specified tasks. It is a method of searching a space of computer programs, and therefore is an automatic way of producing computable probability distributions [8]. Over last few years, the application of GP has become more ambitious, and has been applied to other branches like combinatorial optimization [9][2]. However this new direction is probably due largely to the availability faster machines on which our implementations can be executed, rather than any break through or deep understanding of the search mechanisms of GP. In particular GP can be applied to the task of automated design of components of search algorithms [12] [11] though in these cases random search and iterative hill-climbing were used.

3.2 Function Classes

In the past, researchers use particular functions as a benchmark to test the performance of their algorithms. Our work differs markedly in this respect. We define a set of function classes from which functions are drawn from. In this way, we can train an EP mutation operator and tune it to that function class. It would not make sense to apply an EP algorithm (or any other optimization algorithm for that matter) to arbitrary functions and hope for good performance, a consequence of the No Free Lunch theorems.

As an example of a function class ($a\sum_{i=1}^{n} x_i^2$), where a is a random variable in the range [1, 2], and $f(x) = \sum_{i=1}^{n} x_i^2$ is an instance of a function from this function class (i.e. when $a = 1$). The motivation for defining a function class like this is that we can then evolve a mutation operator which is fit-for-purpose i.e. as a mutation operator on functions drawn from that function class. Evolution is adapting the distribution to fit the environment (function class).

3.3 Algorithm Using GP to Train EP Mutation Operator

Below is the pseudo-code of the training algorithm:

```
1:  Initial gp population
2:  while gpGen < gpMaxGen do
3:    gpPop = 1 /*Set GP iteration*/
4:    while (gpPop < gpMaxPop) do /*Evaluate individuals in GP*/
5:     epIteration = 1  /*Set EP iteration*/
6:     while (epIteration < epMaxIteration) do
7:       Randomly generate a (and b)
8:       Evaluate fitness of pop[gpPop]/*Compute fitness values by EP*/
9:       Set fitness value to fitness[epIteration]
10:      epIteration++
11:    end while
12:    Calculate mean fitness value meanFitness[epMaxIteration]
13:    gpPop++
14:   end while
15:   Select best pop by meanFitness[epMaxIteration]
16:   Crossover pop  /*Crossover pop in GP*/
17:   Mutate pop  /*Mutation pop in GP*/
18: end while
```

The terminal set consists of the Gaussian distribution $N(\mu, \sigma^2)$. We set the value of $\mu = 0$ and the value of σ is randomly assigned from the range $[0, 5]$. The value of μ could be allowed to alter, but it was deemed not necessary in these initial experiments. The value of σ was fixed for a given GP run, but could be allowed to vary between GP programs and within GP programs. We assign the function set as $\{+, -, \times, \div\}$, where $\div$ is protected, if a value a is divided by zero, then the value is a. This simple function set is expressive enough to be able to generate a wide range of functions (and therefore probability distributions). In Step 8, we use EP as fitness function to evaluate the GP-distribution. In Step 9 we assign it the best fitness of each EP run, averaged over the 20 EP runs. When evaluating the fitness value, EP runs 20 times and we calculate the mean value in last generation as fitness value for GP as was done in the original work by Yao [14].

3.4 Unimodal and Multimodal Function Classes

In Table 1, we list all the function classes used in this paper. In the function class suite, f_1-f_7 are unimodal function classes, f_8-f_{10} are multimodal function classes. f_8 is a special case , the f_{min} for function class 8 is not fixed. However for an instance of function class 8, the value of f_{min} is fixed, and depending on the value of a.

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 is random number in the range [1, 2], and b is random number from the specified range or N/A.

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

4 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 the first 10 (see Table 1). This is largely due to the fact that we have to repeatedly run GP to train a mutation operator for each function class. We run EP 20 times with each mutation operator, and use the mean value of all 20 runs as the fitness value for GP. If a mutation operator (GP-distribution) found by GP which has good performance on a function class, it should have good performance on other function instance drawn from that function class.

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 and is discussed in future work.

4.1 Parameters Setting

A different maximum number of generations is used as a termination criterion in EP as provided in section 4.3. Table 2 provides the rest of the parameter values that we used in our approach. We regard the parameters of EP as fixed for this experiment (in the sense we are comparing a method against others for these EP parameter settings). We are not claiming optimality for the GP parameter settings, which are set rather low compared to traditional values, however we did find in these preliminary experiments that these settings were adequate enough to obtain human competitive results.

