Automatically Designing More General Mutation Operators of Evolutionary Programming for Groups of Function Classes Using a Hyper-Heuristic

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

John R. Woodward
Computing Science and Mathematics
University of Stirling U.K.
JRW@cs.stir.ac.uk

Ender Özcan
Department of Computer Science
University of Nottingham U.K.
Ender.Ozcan@nottingham.ac.uk

ABSTRACT

In this study we use Genetic Programming (GP) as an offline hyper-heuristic to evolve a mutation operator for Evolutionary Programming. This is done using the Gaussian and uniform distributions as the terminal set, and arithmetic operators as the function set. The mutation operators are automatically designed for a specific function class. The contribution of this paper is to show that a GP can not only automatically design a mutation operator for Evolutionary Programming (EP) on functions generated from a specific function class, but also can design more general mutation operators on functions generated from groups of function classes. In addition, the automatically designed mutation operators also show good performance on new functions generated from a specific function class or a group of function classes.

Keywords

Hyper-Heuristic, Genetic Programming, Optimization, Function Class

1. INTRODUCTION

A hyper-heuristic searches for heuristics for computational search problems. During the searching process it can generate or select heuristics according to its learning mechanism [4]. Burke et al. [5] proposed a distinction between *online* and *offline* learning, according to the source of the feedback during learning. For an *online* hyper-heuristic, the learning occurs when the heuristic is solving an instance of a problem. For an *offline* hyper-heuristic collects knowledge, from a training a set of instances to solve unknown instances of the same problem. Recently GP has been used with hyper-heuristics for the bin packing problem [6], the multidimensional knapsack problem [8], to evolve highly competitive general algorithms for envelope reduction in sparse matrices [12], to handle multiple conflicting objectives in dynamic job shop scheduling [18], to automatically design a mutation operator for Evolutionary Programming [10], to compare rule representations [11], to evolve due-date assignment models in job shop environments [20], to automatic design schedule policies for dynamic multi-objective job shop scheduling [19], to evolve ensembles of dispatching rules for the job shop scheduling problem [22], for feature selection and question–answer ranking in IBM Watson [2], to automated design production scheduling heuristics [3].

Burke et al. [6] proposed to automatically design heuristic for the bin packing problem and these heuristics are "a Jack-of-all-Trades or a Master of One." Burke et al. [6] point out that heuristics can be evolved to be specialists on a particular sub-problem, or general enough to work on all sub-problems. However there is a trade-off between performance and generalisation. The hypothesis of this paper, inspired by [6]: if the probability distribution over function instances is specific, we can design a specific EP mutation operator for a specific function class. We can also design a mutation operator of EP that performs well on a group of function classes. To verify this hypothesis we designed two types of experiment. In the first experiment, we tailor mutation operators for EP to a specific function class, and the best results are used as the fitness values for GP. This kind of experiment was proposed in [10]. In the second experiment, we tailor more general mutation operators for a group of function classes (each group contains three function classes) and use the number of outright wins as the fitness values for the GP. After completing the experiment, we test the Automatically Designed Mutation Operators (ADM) from both experiments and human designed mutation operators on a separate independent test set of functions. To make a fair comparison, we use Borda count to evaluate the performance. The comparison shows that the more general automatically designed mutation operators from the second experiment has the better performance on average on groups of function classes.

The outline of this paper is as follows. In Section 2, we describe function optimization. In Section 3, we describe

GECCO '16, July 20-24, 2016, Denver, CO, USA

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

the training types and the parameter settings for GP and EP. In Section 4, we describe all the testing of ADMs on function classes to identify the performance of the result. In Section 5, we analyse and compare the testing results. In Section 6, we summarize and conclude the paper.

2. FUNCTION OPTIMIZATION BY EVOLUTIONARY PROGRAMMING

EP is an algorithm to evolve a population of numerical vectors in order to find a global optimum of a function in a limited search region. Mutation is the only operator for EP. Researchers have made great efforts to finding and selecting mutation operators, or developing more advanced mutation strategies for EP in recent years [23, 24, 13, 7, 16, 15, 9].

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. Hence the EP algorithm searches for a global optimum within a limited space. The aim of EP 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 mutation process of EP can be represented by the following equations.

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

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

In the above equations, i is the dimension and j represents $j-th$ component of the vectors x_i, x_i', η_i and η_i', D_j represents the mutation operator, researchers usually use a Cauchy, Gaussian or Lévy distribution as the mutation operator [23, 24, 13]. For a complete description of EP, refer to [1]. Lee et al. [13] point out that the Lévy distribution with α=1.0 is the Cauchy distribution and with α=2.0 is the Gaussian distribution. We use α=1.0 and α=2.0 to represent the Cauchy and Gaussian distribution. In this paper, the framework automatically designs a piece of the program for EP to replace D_j. Then the EP algorithm uses the candidate mutation operator D_j to do the training and testing on functions generated from function classes.

3. USING GP TO AUTOMATICALLY DESIGN A MUTATION OPERATOR FOR EP

In this section we describe the methods that set up connections between GP and EP. The EP parameters we use for this paper are in Table 2. In order to reduce the time cost of the training phase, we set the number of generations for each function class, please refer to Section 3.3. The codes we list in Table 3 are implemented according to Mantegna's description [17]. Equation 3 and 4 show how to generate a random variable from a Lévy distribution with the corresponding α ($0.75 \leq \alpha \leq 1.95$). In equation 3, V is calculated from X and Y, where X is a random variant from a $N(0, \sigma^2)$ distribution and Y is a random variate from a $N(0, 1)$ distribution. $K(\alpha)$ and $C(\alpha)$ are two parameters with real values, which can be looked up in [17], and must be determined properly. The values of σ_x, $K(\alpha)$ and $C(\alpha)$ used in this paper are listed in Table 4. For a more detailed derivation of the equations, please refer to [17].

$$V = \frac{X}{\mid Y \mid^{1/\alpha}} \quad (3)$$

$$W = ((K(\alpha) - 1)exp(-V/C(\alpha)) + 1)V \quad (4)$$

3.1 Functions and Function Classes

In a suite of 23 functions often used in EP research [23, 24], the functions can be classified as: f_1–f_7 are unimodal functions, f_8–f_{13} are multimodal functions with many local optima, f_{14}–f_{23} are multimodal functions with a few local optima [24].