Table 2. Parameter settings for GP and EP

Parameter Meanings	Settings	Parameter Name in Section 3.3
Max Generation of GP	5	gpMaxGen
Population Size of GP	9	gpMaxPop
Operators of GP	Crossover Mutation	N/A
GP Function Set	$\{+, -, \times, \div\}$	N/A
GP Terminal Set	$N(0,\ [0,5]^2)$	N/A
Number of Iteration of EP	20	epMaxIteration
Population Size of EP	100	N/A
Tournament Size of EP	10	N/A

4.2 Analysis and Comparisons

The best GP-distribution found for each of the ten function classes is listed in Table 3. In our test, μ has a fixed value 0, σ is randomly generated in the range $[0, 5]$. To compare the difference between GP-distributions, Gaussian and Cauchy, we plot all distributions in Fig.1 and Fig.2. For each distribution we plot it for 3000 samples (please note the scale of x-axis).

Table 3. All GP-distributions for function classes

Function Class	Best Distribution Survived in GP (GP-distribution)	Value of σ
$f_1(x)$	$(\div\ (\div\ (-(0\ N(0,\sigma^2)))\ N(0,\sigma^2))\ N(0,\sigma^2))$	0.171281
$f_2(x)$	$N(0,\sigma^2)$	0.010408
$f_3(x)$	$N(0,\sigma^2)$	1.749545
$f_4(x)$	$(+(N(0,\sigma^2)\ (-(\div(\div(+(N(0,\sigma^2)\ N(0,\sigma^2))\ N(0,\sigma^2))\ N(0,\sigma^2))\ N(0,\sigma^2)))))$	2.962383
$f_5(x)$	$N(0,\sigma^2)$	0.056501
$f_6(x)$	$(+(N(0,\sigma^2)\ (-(N(0,\sigma^2)\ N(0,\sigma^2)))))$	3.879682
$f_7(x)$	$N(0,\sigma^2)$	4.851848
$f_8(x)$	$(\div(\div(\times(\times(N(0,\sigma^2) \times (N(0,\sigma^2)N(0,\sigma^2)))\ N(0,\sigma^2))\ N(0,\sigma^2))N(0,\sigma^2)))$	4.918542
$f_9(x)$	$(\div(N(0,\sigma^2)\ (-(\div(N(0,\sigma^2)\ (-(N(0,\sigma^2)\ N(0,\sigma^2))))\ N(0,\sigma^2)))))$	0.157557
$f_{10}(x)$	$(+(N(0,\sigma^2)\ (+(N(0,\sigma^2)\ N(0,\sigma^2)))))$	0.276311

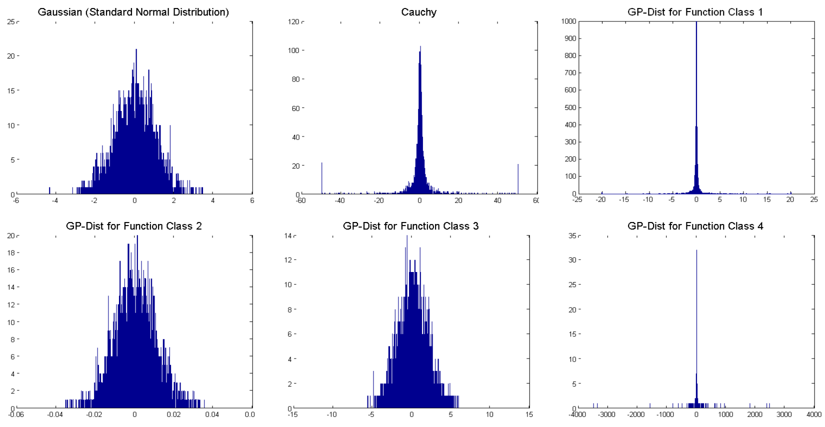

Fig. 1. Histograms of the distributions for 3000 samples

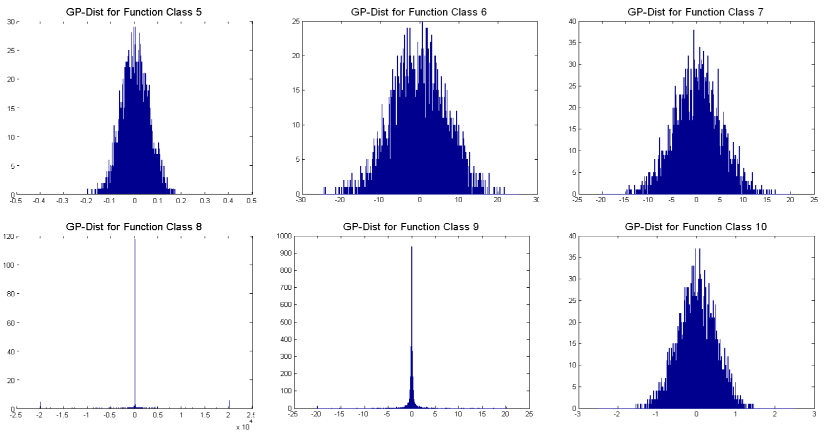

Fig. 2. Histograms of the distributions for 3000 samples

4.3 Test Function Classes

The results in Table 4 show that GP-distribution outperforms both Cauchy and Gaussian on all function classes. The results in Table 5 show that GP-distribution *statistically* outperforms both Cauchy and Gaussian on all function classes except f_{10} at the 0.05 level of confidence. In these initial experiments (which will form the start of a PhD thesis), even by allowing just σ of the Gaussian distribution to be altered we can outperform standard mutation operators on this set of function classes.