In this study, we do not use single functions for benchmark function optimisation. Instead, we use function classes, where each class is a single parameterised function which embodies a set of unique functions each having fixed parameter values. Based on these 23 functions, we have constructed corresponding function classes. To distinguish between functions and function classes, we use the notation f_g to represent function and F_g to represent function class. In this paper, we select 3 function classes from each of the unimodal, and multimodal with many and few local optima (F_1, F_2, F_6, F_{10}, F_{12}, F_{13}, F_{16}, F_{19} and F_{23}). The training and test function classes used in this study are given in Table 1, with the index of each function class corresponding to the original functions of [23]. In Table 1, a_i, b_i and c_i are uniformly distributed in range $[1, 2]$, $[-1, 1]$ and $[-1, 1]$, respectively. An example of a function class is: $y = \sum_{i=1}^{n} a_i x_i^2$, in this case $y = \sum_{i=1}^{n} 1.3x_i^2$ is a function from this function class, while $y = \sum_{i=1}^{n} 0.3x_i^2$ is not from this function class.

3.2 Experimental Design and Fitness Functions for GP

The experiments can be divided into two different training types: one type of training is for a specific function class, e.g., F_1, another type of training is for a group of function classes, e.g., the unimodal F_1, F_2 and F_6. We designed two different fitness functions for each training. In both training types the GP settings are the same, only the functions vary. For the parameter settings of GP, please refer to Table 5. We call a program generated by GP an Automatically Designed Mutation operator (ADM). The tailored ADM automatically designed for a function class F_g by training type 1 is represented by ADM_g, where g is the function class index. ADM_g is called a dedicated ADM for function class F_g. The tailored ADM automatically designed for a group of function classes F_g, F_j and F_k by training type 2 is represented as $ADM_{g,j,k}$, where g, j, k represent the indexes of the different function classes. $ADM_{g,j,k}$ is called a more general ADM for function class on F_g, F_j and F_k. To distinguish the fitness functions for each type of training, we describe training of type 1 and training of type 2 in the paragraphs below.

Training Type 1: Each ADM_g is used as an EP mutation operator on 9 functions drawn from a given function class. The fitness of an ADM_g is the average of the best values obtained in each of the individual 9 EP runs on a given function class. We use the same 9 functions from each function class for the entire run of the GP on a given function class. For one function class, 18 functions are taken for training, 9 of which are used to calculate the fitness value and 9 of which are used to the monitor overfitting.

Table 1 Function Classes with n dimensions and domain S, $a_i \in [1,2]$, $b_i, c_i \in [-1,1]$.

Function Class	n	S
$F_1(x) = \sum_{i=1}^{n} [(a_i x_i - b_i)^2 + c_i]$	30	$[-100, 100]^n$
$F_2(x) = \sum_{i=1}^{n} \mid a_i x_i \mid + \prod_{i=1}^{n} \mid b_i x_i \mid$	30	$[-10, 10]^n$
$F_3(x) = \sum_{i=1}^{n} [a_i \sum_{j=1}^{i} x_j]^2$	30	$[-100, 100]^n$
$F_4(x) = \max_i \{\mid a_i x_i \mid, 1 \leq i \leq n\}$	30	$[-100, 100]^n$
$F_5(x) = \sum_{i=1}^{n} [a_i (x_{i+1} - x_i^2)^2 + b_i (x_i - 1)^2 + c_i]$	30	$[-30, 30]^n$
$F_6(x) = \sum_{i=1}^{n} (\lfloor a_i x_i + 0.5 \rfloor)^2 + b_i$	30	$[-100, 100]^n$
$F_7(x) = \sum_{i=1}^{n} a_i i x_i^4 + random[0, 1)$	30	$[-1.28, 1.28]^n$
$F_8(x) = \sum_{i=1}^{n} -(x_i \sin(\sqrt{\lvert x_i \rvert}) + a_i)$	30	$[-500, 500]^n$
$F_9(x) = \sum_{i=1}^{n} [a_i x_i^2 + b_i (1 - cos(2\pi x_i))]$	30	$[-5.12, 5.12]^n$
$F_{10}(x) = -\exp(-0.2\sqrt{\frac{1}{n} \sum_{i=1}^{n} a_i x_i^2}) - \exp(\frac{1}{n} \sum_{i=1}^{n} b_i \cos 2\pi x_i) + e$	30	$[-32, 32]^n$
$F_{11}(x) = \frac{a_i}{4000} \sum_{i=1}^{n} x_i^2 - b_i \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})$	30	$[-600, 600]^n$
$F_{12}(x) = \frac{\pi}{n} \{10 sin^2(\pi y_i) + a_i \sum_{i=1}^{n-1} (y_i - 1)^2 [1 + 10 sin^2(\pi y_{i+1})$	30	$[-50, 50]^n$
$+ (y_n - 1)^2]\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$,		
$y_i = 1 + \frac{1}{4}(x_i + 1)$		
$u(x_i, w, k, m) = \begin{cases} k(x_i - w)^m, & x_i > w, \\ 0, & -w \leq x_i \leq w, \\ k(-x_i - w)^m, & x_i < -w. \end{cases}$		
$F_{13}(x) = 0.1\{sin^2(3\pi x_1) + a_i \sum_{i=1}^{n-1} (x_i - 1)^2 [1 + sin^2(3\pi x_{i+1})] +$	30	$[-50, 50]^n$
$(x_n - 1)[1 + sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$		
$F_{14}(x) = [\frac{1}{500} + a_i \sum_{i=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - w_{ij})^6}]^{-1}$	2	$[-65.536, 65.536]^n$
$F_{15}(x) = \sum_{i=1}^{11} [w_i - \frac{a_i x_1 (y_i^2 + y_i x_2)}{b_i (y_i^2 + y_i x_3 + x_4)}]^2$	4	$[-5, 5]^n$
$F_{16}(x) = a_1 (4x_1^2 - 2.1x_1^4 + \frac{1}{3} x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4) + b_1$	2	$[-5, 5]^n$
$F_{17}(x) = a_1 (x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6)^2 + 10 b_1 (1 - \frac{1}{8\pi}) cos x_1 + 10$	2	$[-5, 10] \times [0, 15]$
$F_{18}(x) = a_1 [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2$	2	$[-2, 2]^n$
$+ 6x_1 x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2 (18 - 32x_1$		
$+ 12x_1^2 + 48x_2 - 36x_1 x_2 + 27x_2^2)] + b_1$		
$F_{19}(x) = -\sum_{i=1}^{4} y_i exp[-\sum_{j=1}^{4} a_j w_{ij} (x_j - p_{ij})^2 + b_i]$	3	$[0, 1]^n$
$F_{20}(x) = -\sum_{i=1}^{4} y_i exp[-\sum_{j=1}^{6} a_j w_{ij} (x_j - p_{ij})^2 + b_i]$	6	$[0, 1]^n$
$F_{21}(x) = -\sum_{i=1}^{5} a_i [(x - w_i)^T (x - w_i) + y_i + b_i]^{-1}$	4	$[0, 10]^n$
$F_{22}(x) = -\sum_{i=1}^{7} a_i [(x - w_i)^T (x - w_i) + y_i + b_i]^{-1}$	4	$[0, 10]^n$
$F_{23}(x) = -\sum_{i=1}^{10} a_i [(x - w_i)^T (x - w_i) + y_i + b_i]^{-1}$ *where* $y_i = 0.1$	4	$[0, 10]^n$

Table 2 Parameter settings for EP.

Parameter	Settings
population size	100
tournament size	10
initial standard deviations of initial population	3.0

Table 3 How the Lévy distribution is constructed from the normal distribution.

Algorithm for Lévy distribution
$\alpha 1 = 1.0/\alpha$
$X = N(0, \sigma^2)$
$Y = N(0, 1)$
$V = X/(abs(Y)^{\alpha 1})$
$W = ((K(\alpha) - 1) * exp(-abs(V)/C(\alpha)) + 1.0) * V$

Table 4 α values and related σ^2, $K(\alpha)$ and $C(\alpha)$.

α	σ^2	$K(\alpha)$	$C(\alpha)$
1.2	0.878829	1.20519	2.941
1.4	0.759679	1.44647	2.8315
1.6	0.628231	1.79361	2.6125
1.8	0.458638	2.50147	2.206

Table 5 Parameter settings for GP.

Parameter	Settings
crossover proportion	45%
mutation proportion	45%
reproduction proportion	10%
selection method	lexictour [14]
tournament size	2
depthnodes	2 [14]
population size	20
maximum generation	25
elitism	keep best

Table 6 Function set for GP.

Symbol	Function	Arity
$+$	addition	2
$-$	subtraction	2
$\times$	multiplication	2
$\div$	protected division	2
power	power	2
exp	exponential function	1
abs	absolute	1

Table 7 Terminal set for GP.

Symbol	Terminal
$N(\mu, \sigma^2)$	Normal Distribution
U	∼[0, 3]

This type of training was used in [10]. In this training, the fitness value of the GP is from the averaged fitness values of 9 EP runs.

Training Type 2: We train $ADM_{g,j,k}$ on 9 functions (three from each of F_g, F_j and F_k) by GP. Then we validate $ADM_{g,j,k}$ on 9 separate functions (three from each of F_g, F_j and F_k). If the fitness value of the EP that uses $ADM_{g,j,k}$ beats all of the human designed mutation operators (Lévy distribution (with α=1.0, 1.2, 1.4, 1.6, 1.8, 2.0)), Lévy with $\alpha = 1.0$ is Cauchy, Lévy with $\alpha = 2.0$ is Gaussian), on the given function, it scores 1 point, thus it can score between 0 and a maximum of 9 (we call this the number of outright wins). Here we use the number of outright wins because averaging fitness values can be skewed by a single large value, and using a rank-based method is more robust to outliers. In this training, the fitness value of the GP is the number of outright wins.

For both training types, the framework can not only express a number of currently existing human designed EP mutation operators (Cauchy, Gaussian and Lévy distributions), but also can generate new kinds of mutation operators for EP. The main aim of this paper is to set up an algorithmic framework which can automatically design a more general mutation operator for EP on groups of function classes. We use GP as an *offline* hyper-heuristic to evolve a mutation operator for EP. But in contrast to what we have done in [10], in this paper the hyper-heuristic uses three groups of function classes, each group containing three function classes F_g, F_j and F_k, where g, j, k are different indexes of function classes.

3.3 Parameter Settings for EP

The settings for EP are presented in Table 2: the population size is set to 100, the tournament size is set to 10, the initial standard deviations is set to 3.0. The settings for dimensions n and domains S are listed in Table 1. We have to point out that in our experiment the maximum number of generations of EP is set to 1000 for F_1, F_2, F_6, F_{10}, F_{12} and F_{13}. The maximum number of generations of EP is set to 100 for F_{16}, F_{19} and F_{23}.

3.4 Parameter Settings for GP

The parameter settings for GP are listed in Table 5. The function set and terminal set of GP are listed in Table 6 and 7. μ is a random number in $[-2, 2]$, as we wish the designed mutation operator is not Y-axis symmetric. σ^2 is a random number in $[0, 5]$. *depthnodes* is set as 2 indicates restrictions are to be applied in tree size (number of nodes) [14]. U is the uniform distribution with range $[0, 3]$. The other settings of GP are: population size 20, the maximum number of generations 25. The settings of GP in Table 5 are able to generate the values in Table 4, and the (human designed) piece of programs in Table 3 and other programs.

4. TESTING AUTOMATICALLY DESIGNED MUTATION OPERATORS

We employ $ADMs$ (in Table 14) and human designed mutation operators on EP and test them on each function class F_g. For each ADM we record 50 values from 50 independent EP runs, each being the lowest value through all generations of EP, we then average them, this is called the mean best values. We test all ADMs on F_g respectively. In all testing the generated functions from each function class are the same. This means the results in Tables 8, 9, 10, 11, 12 and 13 are based on the same 50 functions generated from each function class F_g.

4.1 Testing the More General ADMs and Human Designed Mutation Operators

We did the testing for $ADM_{g,j,k}$ and human designed mutation operators (Lévy distribution (with α=1.0, 1.2, 1.4, 1.6, 1.8, 2.0)) on F_g. The mean best values and standard deviations are listed in Table 8. Based on the original data for Table 8, we also calculate the Borda counts B_g for all test mutation operators to compare the performance of $ADM_{1,2,6}$, $ADM_{10,12,13}$, $ADM_{16,19,23}$ and human designed mutation operators (in all, 9 mutation operators) in Table 10. We follow the method to calculate Borda counts in [21]: Test each mutation operator for each function and it has a rank R_{mn}, where m is the function index ($1 \leq m \leq 50$) and n is the mutation operator index ($1 \leq n \leq 9$). The value of R_{mn} is in range [1, 9]. The Borda counts $B_g = \sum_{i=1}^{m} R_{mn}(m \in 1, 2, 3, ...50)$, is the sum of R_{mn} on 50 functions generated from the function class F_g; it has its best possible value 50 and the worst possible value 450 (g is the index of the function class). Each mutation operator has Borda counts B_g on F_g and the sum of the Borda counts $B_{g,j,k} = B_g + B_j + B_k$.

4.2 Testing More General ADMs and Dedicated ADMs

To observe the performance of dedicated ADM_g and $ADM_{g,j,k}$ on F_g, we tested ADM_g and $ADM_{g,j,k}$ on F_g. We list the mean best values and standard deviations in Table 11. In this table we consolidate the mean best values and standard deviations for $ADM_{1,2,6}$, $ADM_{10,12,13}$ and $ADM_{16,19,23}$, we also put more decimal places for F_{16} and F_{19}, as otherwise, the results are too close to distinguish. We use the Borda counts to compare the performance of the 12 mutation operators in Table 13. In this comparison, the number of functions $p = 50$ and the number of mutation operators $q = 12$. Therefore, the best possible score is 50, and the worst possible is 600. Each mutation operator has Borda counts B_g on F_g, each mutation operator has the sum of Borda counts $B_{g,j,k} = B_g + B_j + B_k$.