Table 4. The results for GP-distribution, FEP and CEP on f_1-f_{10}. All results have been averaged over 50 test runs, where "Mean Best" is the mean best function values found in the last generation, and "Std Dev" is the standard deviation.

Function	FEP		CEP		GP-distribution	
Class	*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}	$\mathbf{6.37\times10^{-5}}$	5.56×10^{-5}
f_2	1.53×10^{-1}	2.72×10^{-2}	4.30×10^{-2}	9.08×10^{-3}	$\mathbf{8.14\times10^{-4}}$	8.50×10^{-4}
f_3	2.74×10^{-2}	2.43×10^{-2}	5.15×10^{-2}	9.52×10^{-2}	$\mathbf{6.14\times10^{-3}}$	8.78×10^{-3}
f_4	1.79	1.84	1.75×10	6.10	$\mathbf{2.16\times10^{-1}}$	6.54×10^{-1}
f_5	2.52×10^{-3}	4.96×10^{-4}	2.66×10^{-4}	4.65×10^{-5}	$\mathbf{8.39\times10^{-7}}$	1.43×10^{-7}
f_6	3.86×10^{-2}	3.12×10^{-2}	4.40×10	1.42×10^{2}	$\mathbf{9.20\times10^{-3}}$	1.34×10^{-2}
f_7	6.49×10^{-2}	1.04×10^{-2}	6.64×10^{-2}	1.21×10^{-2}	$\mathbf{5.25\times10^{-2}}$	8.46×10^{-3}
f_8	-11342.0	3.26×10^{2}	-7894.6	6.14×10^{2}	**-12611.6**	2.30×10
f_9	6.24×10^{-2}	1.30×10^{-2}	1.09×10^{2}	3.58×10	$\mathbf{1.74\times10^{-3}}$	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 5. 2-tailed t-tests comparing EP with GP-distributions, FEP and CEP on f_1-f_{10}

Function Class	Number of Generations	GP-distribution vs FEP *t-test*	GP-distribution vs CEP *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	7.99×10^{-48}	5.82×10^{-75}
f_9	5000	6.37×10^{-55}	6.54×10^{-39}
f_{10}	1500	9.23×10^{-5}	1.93×10^{-1}

Note that if we had only allowed EP to alter σ, then this method would have been regarded as parameter tuning (i.e. σ is simply a parameter of the algorithm). However we are automatically synthesizing new distributions by combining (by adding, subtracting, dividing and multiplying) Gaussian distributions so are engaging in an activity more expressive than tuning a numerical parameter. We have only allowed the Gaussian distributions to vary in their standard deviation, and while it makes complete sense to allow their *means* to be evolved too, this result supports the approach of the automatic design of algorithms, or a component of (in this case probability distributions). As it is sufficient to outperform human designed heuristics. Further work will address the shortcomings of these initial experiments, which we will now consider.

5 Discussion and Future Work

The initial aim of this paper is to build a system which is capable for synthesizing distributions for use as a mutation operator in EP. So far we have only compared it with FEP and CEP which has been successful. Later work therefore will address comparisons with more recent developments in EP including LEP [5], IFEP [13] and MSEP [3].

We have run the GP system for a fixed number of iterations, but have not optimized these parameters. Hence there is further scope for improvement of results in this regard. Further work includes using more sophisticated methods of terminating the meta-search (i.e. GP), such as early stopping to prevent either under-fitting or over-fitting. This is a more crucial issue than with traditional base-level only approaches as each evaluation is itself done over 20 EP runs.

We have defined function classes in terms of a random variable which is 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 the GP terminal set we used a normal distribution with fixed μ (μ=0) and allowed σ to be set in the training phase. However, just by allowing σ to vary was enough to generate distributions which could beat Gaussian or Cauchy. Due to not allowing μ to vary meant, we could only generate symmetric distributions (see Fig.1 and Fig.2), but this is a reasonable assumption in the knowledge that all of the functions we are optimizing are symmetrical (why would be bias the search in one direction over another). However a hypothesis that comes out of this is the following: if we know a function is symmetrical (due to some real world domain knowledge) then a symmetric mutation operator will outperform an asymmetrical distribution. Similarly if we have little or no knowledge about the functions we are optimizing, then restricting the GP system to only produce symmetric distributions might over-constrain it.