4.3 Testing More General ADMs and Human Designed Mutation Operators on Non-Trained Function Classes

To observe the performance of $ADM_{g,j,k}$ on Non-Trained Function Classes (F_3, F_4, F_5, F_7, F_8, F_9, F_{11}, F_{14}, F_{15}, F_{17}, F_{18}, F_{20}, F_{21} and F_{22}), we tested $ADM_{g,j,k}$ and human designed mutation operators (Lévy (α =1.0, 1.2, 1.4, 1.6, 1.8, 2.0)) on Non-Trained Function Classes. The results are in Table 15.

5. ANALYSIS AND COMPARISON

In this section we compare the mutation operators $ADM_{g,j,k}$, ADM_g and the human designed mutation operators. An ADM_g designed for the function class F_g is called a tailored mutation operator, while an ADM_g tested on F_j is called a non-tailored mutation operator. For example, ADM_1 is tailored mutation operator for F_1, but it is a non-tailored mutation operator for the function class F_2. Similarly for a group of function classes F_1, F_2 and F_6, $ADM_{1,2,6}$ is a more general tailored mutation operator for this group of function classes, while $ADM_{10,12,13}$ and $ADM_{16,19,23}$ are non-tailored mutation operators.

5.1 Analysis and Comparison of More General ADMs and Human Designed Mutation Operators

From Table 8, 9, 11 and 12 we can see that $ADM_{g,j,k}$ show the outstanding performance on all F_g in most cases. In Table 10, which presents the Borda counts, among all Borda counts of tested mutation operators, $ADM_{g,j,k}$ has the best/lowest scores on the groups of function classes: the Borda counts $B_{1,2,6}$ of $ADM_{1,2,6}$ is 344, which is the best/lowest value on F_1, F_2 and F_6. The Borda counts $B_{10,12,13}$ of $ADM_{10,12,13}$ is 320, which is the best/lowest value on F_{10}, F_{12} and F_{13}. The Borda counts $B_{16,19,23}$ of $ADM_{16,19,23}$ is 213, which is the best/lowest value on F_{16}, F_{19} and F_{23}. Although B_1, B_2 and B_6 for $ADM_{1,2,6}$ are not the best values for F_1, F_2 and F_6, $B_{1,2,6}$ the sum of B_1, B_2 and B_6 shows that $ADM_{1,2,6}$ has the best performance among all tested mutation operators. B_{12} and B_{13} for $ADM_{10,12,13}$ are the best/lowest value on F_{12} and F_{13} respectively, B_{10} of $ADM_{1,2,6}$ show the best performance on F_{10}. We think this is because the function characteristics of F_{10} are different from the characteristics of F_{12} and F_{13}. B_{16}, B_{19} and B_{23} for $ADM_{16,19,23}$ are the best/lowest value on F_{16}, F_{19} and F_{23} respectively. In general, a tailored $ADM_{g,j,k}$ always show a better performance than the human designed mutation operator and non-tailored $ADM_{g,j,k}$.

Table 9 shows the results of the Wilcoxon Signed-Rank Test within 5% significance level comparing a tailored $ADM_{g,j,k}$ compared with human designed mutation operators (Lévy distribution (with α=1.0, 1.2, 1.4, 1.6, 1.8, 2.0)). Table 12 shows the results of a Wilcoxon Signed-Rank Test within 5% significance level comparing a tailored $ADM_{g,j,k}$ compared with other ADMs. In both tables, "$\geq$" and "$\leq$" indicate that the $ADM_{g,j,k}$ performs better or worse on F_g, F_j and F_k respectively, compared to human designed mutation operators or ADMs. In the case that this difference is statistically significant, "$>$" and "$<$" are used.

5.2 Analysis and Comparison of More General ADMs and ADMs

In Table 11 the results using ADM_g to do test on F_g. ADM_1, ADM_2, ADM_6, ADM_{12}, ADM_{19} and ADM_{23} show the best performance on F_1, F_2, F_6, F_{12}, F_{19} and F_{23} respectively. ADM_{10} shows the second best performance on F_{10}, as $ADM_{1,2,6}$ has the best performance on F_{10}. ADM_{13} has the third best performance on F_{13}, as ADM_{12} and $ADM_{10,12,13}$ beat ADM_{13} on F_{13}. ADM_{16} is a special case, ADM_{16} does not show any outstanding performance among the ADMs on F_{16}. We think this is because the training of ADM_{16} was insufficient, or we need to record more decimals to do the analysis, as the values in **boldface** are the same.

In Table 13 the Borda counts $B_{1,2,6}$ of $ADM_{1,2,6}$ is 434, which beats all other ADMs on F_1, F_2 and F_6. The Borda counts $B_{10,12,13}$ of $ADM_{10,12,13}$ is 441, which beats all other ADMs on F_{10}, F_{12} and F_{13} as well. However, the Borda counts $B_{16,19,23}$ of $ADM_{16,19,23}$, which is the second best value, is 393 on F_{16}, F_{19} and F_{23}. The best value 284 is that of the Borda counts $B_{16,19,23}$ of ADM_{19} on F_{16}, F_{19} and F_{23}. This is an acceptable exception, as in the GP training system we only use the human designed mutation operator to evaluate the performance of the ADMs, both ADM_g and $ADM_{g,j,k}$ beat the human designed mutation operator separately.

Overall, the results in Table 11 and 13 demonstrate that the tailored mutation operator ADM_g has better mean best values than the non-tailored mutation operators and $ADM_{g,j,k}$ on F_g, although there are some exceptions (for example, ADM_{10} on F_{10} and ADM_{16} on F_{16} are not the best). The Borda counts in Table 13 demonstrate that the tailored general mutation operator $ADM_{g,j,k}$ has better performance on the groups of function classes F_g, F_j and F_k. Both $ADM_{g,j,k}$ and ADM_g have better performances than the human designed mutation operators on average; the experiment we designed successfully found a more general mutation operator $ADM_{g,j,k}$ that has better performance than other mutation operators on a group of function classes F_g, F_j, F_k on average.

5.3 Analysis and Comparison of More General ADMs and Human Designed Mutation Operators on Non-Trained Function Classes

In Table 15, we test the more general mutation operator $ADM_{g,j,k}$ and the human designed mutation operators on the non-trained function classes F_g over 50 runs. From this table, although $ADM_{1,2,6}$ does not show the best performance on F_3, F_4, F_5, F_7, its performance is not the worst, we think this is because $ADM_{1,2,6}$ fit F_1, F_2 and F_6 well, but may have over-fit on F_3, F_4, F_5 and F_7. $ADM_{10,12,13}$ show the best performance on F_8, and the second best on F_9, F_{11}. $ADM_{16,19,23}$ has the best performance on F_{17}, F_{18}, F_{20}, F_{21}, F_{22}, but has the worst performance on F_{14} and F_{15}, we think this is because F_{14} and F_{15} has different function characteristics, and hence $ADM_{10,12,13}$ cannot fit them well.

To make the results can be easily observed. In Table 8 and 15 the mean best values are in **bold**. In Table 9 and 12 "$>$" are in **bold**. In Table 10 and 13 the lowest Borda counts B_g and $B_{g,j,k}$ of the tested mutation operator are in **bold**. In Table 11 the mean best values using ADM_g to do test on F_g are in **bold** and the results which are lower than test result of ADM_g on F_g are also in **bold**.

6. SUMMARY AND CONCLUSIONS

In this paper we designed a framework to automatically design more general tailored mutation operators for several groups of function classes. Previously, researchers have used GP to tailor mutation operators [10] for EP on a specific function class. We proposed using the number of outright wins, the number of times that automatically designed mutation operator has beaten the human designed mutation operators, as the fitness value for the GP. We did the test to evaluate the performance of the more general tailored mutation operators, tailored mutation operators and human designed mutation operators on a specific function class and on groups of function classes.

The main conclusions of this paper are: Firstly, on new functions generated from a particular function class, a tailored mutation operator evolved on functions drawn from that function class will perform better on average than a tailored mutation operator evolved on functions from a different function class. Secondly, a more general tailored mutation operator can be evolved to be specialists on a particular group of function classes. Thirdly, both tailored mutation operator and more general tailored mutation operator have better performances than human designed mutation operators. Fourthly, compared with the more general tailored mutation operator and the tailored mutation operator on a specific function class, tailored mutation operator usually has better performance on a specific function class, but the more general tailored mutation operator usually has better performance on a group of function classes on average.

7. REFERENCES

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

[2] U. Bhowan and D. McCloskey. Genetic programming for feature selection and question-answer ranking in ibm watson. In *Genetic Programming*, volume 9025 of *Lecture Notes in Computer Science*, pages 153–166. Springer International Publishing, 2015.

[3] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124, Feb 2016.

[4] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, pages 1695–1724, 2013.

[5] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*, pages 449–468. Springer US, 2010.

[6] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pages 1559–1565. 2007.

[7] H. Dong, J. He, H. Huang, and W. Hou. Evolutionary programming using a mixed mutation strategy. *Information Sciences*, 177(1):312–327, 2007.

[8] J. H. Drake, M. Hyde, K. Ibrahim, and E. Özcan. A genetic programming hyper-heuristic for the multidimensional knapsack problem. 43:1500–1511, 2014.

[9] L. Hong, J. H. Drake, and E. Özcan. A step size based self-adaptive mutation operator for evolutionary programming. In *Proceedings of Genetic and Evolutionary Computation Conference 2014*, pages 1381–1388. ACM, 2014.

[10] 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.

[11] T. H. J. Branke and B. Scholz-Reiter. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. volume 23, pages 249–277. Evolutionary Computation, 2014.

[12] B. Koohestani and R. Poli. A genetic programming approach for evolving highly-competitive general algorithms for envelope reduction in sparse matrices. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature*, volume 2, pages 287–296. Springer-Verlag, Berlin, Heidelberg, 2012.

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

[14] S. Luke and L. Panait. Lexicographic parsimony pressure. In *Proceedings of Genetic and Evolutionary Computation Conference 2002*, pages 829–836. Morgan Kaufmann Publishers, 2002.

[15] R. Mallipeddi, S. Mallipeddi, and P. Suganthan. Ensemble strategies with adaptive evolutionary programming. *Information Sciences*, 180(9):1571–1581, 2010.

[16] R. Mallipeddi and P. N. Suganthan. Evaluation of novel adaptive evolutionary programming on four constraint handling techniques. In *Proceedings of IEEE World Congress on Computational Intelligence*, pages 4045–4052. 2008.

[17] R. N. Mantegna. Fast, accurate algorithm for numerical simulation of lévy stable stochastic processes. *Physical Review E*, 49:4677–4683, May 1994.

[18] S. Nguyen, M. Zhang, M. Johnston, and K. Tan. Dynamic multi-objective job shop scheduling: A genetic programming approach. In *Automated Scheduling and Planning*, volume 505 of *Studies in Computational Intelligence*, pages 251–282. Springer Berlin Heidelberg, 2013.

[19] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan. Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation*, 18(2):193–208, April 2014.

[20] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan. Genetic programming for evolving due-date assignment models in job shop environments. *Evolutionary Computation*, 22(1):105–138, March 2014.

[21] G. Ochoa, J. Walker, M. Hyde, and T. Curtois. Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework. In *Proceedings of Parallel Problem Solving from Nature - PPSN XII: 12th International Conference*, volume 7492 of *Lecture Notes in Computer Science*, pages 418–427. Springer Berlin Heidelberg, 2012.

[22] J. Park, S. Nguyen, M. Zhang, and M. Johnston. Evolving ensembles of dispatching rules using genetic programming for job shop scheduling. In *Genetic Programming*, volume 9025 of *Lecture Notes in Computer Science*, pages 92–104. Springer International Publishing, 2015.

[23] X. Yao and Y. Liu. Fast evolutionary programming. In *Proceedings of the 5th Annual Conference on Evolutionary Programming*, pages 451–460. MIT Press, 1996.

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

Table 8 The results of $ADM_{g,j,k}$, Lévy(1.0, 1.2, 1.4, 1.6, 1.8, 2.0) on trained function classes F_g. Mean indicates the mean of the best values found over all generations over 50 runs for F_g. Std Dev stands for standard deviations.