One avenue of further work in GP is always to examine the parameters and components in more detail. In this paper we used a Gaussian distribution in the terminal set, but obviously it would be interesting to see if other distributions would give better results. However, we do not want to get involved in a circular argument, as we could try a Cauchy distribution in the GP terminal set (instead of a Gaussian). However nominating specific distributions (in the context of EP) is what we are trying to avoid in the first place (i.e. the whole point of the paper), and we are just raising the level of abstraction from the base-level to the meta-level [10]. Regardless of this dilemma we have still produced a system which outperforms the two human proposed systems we are comparing with.

6 Summary and Conclusions

EP is a robust method of solving numerical optimization problems. In the past this has involved using probability distributions (Gaussian and Cauchy) nominated by researchers as the mutation operator. In this paper we automatically generate probability distributions using GP, in a meta-learning approach, for use in EP. GP operates at the meta-level and contains a population of probability distributions which are inserted into an EP algorithm operating at the base-level and contains a population of numerical vectors. The fitness of a probability distribution is given by its performance over a number of function instances optimized by it in an EP algorithm. While EP is learning about values of single functions, GP is learning about distributions to be used by EP on functions drawn from a particular function class.

In a deviation from the approach used by other researcher, who tackle single isolated benchmark instances, we tackle function instances drawn from a function class by effectively implementing a probability distribution over function instances. During the training and testing phases the same function is highly unlikely to be seen twice. This is to demonstrate that the mutation operator has learned to generalize to the function class as a whole, rather than to any single instances.

Our initial results are highly encouraging. While we cannot claim that the distributions our method produces outperforms either Gaussian or Cauchy distributions on a single function (due to the statistical nature of EP), we can claim that on all but one function class (f_{10}) our method does produce distributions which statistically outperform the others (at a confidence level of 0.05) and at least does not under-perform on function class f_{10}.

One possible criticism of this method is the long training time required to evolve the distributions. After all we are evolving an evolutionary process itself. One line of future research is to speed-up this method, which is a central question for the GP community. For example, are GP trees the best representation for distributions? However, we claim that the amount of processor time required to generate distributions is still vastly less than the number of man-hours typically used in the design phase of new mutation operators (though of course the two are not directly comparable), and therefore the methodology proposed in this paper is a viable one.

References

1. Back, T., Schwefel, H.P.: An overview of evolutionary algorithms for parameter optimization. Evolutionary Computation 1, 1–23 (1993)
2. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R.: Exploring Hyper-heuristic Methodologies with Genetic Programming. In: Mumford, C.L., Jain, L.C. (eds.) Computational Intelligence. ISRL, vol. 1, pp. 177–201. Springer, Heidelberg (2009)
3. Dong, H., He, J., Huang, H., Hou, W.: Evolutionary programming using a mixed mutation strategy. Information Science, 312–327 (2007)
4. Fogel, D.B.: Evolving artificial intelligence. PhD thesis, University of California, San Diego (1992)
5. Lee, C.Y., Yao, X.: Evolutionary programming using mutations based on the lévy probability distribution. IEEE Transactions on Evolutionary Computation 8 (2004)

6. Mallipeddi, R., Suganthan, P.N.: Evaluation of novel adaptive evolutionary programming on four constraint handling techniques. In: IEEE Congress on Evolutionary Computation, pp. 4045–4052 (2008)
7. Mallipeddi, R., Mallipeddi, S., Suganthan, P.N.: Ensemble strategies with adaptive evolutionary programming. Information Science, 1571–1581 (2010)
8. Poli, R., Langdon, W.B., et al.: A field guide to genetic programming (2008) ISBN 978-1-4092-0073-4
9. Su Nguyen, M.Z., Johnston, M.: A genetic programming based hyper-heuristic approach for combinatorial optimization. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, pp. 1299–1306 (2011) ISBN 978-1-4503-0557-0
10. Woodward, J.: The necessity of meta bias in search algorithms. In: IEEE International Conference on Computational Intelligence and Software Engineering, CiSE (2010)
11. Woodward, J., Swan, J.: Automatically designing selection heuristics. In: ACM Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 583–590 (2011)
12. Woodward, J., Swan, J.: The automatic generation of mutation operators for genetic algorithms. In: ACM Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion, pp. 67–74 (2012)
13. Xin Yao, Y.L., Lin, G.: Evolutionary programming made faster. IEEE Transactions on Evolutionary Computation 3, 82–102 (1999)
14. Yao, X., Liu, Y.: Fast evolutionary programming. In: Proceedings of the Fifth Annual Conference on Evolutionary Programming, pp. 451–460. MIT Press (1996)