F_g		$ADM_{1,2,6}$	$ADM_{10,12,13}$	$ADM_{16,19,23}$	$\alpha = 1.0$	$\alpha = 1.2$	$\alpha = 1.4$	$\alpha = 1.6$	$\alpha = 1.8$	$\alpha = 2.0$
F_1	Mean	0.3711	**0.3624**	166.2	**0.3624**	**0.3654**	0.3797	0.5124	0.5567	1.0003
	Std Dev	(3.11)	(3.19)	(883.8)	(3.19)	(3.19)	(3.19)	(3.17)	(3.12)	(3.25)
F_2	Mean	**0.040**	0.072	0.043	0.160	0.109	0.089	0.077	0.068	0.048
	Std Dev	(4.40E-03)	(9.16E-03)	(9.59E-02)	(1.48E-02)	(1.42E-02)	(9.03E-03)	(8.54E-03)	(6.13E-03)	(6.51E-03)
F_6	Mean	0.50	**0.46**	729.1	**0.46**	0.52	1.48	178.5	111.4	605.9
	Std Dev	(3.41)	(3.41)	(1621.6)	(3.41)	(3.49)	(5.92)	(1179.3)	(312.1)	(1578.1)
F_{10}	Mean	**-20.04**	-19.24	-7.77	-3.66	-15.58	-19.05	-17.32	-14.32	-10.71
	Std Dev	(0.31)	(3.86)	(5.93)	(7.58)	(8.24)	(2.78)	(2.52)	(3.99)	(4.09)
F_{12}	Mean	0.0148	**0.0031**	3.1889	0.0135	0.1117	0.7234	1.1435	2.2114	4.0837
	Std Dev	(0.03)	(0.02)	(2.91)	(0.05)	(0.20)	(1.00)	(1.14)	(1.88)	(3.85)
F_{13}	Mean	0.0936	**0.0042**	20.85	0.0074	0.4746	4.10	12.42	14.21	27.54
	Std Dev	(0.24)	(0.01)	(25.4)	(0.02)	(1.52)	(10.2)	(12.9)	(19.9)	(28.3)
F_{16}	Mean	**-1.803489196**	**-1.803489196**	**-1.803489196**	-1.80348918	-1.803489185	-1.803489184	-1.803489186	-1.803489186	-1.803489192
	Std Dev	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)
F_{19}	Mean	-5.23096752	-5.230967433	**-5.230967576**	-5.225685243	-5.225723116	-5.230937535	-5.225024015	-5.225726632	-5.230966751
	Std Dev	(1.99)	(1.99)	(1.99)	(1.99)	(1.98)	(1.99)	(1.99)	(1.98)	(1.99)
F_{23}	Mean	-1.55E+07	-1.83E+07	**-9.75E+07**	-1.33E+06	-3.78E+06	-7.60E+06	-3.10E+06	-6.41E+06	-3.50E+06
	Std Dev	(4.22E+07)	(7.02E+07)	(3.26E+08)	(2.46E+06)	(1.19E+07)	(2.62E+07)	(6.94E+06)	(2.61E+07)	(5.76E+06)

Table 9 Wilcoxon Signed-Rank Test of $ADM_{g,j,k}$ versus Lévy Distribution (with α = 1.0, 1.2, 1.4, 1.6, 1.8, 2.0).

F_g	$ADM_{g,j,k}$	$ADM_{1,2,6}$	$ADM_{10,12,13}$	$ADM_{16,19,23}$	$\alpha = 1.0$	$\alpha = 1.2$	$\alpha = 1.4$	$\alpha = 1.6$	$\alpha = 1.8$	$\alpha = 2.0$
F_1	$ADM_{1,2,6}$	N/A	$<$	$>$	$<$	$\leq$	$\geq$	$>$	$>$	$>$
F_2	$ADM_{1,2,6}$	N/A	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$
F_6	$ADM_{1,2,6}$	N/A	$=$	$>$	$=$	$=$	$>$	$>$	$>$	$>$
F_{10}	$ADM_{10,12,13}$	$>$	N/A	$>$	$>$	$>$	$>$	$>$	$>$	$>$
F_{12}	$ADM_{10,12,13}$	$>$	N/A	$>$	$\geq$	$>$	$>$	$>$	$>$	$>$
F_{13}	$ADM_{10,12,13}$	$>$	N/A	$>$	$\geq$	$>$	$>$	$>$	$>$	$>$
F_{16}	$ADM_{16,19,23}$	$=$	$=$	N/A	$>$	$>$	$>$	$>$	$>$	$\geq$
F_{19}	$ADM_{16,19,23}$	$>$	$>$	N/A	$>$	$>$	$>$	$>$	$>$	$>$
F_{23}	$ADM_{16,19,23}$	$\geq$	$\geq$	N/A	$>$	$\geq$	$\geq$	$\geq$	$>$	$>$

Table 10 Borda counts for different $ADMs$ on F_g.

F_g	$B_g/B_{g,j,k}$	$ADM_{1,2,6}$	$ADM_{10,12,13}$	$ADM_{16,19,23}$	$\alpha = 1.0$	$\alpha = 1.2$	$\alpha = 1.4$	$\alpha = 1.6$	$\alpha = 1.8$	$\alpha = 2.0$
F_1	B_1	187	**125**	447	154	167	195	293	312	370
F_2	B_2	99	241	**95**	447	388	343	276	225	136
F_6	B_6	58	**50**	408	**50**	57	129	280	336	404
F_1 F_2 F_6	$B_{1,2,6}$	**344**	416	950	651	612	667	849	873	910
F_{10}	B_{10}	**72**	122	364	403	214	182	246	300	347
F_{12}	B_{12}	134	**105**	380	116	192	260	313	357	393
F_{13}	B_{13}	157	**93**	376	104	184	250	347	352	387
F_{10} F_{12} F_{13}	$B_{10,12,13}$	363	**320**	1120	623	590	692	906	1009	1127
F_{16}	B_{16}	55	55	**50**	267	247	228	222	221	130
F_{19}	B_{19}	110	140	**50**	354	344	345	340	312	245
F_{23}	B_{23}	184	207	**113**	346	305	270	286	270	269
F_{16} F_{19} F_{23}	$B_{16,19,23}$	349	402	**213**	967	896	843	848	803	644

Table 11 The means and standard deviations for different $ADMs$ tested on different function classes.

F_g		$ADM_{1,2,6}$	$ADM_{10,12,13}$	$ADM_{16,19,23}$	ADM_1	ADM_2	ADM_6	ADM_{10}	ADM_{12}	ADM_{13}	ADM_{16}	ADM_{19}	ADM_{23}
F_1	Mean	0.3711	0.3624	166.2	**0.3622**	1.5873	0.3951	0.6019	0.3630	0.3625	3553.7	17432.0	54819.6
	Std Dev	(3.11)	(3.19)	(883.8)	(3.19)	(3.43)	(3.18)	(3.25)	(3.19)	(3.19)	(2653.7)	(8861.4)	(12936.5)
F_2	Mean	0.040	0.072	0.043	0.085	**0.034**	0.383	0.035	0.136	0.081	15.48	44.78	94.83
	Std Dev	(4.40E-03)	(9.16E-03)	(9.59E-02)	(1.11E-02)	(7.12E-03)	(9.87E-02)	(5.42E-03)	(1.77E-02)	(1.15E-02)	(5.94)	(11.87)	(13.41)
F_6	Mean	0.50	**0.46**	729.1	**0.46**	1759.5	**0.46**	2.86	**0.46**	**0.46**	12656.2	33359.4	84031.8
	Std Dev	(3.41)	(3.41)	(1621.6)	(3.41)	(3887.0)	(3.41)	(5.42)	(3.41)	(3.41)	(7885.0)	(15600.2)	(22680.6)
F_{10}	Mean	**-20.04**	-19.24	-7.77	**-20.03**	-10.26	-0.47	**-19.67**	-12.04	-17.23	-3.22	-0.46	-0.74
	Std Dev	(0.31)	(3.86)	(5.93)	(0.31)	(3.11)	(2.78)	(0.85)	(9.72)	(6.92)	(1.82)	(0.32)	(0.48)
F_{12}	Mean	0.0148	0.0031	3.1889	0.0018	4.9436	0.0475	0.4014	**0.0025**	0.0031	79742.8	9159012.9	187046819.3
	Std Dev	(0.03)	(0.02)	(2.91)	(0.01)	(3.32)	(0.12)	(0.71)	(0.02)	(0.01)	(157289.1)	(12782055.1)	(78867738.0)
F_{13}	Mean	0.0936	**0.0042**	20.85	0.0172	26.99	0.0972	0.8932	**0.0036**	**0.0054**	17023.8	7383897.8	217114903.0
	Std Dev	(0.24)	(0.01)	(25.4)	(0.06)	(27.2)	(0.19)	(2.02)	(0.01)	(0.01)	(28202.4)	(10250182.6)	(80527486.2)
F_{16}	Mean	**-1.803489196**	**-1.803489196**	**-1.803489196**	-1.803489194	-1.803489194	-1.803489191	**-1.803489196**	-1.803489194	**-1.803489196**	**-1.803489195**	**-1.803489196**	-1.767501128
	Std Dev	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)
F_{19}	Mean	-5.23096752	-5.230967433	-5.230967576	-5.230967286	-5.22617391	-5.221829996	-5.230967563	-5.230794029	-5.230967504	-5.23096744	**-5.230967578**	-4.966909097
	Std Dev	(1.99)	(1.99)	(1.99)	(1.99)	(1.99)	(1.99)	(1.99)	(1.99)	(1.99)	(1.99)	(1.99)	(1.91)
F_{23}	Mean	-1.55E+07	-1.83E+07	-9.75E+07	-9.24E+06	-3.20E+07	-1.55E+07	-5.10E+07	-7.03E+06	-1.12E+07	-1.34E+07	-4.27E+08	**-3.86E+10**
	Std Dev	(4.22E+07)	(7.02E+07)	(3.26E+08)	(1.81E+07)	(1.54E+08)	(7.97E+07)	(1.67E+08)	(2.72E+07)	(4.11E+07)	(2.67E+07)	(9.05E+08)	(2.49E+11)

Table 12 Statistically significant (Wilcoxon) comparison of different $ADMs$.

F_g	$ADM_{g,j,k}$	$ADM_{1,2,6}$	$ADM_{10,12,13}$	$ADM_{16,19,23}$	ADM_1	ADM_2	ADM_6	ADM_{10}	ADM_{12}	ADM_{13}	ADM_{16}	ADM_{19}	ADM_{23}
F_1	$ADM_{1,2,6}$	N/A	$<$	$>$	$<$	$>$	$>$	$>$	$<$	$<$	$>$	$>$	$>$
F_2	$ADM_{1,2,6}$	N/A	$>$	$>$	$>$	$>$	$>$	$<$	$>$	$>$	$>$	$>$	$>$
F_6	$ADM_{1,2,6}$	N/A	$=$	$>$	$=$	$>$	$=$	$>$	$=$	$=$	$>$	$>$	$>$
F_{10}	$ADM_{10,12,13}$	$<$	N/A	$>$	$\leq$	$>$	$>$	$<$	$>$	$>$	$>$	$>$	$>$
F_{12}	$ADM_{10,12,13}$	$>$	N/A	$>$	$\leq$	$>$	$>$	$>$	$\leq$	$\leq$	$>$	$>$	$>$
F_{13}	$ADM_{10,12,13}$	$>$	N/A	$>$	$\geq$	$>$	$>$	$>$	$\leq$	$\geq$	$>$	$>$	$>$
F_{16}	$ADM_{16,19,23}$	$=$	$=$	N/A	$=$	$>$	$\geq$	$=$	$=$	$=$	$=$	$=$	$>$
F_{19}	$ADM_{16,19,23}$	$>$	$>$	N/A	$>$	$>$	$>$	$\geq$	$>$	$>$	$>$	$=$	$>$
F_{23}	$ADM_{16,19,23}$	$\geq$	$\geq$	N/A	$\geq$	$>$	$>$	$\geq$	$>$	$\geq$	$\geq$	$\leq$	$\leq$

Table 13 Borda Counts for different $ADMs$ on F_g.

F_g	$B_g/B_{g,j,k}$	$ADM_{1,2,6}$	$ADM_{10,12,13}$	$ADM_{16,19,23}$	ADM_1	ADM_2	ADM_6	ADM_{10}	ADM_{12}	ADM_{13}	ADM_{16}	ADM_{19}	ADM_{23}
F_1	B_1	202	**135**	445	141	397	314	298	183	**135**	468	548	600
F_2	B_2	172	263	**97**	318	116	449	134	398	303	501	549	600
F_6	B_6	60	**50**	416	**50**	432	**50**	260	**50**	**50**	451	547	598
F_1 F_2 F_6	$B_{1,2,6}$	**434**	448	958	509	945	813	692	631	488	1420	1644	1798
F_{10}	B_{10}	**96**	145	356	158	338	427	223	313	194	398	509	481
F_{12}	B_{12}	204	154	410	156	433	276	327	158	**132**	468	549	600
F_{13}	B_{13}	239	142	418	139	428	282	319	145	**138**	462	549	600
F_{10} F_{12} F_{13}	$B_{10,12,13}$	539	**441**	1184	453	1199	985	869	616	464	1328	1607	1681
F_{16}	B_{16}	65	66	53	101	117	210	53	96	53	73	**50**	567
F_{19}	B_{19}	243	313	111	371	393	491	147	426	243	320	**50**	583
F_{23}	B_{23}	331	362	229	370	342	459	273	396	352	305	**184**	308
F_{16} F_{19} F_{23}	$B_{16,19,23}$	639	741	393	842	852	1160	473	918	648	698	**284**	1458

Table 14 *ADM*s discovered by Genetic Programming.

ADM	Best ADM found by Genetic Programming
ADM_1	×(1.5994 ÷(÷(0.30352 −(N(1.6042,3.7606) N(-0.52902,1.423))) N(0.23588,0.95197)))
ADM_2	−(exp(1) abs(−(N(0.062744,0.62018) abs(2.7811))))
ADM_6	÷(÷(abs(N(1.7515,1.0711)) abs(N(-0.31211,3.1983))) N(1.3303,1.8317))
$ADM_{1,2,6}$	÷(÷(×(1.2807 *power*(0.421291))*N*(1.5056, 1.7563))*N*(−1.4591, 4.5805))
ADM_{10}	×(÷(−(abs(N(0.88065,1.2351)) N(0.98194,2.5068)) abs(exp(2.8935))) exp(−(N(-0.37126,2.3556) 0.36447)))
ADM_{12}	×(÷(N(1.9071,2.3711) +(÷(N(0.42263,3.6694) +(2.8295 N(-0.57212,4.419))) N(-0.14811,2.3614))) ÷(N(-1.3488,0.46446) +(2.6679 N(0.80107,4.4015))))
ADM_{13}	×(+(abs(÷(power(N(-0.14435,4.3745) 1.9102) +(+(N(-0.15717,3.4899) 2.2994) 1))) N(-1.9028, 2.9896)) ÷(exp(N(-1.38,1.0231)) −(0.24842 ×(N(0.30698,3.0673) exp(exp(N(0.12445,1.7122)))))))
$ADM_{10,12,13}$	÷(−(÷(−(*N*(1.1547, 1.1671)*N*(−0.11351, 0.017279))−(*N*(0.69177, 4.5311)*N*(−1.8809, 0.38345)))÷(÷(÷(*exp*(*N*(−0.87195, 0.80431))2.0028)−(*N*(−0.81015, 0.21294)*N*(−1.6028, 4.0077)))0.55601))*N*(−0.17811, 4.2532))
ADM_{16}	−(N(1.3936,0.31908) 1.4866)
ADM_{19}	×(−(1.6446 abs(N(-1.7989,0.29046))) exp(−(N(-0.65486,0.91397) 2.8009)))
ADM_{23}	÷(÷(2.9833 plus(1.4673,−(exp(exp(N(0.40255,2.1374))) ÷(power(0 N(-1.6585,2.4744)) N(1.8439, 4.2323))))) exp(0.45235))
$ADM_{16,19,23}$	÷(*power*(*abs*(0.18145)1.7633)*N*(0.26527, 4.8048))

Table 15 The mean and standard deviations of different mutation operators (ADM and Lévy distribution) on different function classes.

F_g		$ADM_{1,2,6}$	$ADM_{10,12,13}$	$ADM_{16,19,23}$	$\alpha = 1.0$	$\alpha = 1.2$	$\alpha = 1.4$	$\alpha = 1.6$	$\alpha = 1.8$	$\alpha = 2.0$
F_3	Mean	4255.7	3792.5	9640.8	**2828.6**	3241.0	2945.1	3431.5	3802.6	4519.7
	Std Dev	(1850.8)	(1581.5)	(4419.0)	(1179.3)	(1372.2)	(1486.6)	(1956.8)	(2293.6)	(2048.7)
F_4	Mean	38.09	34.55	52.12	**31.61**	34.66	34.88	38.81	39.12	42.97
	Std Dev	(9.97)	(7.62)	(10.86)	(7.91)	(7.44)	(7.84)	(8.41)	(9.18)	(7.12)
F_5	Mean	-6.50	-7.25	5.93	-7.61	**-8.19**	-6.37	-6.76	-5.37	-3.77
	Std Dev	(6.63)	(5.53)	(16.27)	(6.18)	(6.39)	(5.30)	(6.12)	(6.32)	(8.11)
F_7	Mean	0.0700	0.0584	0.1276	**0.0354**	0.0401	0.0519	0.0657	0.0723	0.1133
	Std Dev	(0.02)	(0.02)	(0.06)	(0.01)	(0.01)	(0.02)	(0.03)	(0.02)	(0.04)
F_8	Mean	-11205.1	**-11318.7**	-10484.1	-11111.0	-10503.4	-10089.0	-9664.8	-8949.3	-8072.7
	Std Dev	(289.4)	(358.7)	(414.2)	(444.4)	(479.1)	(518.7)	(597.4)	(589.9)	(742.4)
F_9	Mean	**-10.2267**	-10.2257	-10.1072	-10.2207	-10.2238	-10.1823	-10.0411	-9.1189	-7.1658
	Std Dev	(3.16)	(3.16)	(3.25)	(3.16)	(3.16)	(3.16)	(3.11)	(3.65)	(3.66)
F_{11}	Mean	0.0246	0.0039	10.66	**0.0011**	0.0169	0.0313	0.0402	0.0639	0.1567
	Std Dev	(0.12)	(0.01)	(18.61)	(0.003)	(0.03)	(0.07)	(0.06)	(0.15)	(0.30)
F_{14}	Mean	1.0662	0.8955	2.4611	**0.7973**	0.9778	1.0578	1.1140	1.0665	1.2546
	Std Dev	(1.17)	(0.56)	(2.28)	(0.38)	(0.53)	(0.79)	(0.70)	(0.62)	(0.75)
F_{15}	Mean	0.0453	0.0415	0.0465	**0.0411**	0.0426	0.0433	0.0426	0.0464	0.0446
	Std Dev	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)
F_{17}	Mean	**4.942067975**	**4.942067975**	**4.942067975**	4.942067978	4.942067978	4.942067979	4.942067977	4.942067978	4.942067977
	Std Dev	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)
F_{18}	Mean	4.510484905	4.510484905	**4.510484889**	4.510485582	4.510485593	4.510485625	4.510485455	4.510485479	4.510485176
	Std Dev	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)
F_{20}	Mean	-4.9265	-4.8417	**-4.9847**	-4.6685	-4.6149	-4.6951	-4.5936	-4.8405	-4.6868
	Std Dev	(1.97)	(2.06)	(1.98)	(2.09)	(2.11)	(2.09)	(2.17)	(2.04)	(2.15)
F_{21}	Mean	-6.03E+06	-5.29E+06	**-5.56E+07**	-4.75E+06	-2.89E+06	-6.24E+06	-2.23E+06	-5.84E+06	-4.30E+06
	Std Dev	(7.19E+06)	(9.35E+06)	(7.24E+07)	(2.17E+07)	(6.49E+06)	(1.99E+07)	(4.31E+06)	(1.35E+07)	(7.83E+06)
F_{22}	Mean	-4.48E+07	-6.60E+06	**-7.19E+07**	-3.29E+06	-4.46E+06	-3.32E+06	-1.04E+07	-4.06E+06	-4.29E+07
	Std Dev	(2.42E+08)	(9.44E+06)	(1.04E+08)	(7.30E+06)	(1.69E+07)	(4.22E+06)	(3.52E+07)	(1.10E+07)	(2.26E+08)